Clocks, communications, and correctness

DOI:
10.6100/IR405965

Document status and date:
Published: 01/01/1993

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Clocks, Communications, and Correctness

P. Zhou

Clocks, Communications, and Correctness

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof. dr. J.H. van Lint, voor een commissie aangewezen door het College van Dekanen in het openbaar te verdedigen op donderdag 2 december 1993 om 14.00 uur

door

PING ZHOU

geboren te Sichuan, CHINA

druk: wibro dissertatiedrukkerij, helmond.

Dit proefschrift is goedgekeurd
door de promotoren
prof. dr. W.-P. de Roever
prof. dr. J.C.M. Baeten

en de copromotor
dr. J.J.M. Hooman

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Zhou, P.

Clocks, communications, and correctness / P. Zhou.
- Eindhoven: Eindhoven University of Technology
Thesis Eindhoven. - With ref. - With summary in Dutch.
ISBN 90-386-0442-4
Subject headings: real time / fault tolerance / verification.

Acknowledgements

I would like to express my sincere appreciation to prof. Willem-Paul de Roever, my first promotor, who provided me with the opportunity to work at Eindhoven University of Technology. During the last years, his criticism, guidance, and encouragement have always stimulated my work. I am also very grateful to his wife, Corinne de Roever, who kindly helped me get used to my new life in the Netherlands.

Many thanks go to prof. Jos Baeten for his willingness to be my second promotor and for his constructive remarks on my thesis draft. Prof. Dieter Hammer is thanked for his practical view and suggestions concerning my work. I am specially grateful to prof. Chaochen Zhou for being a member of my promotion committee and for carefully reading my thesis draft. Prof. Flaviu Cristian is appreciated for his interest and detailed comments on a manuscript of chapter 4 of this thesis.

Dr. Ruurd Kuiper is thanked for helping me start my research work and for cooperating on a joint paper. It is my pleasure to thank dr. Jozef Hooman, my copromotor, who helped me produce this thesis, from the selection of topics to the presentation of my papers. My colleagues in the section of theoretical computer science are also thanked for their kindness and help in many aspects.

My deep appreciation goes to my parents and my elder brother without whose love and encouragement I could not have finished this thesis. Finally, I thank my husband, Haoran, who gives me such an enjoyable life and all the support I need during my work.

Contents

Chapter 1

Introduction

Computer systems are being used in a wide variety of real-time applications, such as: nuclear power plant control, industrial manufacturing control, medical monitoring, and flight systems. Such real-time systems are characterized by timing constraints relating occurrences of events. For instance, it is often required that an event is followed by another event in less than 7 time units, two consecutive occurrences of an event should be at least 3 time units apart, or a process should terminate by some deadline. Thus not only the functional but also the timing behavior of these systems is essential. Traditionally, the correctness of untimed computer systems is determined only by their logical and functional behavior. For real-time systems, their correctness depends on the temporal properties of their behavior as well.

Real-time systems are usually very complicated. It is not easy to guarantee that they will always meet their timing requirements. When failures occur, it is even more difficult to ensure that they will function correctly. Fault-tolerance techniques are often applied in real-time systems to ensure their correctness despite the presence of faults. All techniques for achieving fault-tolerance depend on the effective utilization of redundancy, that is, extra elements in the system which are redundant in the sense that they would not be required in a system which could be guaranteed to be free from faults [LA90]. However, the introduction of redundancy does influence the timing behavior of a system. For instance, the termination time of some process could be delayed and thus some deadline might not be met. Therefore real-time and fault-tolerance are closely related. Since there is hardly any existing theory for specifying and verifying real-time and fault-tolerant systems, it is a challenging problem to ensure the correctness of these systems.

In this thesis we investigate formalisms for specifying and verifying real-time and fault-tolerant systems and their applications. The rest of this introduction consists of four sections: in section 1.1 we explain the development of real-time formalisms, in section 1.2 we describe the specification and verification of real-time and fault-tolerant applications, in section 1.3 we discuss the notion of time, and in section 1.4 we give the structure of this thesis.

1.1 Real-Time Formalisms

1.1.1 Programming Language and Semantics

We start with a real-time programming language which is similar to Occam [Occ88]. This language is equipped with parallel composition and communication via message passing along channels, each of which is unidirectional and connects exactly two processes. A delay-statement is introduced to suspend the execution for some specified time. This statement may occur in the guard of a guarded command (similar to a delay-statement in the select-construct of Ada [Ada83]). We consider the following two versions of this language which differ in communication mechanisms.

In chapter 2, we study the first version in which communication is *synchronous*, i.e., a sender and a receiver both have to wait with communication until a communication partner is available. This version is similar to the CSP language in [Hoa85]. In contrast with this, we investigate in chapter 3 the second version of the programming language where communication is *asynchronous*, namely, a sender does not wait to synchronize with a receiver, but a receiver still has to wait for a message arriving if there are no messages in the buffer of a particular channel. It is assumed that all channels are capable of buffering an arbitrary number of messages. This is similar to the asynchronous communication mechanism defined in [JJH90].

Our aim is to develop a *compositional* proof system for the programming language. Compositionality enables us to derive the specification of a compound programming language construct from specifications of its constituent parts without any information about the internal structure of these parts [Ger84,Roe85]. A good starting point for a compositional proof system is a compositional semantics, i.e., the meaning of a process can be derived from the meanings of its components. Thus, for each of the two versions, the meaning of the programming language is defined by a compositional semantics. To achieve compositionality, the semantics of a process contains all possible computations of the process in any arbitrary environment, since the actual environment is not known in advance. Later, when we compose this process with some environment, impossible computations with respect to the given environment are excluded from the semantics of the composition of the process and this environment.

The two versions of the programming language have different models of computation, since they have different communication mechanisms. For both versions, their models describe for each process its states, i.e., mappings from variables to values, and its communication behavior, i.e., sending and receiving of messages. In particular, the model for the synchronous version also records when a process is waiting to send or to receive on a specific channel. This waiting information is needed to obtain a compositional semantics for this language. This is justified by the fact that this extra information appears in the fully abstract semantics for a similar language given in [HGR87]. For the asynchronous version, the model does not include waiting information of processes but contains explicit assumptions about the environment. This is consistent with [BH92] in which a fully abstract semantics for a similar language does not contain such waiting information.

In order to describe the real-time behavior of processes written in the programming language, we need to make assumptions about the execution time of statements. In general, there are two approaches to model the timing aspects of statements. One, taken for example in [NRSV90,BB91,HMP92], assumes that all statements except delays take zero time. The other, which is taken in this thesis as well as in timed CSP [RR86], assumes that every statement takes some amount of positive time. We will use parameters to represent the execution time of atomic statements and the time needed for the execution of compound statements. The correctness of a process with respect to a specification, which may express timing properties, is verified relative to these assumptions.

Another important assumption involves parallel composition. In this thesis, we use the *maximal parallelism* model [SM81,KSR$^+$88] to indicate that each parallel process runs at a distinct processor. Consequently, any action is executed as soon as possible without unnecessary waiting. Notice that maximal parallelism has different implications when applied to the two versions of the language. In the synchronous case, it implies that a process only waits when it tries to execute an input or output statement but the communication partner is not available. In the asynchronous case, maximal parallelism implies that a process only waits when it tries to receive a message along a channel for which the buffer is empty. This will be explained in chapters 2 and 3.

1.1.2 Specification

To express properties of real-time systems, a specification language is needed. As observed for example in [Lam83b], linear time temporal logic [Pnu77,MP82,OL82,MP91] is good for specifying and reasoning about untimed concurrent systems. This logic can express safety properties and liveness properties. Moreover, it supports reasoning in a simple and natural way. Unfortunately, this logic allows only the treatment of qualitative timing requirements, such as the demand that an event happens "eventually" or "always". To specify real-time properties, we have to extend temporal logic with a quantitative notion of time. Basically, there are two approaches.

In one approach, new temporal operators are introduced by extending the standard ones with time bounds. This extension of temporal logic is called Metric Temporal Logic (MTL). A typical timing property that "every event p is followed by another event q in less than 5 time units" can be expressed in MTL as

$$\Box(p \rightarrow \Diamond_{<5}\, q).$$

A general discussion about MTL and specification examples using MTL can be found in [Koy92]. This logic has been adopted to the specification of real-time properties of a transmission medium [KVR83]. Verification methods based on MTL for real-time transition systems can be found in [Har88,Hen91]. Compositional proof systems based on MTL for different versions of a programming language similar to the one studied in chapter 2 of this thesis have been formulated in [Hoo91].

In chapters 2 and 3 of this thesis, we investigate an alternative approach, called *Explicit Clock Temporal Logic (ECTL)*, in which temporal logic is extended with a distinguished time variable T that explicitly refers to the values of a global clock.

A similar logic, called RTTL (Real-Time Temporal Logic), has been used in [Ost89] to reason about real-time discrete event systems. There except the time variable, the universal quantifier is also allowed over global variables (i.e., variables whose values do not change over time). The above example can then be expressed in RTTL as

$$\forall x. \Box\,[(p \wedge T = x) \rightarrow \Diamond\,(q \wedge T < x + 5)].$$

Another extension appears in [PH88,Har88,HLP90], where it is referred to as GCTL (Global Clock Temporal Logic) and XCTL (Explicit Clock Temporal Logic), respectively. In addition to the time variable T, GCTL and XCTL also use global variables. But it is assumed that all global variables are universally quantified and thus no quantifier appears in any formula.

In [AH89] a logic called TPTL (Timed Propositional Temporal Logic) has been proposed. There global variables are also used and the explicit reference to the clock, i.e., the time variable, is replaced by a special freezing quantification. The freeze quantifer $x.$ binds the value of the clock to the quantified variable x. An extensive discussion about TPTL can be found in [Hen91]. The above example may be expressed in TPTL as

$$\Box\, x.[p \rightarrow \Diamond\, y.(q \wedge y < x + 5)],$$

which can be read as "in every state with time x, if p holds, then there is a later state with time y such that q holds and y is less than $x + 5$". A survey about the above mentioned extensions of linear time temporal logic can be found in [AH92].

This example is chosen to show the different ways of expression in those logics. Unfortunately, the ECTL presented in this thesis cannot express the example, since it does not contain global variables to record the values of the clock at different states. If the property is modified as "if p holds at the beginning of the execution, then q will hold in less than 5 time units", then it can be expressed in ECTL as

$$p \rightarrow \Diamond(q \wedge T < start + 5),$$

where *start* denotes the starting time of the execution. In this thesis, we would like to use the ECTL-based specification language to characterize all the possible executions of a process. It turns out that global variables are not needed.

In correspondence with the two versions of the programming language, the specification language based on ECTL has also two versions. In chapter 2, we present its synchronous version which includes primitives $comm(c, vexp)$, $wait(c!)$, and $wait(c?)$, which mean, respectively, that a process is communicating with its partner along channel c with value $vexp$, a process is waiting to send a message along channel c, and a process is waiting to receive a message on channel c. In the asynchronous version of the specification language shown in chapter 3, to describe the communication behavior, it is sufficient to include primitives $send(c, vexp)$ and $receive(c, vexp)$, which denote that a process has finished with sending and receiving value $vexp$ along channel c, respectively.

After having used an ECTL-based specification language in chapters 2 and 3, it appears that it is not easy to specify a system by using ECTL. As we will see in chapters 2 and 3, proving a simple process correct needs many steps of reasoning. In chapter 4, a fault-tolerant protocol presented in [CASD89] will be specified and verified. We would like to start with a simple specification language and to follow the informal proofs proposed in that paper. Therefore we adopt another specification language based on first-order logic. In the protocol, parallel processes are assumed to communicate asynchronously along communication links. The primitives for communication are $send(p, m, l)$ **at** t and $receive(p, m, l)$ **at** t, indicating, respectively, that processor p starts to send message m along link l at time t and p finishes with receiving m along l at time t.

1.1.3 Verification

To express that a process S satisfies a specification φ, we use a correctness formula of the form S **sat** φ. To verify that a system satisfies a specification, usually a proof system is used to derive the correctness formula. Such a proof system consists of axioms for atomic statements and rules for compound statements. Global proof systems, such as [MP82] for temporal logic, require the complete program text. In contrast with them, we formulate a compositional proof system to reduce the complexity of verification. Using a compositional proof system, we reason with specifications of processes instead of their program texts and thus abstract from their implementations. Such compositional proof systems have been developed for untimed systems, e.g. [Zwi89], and real-time systems, such as [Hoo91]. Other compositional theories can be found in [Lar90].

To verify compositionally that a system satisfies a requirement, there are generally two phases:

1. A system is decomposed into several smaller subsystems and, by using the specifications of these subsystems and an appropriate compositional proof system, we verify that the composition of these subsystems satisfies the the requirement of the system.

   This phase is performed repeatedly until it is possible to perform the second phase.

2. We implement these subsystems in some programming language and verify, by a proof system for this programming language, that the implementations indeed satisfy the specifications of those subsystems.

This approach is illustrated in chapter 2 by verifying a small part of an avionics system. The principle also guides us in verifying a fault-tolerant protocol in chapter 4.

For each of the two versions of the programming and specification languages, we formulate a compositional proof system. By examples we show how the proof systems can be used to reason about real-time properties. These two proof systems are shown to be sound with respect to the semantics (i.e., all correctness formulae derivable from the proof system are valid) and relatively complete [Bak80,Apt81] with respect to a proof system for ECTL (i.e., all valid correctness formulae can be derived from the proof system, provided all valid ECTL formulae are axioms of the proof system).

1.2 Real-Time and Fault-Tolerant Applications

For non-fault-tolerant systems, like the ones considered in chapters 2 and 3, it is implicitly assumed that all computing components are correct and remain correct during execution of these systems, i.e., these systems (including software and hardware) are free from faults. In reality, however, computer systems are composed of both hardware and software in which faults may exist and cause failures. A failure occurs when the behavior of the system deviates from its specification [RLT78]. In general, (software or hardware) faults are causes of failures and failures are manifestation of faults [LA90]. Such failures are taken into account in fault-tolerant systems.

In chapter 4, we study a formalism for specifying and verifying real-time and fault-tolerant systems and apply it to a protocol. A processor or link is correct if and only if it behaves as specified. Otherwise it suffers failures. We use primitives $correct(p)$ **at** t and $correct(l)$ **at** t to indicate, respectively, that processor p and link l are correct at time t. Typically for fault-tolerant systems, we also need to express the kind of failures which are considered when designing such systems (e.g. how much time it takes a spare generator to step in when electricity supply fails, in case of specifying a fault-tolerant electricity supply system for a hospital). Such assumptions about failures are called "failure assumptions" or "failure hypotheses".

Failures of components of a system can lead to unpredictable behavior and unavailability of service. To achieve a high reliability of a service in spite of failures, a key idea is to implement the service by replicating a server process on all processors in a network [Cri90]. A server process is a piece of software which fulfills the specific task. Given a network of distributed processors and replicated server processes, verifying that the service is indeed provided by the parallel execution of the server processes requires a *parallel composition rule*. With the assumption of maximal parallelism (i.e., each server process runs on its own processor), this rule states that parallel execution of server processes satisfies the conjunction of all server specifications, provided that each server specification only refers to the interface of the processor on which the server runs. Moreover, we need a *consequence rule* which enables us to weaken a specification and a *conjunction rule* which allows us to take the conjunction of specifications. To verify compositionally that the service is provided correctly, we follow the principle presented in section 1.1.3 and refine the first phase into four steps:

- First, the top-level requirement of the service should be described in some formal language. We call this description the *top-level specification*.

- Second, the general *system assumptions* should be axiomatized. For instance, the failure assumptions should be expressed and, when the service involves a lower level communication between processors and local clocks of processors, the communication mechanism and the clock synchronization assumptions should also be formalized.

- Third, the properties which the server process should satisfy must be characterized by a *server specification*. Such a server specification only refers to the interface of the processor on which the server is running. We assume that the server process running on processor p satisfies the server specification with parameter p.

  By the parallel composition rule, the parallel execution of the server processes satisfies the conjunction of the server specifications. Notice that the execution also satisfies the system assumptions formulated in step 2. Thus, by the conjunction rule, the execution satisfies the conjunction of the server specifications and the system assumptions. The next, and final, step is easy to formulate.

- Fourth, we prove that the conjunction of the server specifications and the system assumptions imply the top-level specification. Then, by the consequence rule, the parallel execution of the server processes satisfies the top-level specification.

After performing these steps, it remains to implement the server process such that the server specification is satisfied. This is, however, not done in this thesis and might be a topic for future work.

After this more theoretical research, we would like to apply the formal method to examples. As a starting point of verifying real-time and fault-tolerant systems, we choose a realistic application and apply the four steps of the compositional approach to it. Since atomic broadcast service is one of the fundamental issues in fault-tolerance, we selected an atomic broadcast protocol as our case study.

The atomic broadcast protocol is executed on a network of processors and links and is characterized by three properties [CASD89]: termination, atomicity, and order. These properties can be described as follows: if a correct processor broadcasts a message then all correct processors should receive this message by some time bound (termination), if a correct processor receives a message at some time then all correct processors should receive this message at more or less the same time (atomicity), and all correct processors should receive messages in the same ordering (order). This protocol is implemented by replicating a server process on all processors of the network. The parallel execution of these server processes should lead to the properties of the protocol.

In [CASD89] there is a series of protocols tolerating, respectively, omission failures, timing failures, and authentication-detectable byzantine failures. We chose a fairly simple protocol which tolerates omission failures. When a processor suffers an omission failure, it cannot send messages to other processors. When a link suffers an omission failure, the messages traveling along this link may be lost. But those messages received by a processor are correctly received in both timing and contents. In the network of processors, each processor has access to a local clock. It is assumed that local clocks of correct processors are synchronized within a certain bound.

This atomic broadcast protocol is called *synchronous* in [Cri90] in the sense that the underlying communication delay between correct processors is bounded. Other synchronous protocols can be found in, for instance, [BD85,Cri90]. There also exist *asynchronous* atomic broadcast protocols which do not assume bounded message transmission delay between correct processors. Examples of asynchronous protocols are [BJ87] and [CM84]. Also notice that, in the chosen synchronous atomic broadcast protocol for this thesis the underlying communication is asynchronous in the sense explained in section 1.1.1, i.e., a sender does not wait to synchronize with a receiver, and messages are buffered by links.

1.3 Notion of Time

In this thesis we assume maximal parallelism, i.e., each parallel process runs at its own processor. Notice that every processor has its own local clock. But, like many formalisms for real-time systems (e.g. see [BHRR91]), the timing behavior of a process is described in chapters 2 and 3 from the viewpoint of an external observer with his own clock, i.e., a global clock. Consequently, verification is done compositionally by using specifications in which timing is expressed by global clock values.

In chapter 4, we specify and verify an atomic broadcast protocol whose specification uses real time values as well as local clock values. Real time can be considered as a perfect, standard, global clock, e.g., Greenwich standard time. We have primitives like $send(p,m,l)$ **at** t, where t refers to real time. We use $C_p(t)$ to denote the local clock value of processor p at real time t. Using this notation, primitives written in terms of real time values can be transformed into abbreviations written in terms of local clock values. For instance, $send(p,m,l)$ $\mathbf{at_p}$ U, which intuitively means that processor p sends a message m along link l at local clock time U, is an abbreviation of $\exists u$: $(send(p,m,l)$ **at** $u \wedge C_p(u) = U)$, where u refers to some real time value and U refers to the corresponding local clock value on processor p. We will follow [CASD89] and specify the properties of the atomic broadcast protocol by using local clock values. We show that the verification of the protocol can be done compositionally by using specifications in which timing is expressed by local clock values.

In chapters 2 and 3, we assume a dense time domain called $TIME$ over which the values of a global clock range. In chapter 4, we have a dense time domain called $RTIME$ over which all real time values range. Furthermore, there exists a discrete time domain called $CVAL$ which contains all local clock values.

Comparing our notion of time with that in MTL, we make the following observations. In chapters 2 and 3, ECTL is the basis of our specification language and thus we can use absolute time in the sense that time points in a specification refer directly to actual global clock values. For instance, the property that in less than 8 time units after the start of execution, process S communicates with value 7 on channel d is expressed as follows:

$$S \ \mathbf{sat} \ \Diamond [T < start + 8 \wedge comm(d,7)].$$

In chapter 4, we also use absolute time and it can refer to both local clock values and real time values.

In the framework of MTL, a specification can only use relative time in the sense that time points in the specification are relative to some fixed time point. The example above can be described in MTL-style by

$$S \ \mathbf{sat} \ \Diamond_{<8} \ comm(d, 7).$$

Here the time points are relative to the starting point of the execution of S.

The primitives from the specification language in chapters 2 and 3 do not refer to the time at which an action is happening. For example, in the specification language in chapter 2, we have primitive $comm(c, vexp)$. The time when the communication occurs is implicit in this primitive and it should be obtained from the context. For instance, from formula $\Box(T = 5 \rightarrow comm(c, vexp))$, we know that this communication will happen when the global clock reaches 5. On the other hand, the primitives from the specification language in chapter 4 do explicitly refer to the time. For example, primitive $send(p, m, l)$ **at** t indicates clearly that processor p starts to send message m along link l at real time t. It appears in chapter 4 that referring to the time in the primitives makes the specification and verification of the protocol easier, since the primitives have already provided the timing information and thus we do not bother ourselves with the precise interpretation of the specification language.

1.4 Overview

The remainder of this thesis is structured as follows.

In chapter 2, we follow the outline of [Hoo91] and develop a formalism for specifying and verifying synchronously communicating real-time systems. The synchronous version of the programming language is described in section 2.1. A compositional semantics for this version of the language can be found in section 2.2. The synchronous version of the specification language based on ECTL is formulated in section 2.3. Section 2.4 contains a compositional proof system for the synchronous version of the programming and specification languages. This formalism is applied to specify and verify a small part of an avionics system in section 2.5. Soundness and relative completeness of this proof system are discussed in section 2.6. The proof system and the full version of this chapter are published in [HKZ91] and [ZHK93], respectively, which are joint work with J. Hooman and R. Kuiper.

In chapter 3, we present the asynchronous version of the formalism. The asynchronous version of the programming language is given in section 3.1. A compositional semantics for this version of the language is defined in section 3.2. The asynchronous version of the specification language based on ECTL is described in section 3.3. A compositional proof system for this asynchronous version of the programming and specification languages is proposed in section 3.4. The soundness and relative completeness issues are discussed in section 3.5. Most of the results in this chapter appear in [ZH92].

In chapter 4, we start with an introduction about the specification and verification of the atomic broadcast protocol in section 4.1. The top-level specification of the atomic broadcast service is described in section 4.2. The general system assumptions are axiomatized in section 4.3. The properties of the server process are expressed in section 4.4. In sections 4.5, 4.6, and 4.7, we verify that the parallel execution of the server processes leads to the desired top-level specification. Then we compare our results with [CASD89] in section 4.8. The primary results of this chapter appear in [ZH93b]. A full version of this chapter can be found in [ZH93a].

In chapter 5, we summarize our work and mention some related research.

Appendix A contains proofs of lemmas in chapter 2. Soundness and relative completeness of the proof system in chapter 2 are proved in Appendices B and C, respectively. Proofs of some lemmas in chapter 3 appear in Appendix D. Soundness proofs of a few modified axioms and rules of the proof system in chapter 3 can be found in Appendix E. Precise specifications for the statements of the programming language in chapter 3 are shown in Appendix F.

Chapter 2

Synchronous Communication

In this chapter, we investigate a theory for proving the correctness of synchronously communicating real-time systems. In section 2.1, we present the synchronous version of our real-time programming language in which parallel processes communicate via synchronous message passing. A compositional semantics of this language is defined in section 2.2. The synchronous version of our specification language is given in section 2.3. A compositional proof system is developed in section 2.4. An application of the proof theory is shown in section 2.5. Soundness and completeness of the proof system are discussed in section 2.6.

2.1 Real-Time Programming Language

2.1.1 Syntax and Informal Semantics

We consider a real-time programming language which is akin to Occam [Occ88]. The language is based on a real-time extension of CSP with nested parallelism and synchronous message passing via channels [KSR$^+$88]. A real-time statement **delay** e is added which suspends the execution for e time units if e is not negative. Such a delay-statement may also occur in the guard of a guarded command. Processes communicate by message passing via unidirectional channels, each of which connects exactly two processes. Communication is synchronous in the sense that a sender or a receiver has to wait for communication until a communication partner is available.

Let VAR be a nonempty set of variables, $CHAN$ be a nonempty set of channel names, and VAL be a nonempty domain of values. Let $I\!N$ denotes the set of all natural numbers (including 0). The syntax of the real-time programming language is given in Table 2.1, with $c, c_i \in CHAN$, $x, x_i \in VAR$, $\vartheta \in VAL$, $n \in I\!N$, and $n \geq 1$.

Any statement in the programming language is called a *process*. A write-variable is a variable which occurs in an input statement or in the left hand side of an assignment. Let

Table 2.1: Syntax of the Programming Language in Chapter 2

Expression	$e ::=$	$\vartheta \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2$
Guard	$g ::=$	$e_1 = e_2 \mid e_1 < e_2 \mid \neg g \mid g_1 \vee g_2$
Statement	$S ::=$	$\mathbf{skip} \mid x := e \mid \mathbf{delay}\ e \mid c!e \mid c?x \mid$
		$S_1; S_2 \mid G \mid \star G \mid S_1 \| S_2$
Guarded Command	$G ::=$	$[[]_{i=1}^{n} g_i \to S_i] \mid [[]_{i=1}^{n} g_i; c_i?x_i \to S_i [] g_0; \mathbf{delay}\ e \to S_0]$

S be any statement. Define $var(S)$ as the set of variables occurring in S and $wvar(S)$ as the set of all write-variables in S. Obviously, $wvar(S) \subseteq var(S)$. The set of (directional) channels occurring in a statement S, denoted by $dch(S)$, is defined as the set containing all channels occurring in S together with all directional channels $c!$ and $c?$ occurring in S. For instance, $dch(c!5; d?y \| c?x) = \{c, c!, c?, d, d?\}$.

Informally, the statements have the following meanings.

Atomic statements

- **skip** terminates immediately.
- $x := e$ assigns the value of expression e to variable x.
- **delay** e suspends execution for e time units if the value of e is not negative. Otherwise it is equivalent to **skip**.
- $c!e$ sends the value of expression e on channel c as soon as a corresponding input statement is available. Since we assume synchronous communication, such an output statement is suspended until a parallel process executes an input statement of the form $c?x$.
- $c?x$ receives a value via channel c and assigns this value to variable x. Similar to the output statement, such an input statement has to wait for a corresponding output statement before a synchronous communication takes place.

Compound statements

- $S_1; S_2$ indicates sequential composition of S_1 and S_2.
- Guarded command $[[]_{i=1}^{n} g_i \to S_i]$ is executed as follows. If none of the g_i evaluates to true, then the command terminates after the evaluation of the guards. Otherwise, nondeterministically select one of the g_i which evaluate to true and execute the corresponding statement S_i.
- During an execution of guarded command $[[]_{i=1}^{n} g_i; c_i?x_i \to S_i [] g_0; \mathbf{delay}\ e \to S_0]$, first the guards g_i, for $i = 0, 1, \ldots, n$, are evaluated. Next,

- if none of the g_i evaluates to true, then the command terminates;
- if g_0 evaluates to true, e is positive, and at least one of the $c_i?x_i$ for which g_i evaluate to true can start reading messages in less than e time units, then one of the first possible $c_i?x_i$ and its corresponding S_i are executed;
- if g_0 evaluates to true and either e is not positive or none of the $c_i?x_i$ for which g_i are true can start reading in less than e time units, then S_0 is executed;
- if g_0 evaluates to false, then the command waits until one of the $c_i?x_i$ for which g_i are true can read messages. Then one of the first possible $c_i?x_i$ and its corresponding S_i are executed.

A guard g_i which is equivalent to true is often omitted in a guarded command.

Example 2.1.1 Observe that delay-values can be arbitrary expressions, for instance, $x := y; [d?x \rightarrow y := x \;[]\; \textbf{delay}\ x \rightarrow c!x]$, where the value of x in **delay** x is obtained from executing the assignment $x := y$. □

Example 2.1.2 By means of a guarded command, we can easily express a time-out. For instance, $[x > 0; c?y \rightarrow x := y \;[]\; \textbf{delay}\ 10 \rightarrow \textbf{skip}]$ informally means that if $x > 0$ and the input communication can take place in less than 10 time units then the assignment is executed, otherwise after 10 time units there is a time-out and **skip** is executed. □

Notice that the semantics of the guarded command G in this thesis differs from that of Dijkstra for the case that all the boolean guards are false [Dij76], where it is interpreted that the program aborts.

- $\star G$ indicates repeated execution of guarded command G as long as one of the guards is true. When none of the guards is true, $\star G$ terminates.
- $S_1 \| S_2$ indicates parallel execution of S_1 and S_2. No variable should occur in both S_1 and S_2, i.e., $var(S_1) \cap var(S_2) = \emptyset$.

Henceforth we use $\equiv$ to denote syntactic equality.

2.1.2 Basic Assumptions

In this chapter, we assume that there is no overhead for compound statements and a **delay** e statement takes exactly e time units if the value of e is not negative. Furthermore we assume given positive parameters K_a, K_c, and K_g such that every assignment takes K_a time units, each communication takes K_c time units, and the evaluation of the guards in a guarded command takes K_g time units. Notice that, to avoid an infinite loop in finite time, we assume $K_g > 0$. These assumptions can be extended to more general cases, for instance, assignment and communication take some time between a lower and an upper bounds, etc..

We also assume the *maximal parallelism* model for the execution of parallel composition, which means that each parallel process has its own processor. Therefore, a process only waits when it tries to execute an input or output statement and the communication partner is not available. Hence it is never the case that one process waits to perform $c!e$ and, simultaneously, another process waits to execute $c?x$.

2.2 Compositional Semantics

To formally define the meaning of a process, we give a compositional semantics for our programming language. In section 2.2.1 we define a model to describe the computation of processes. This semantic model is used in section 2.3 to interpret our specification language. In section 2.2.2 we give the compositional semantics which is used to define validity of correctness formulae, that is, to define formally when a process satisfies a specification. Finally, in section 2.2.3 we discuss some properties of the semantics.

2.2.1 Computational Model

In our semantics the timing behavior of a process is expressed from the viewpoint of an external observer with his own clock. Let this clock range over a time domain *TIME*. Thus, although parallel components of a system have their own, physical, local clocks, the observable behavior of a system is described in terms of a single, conceptual, global clock.

Assume $TIME = \{\tau \in I\!R \mid \tau \geq 0\}$, where $I\!R$ is the set of all reals. Thus the time domain is dense (a domain is dense if between every two points there exists a third point) and linearly ordered. The standard arithmetical operators $+$, $-$, $\times$, and $\leq$ are defined on $TIME$. To define the timing behavior of statement **delay** e, we have to relate expressions in the programming language to our time domain. Since we have assumed that **delay** e takes e time units if e is not negative, we also assume $\{\vartheta \in VAL \mid \vartheta \geq 0\} \subseteq TIME$.

Henceforth, we use $i, j, \ldots$ to denote nonnegative integers, and $\tau, \hat{\tau}, \tau_0, \ldots$ to denote values of *TIME*. For notational convenience, we use a special value ∞ with the usual properties, such as $\infty \notin TIME$ and for all $\tau \in TIME \cup \{\infty\}$: $\tau \leq \infty$, $\tau + \infty = \infty + \tau = \infty$, etc.

A computation of a process is represented by a mapping which assigns to each point of time during this computation a pair consisting of a state and a set of communication records. The state represents values of variables at that point of time. The communication records denote the state of affairs on the channels of the process. We use records of the form (c, ϑ) to indicate that a communication occurs along channel c with value ϑ. Moreover, the model includes additional information that shows which processes are waiting to send or waiting to receive messages on which channels at any given time. Using this information, the formalism enforces *minimal waiting* in our maximal parallelism model by requiring that no pair of processes is ever simultaneously waiting to send and waiting to receive, respectively, on a shared channel. The informal description above is formalized in the following definitions.

Definition 2.2.1 (States) The set of states *STATE* is defined as the set of mappings from *VAR* to *VAL*: $STATE = \{s \mid s : VAR \rightarrow VAL\}$.

Thus a state $s \in STATE$ assigns to each variable x a value $s(x)$.

Definition 2.2.2 (Variant) The *variant* of a state s with respect to a variable x and a value ϑ, denoted by $(s : x \mapsto \vartheta)$, is defined as $(s : x \mapsto \vartheta)(y) = \begin{cases} \vartheta & \text{if } y \equiv x \\ s(y) & \text{if } y \not\equiv x \end{cases}$

Definition 2.2.3 (Communication Records) The set of communication records *COMM* is defined as:
$COMM = \{c! \mid c \in CHAN\} \cup \{c? \mid c \in CHAN\} \cup \{(c, \vartheta) \mid c \in CHAN \text{ and } \vartheta \in VAL\}$.

Assume $\tau_0 \in TIME$ and $\tau_1 \in TIME \cup \{\infty\}$. If $\tau_1 \neq \infty$, let $[\tau_0, \tau_1]$ denote a closed interval of time points: $[\tau_0, \tau_1] = \{\tau \in TIME \mid \tau_0 \leq \tau \leq \tau_1\}$. If $\tau_1 = \infty$, then $[\tau_0, \tau_1]$ is the same as $[\tau_0, \infty)$ with $[\tau_0, \infty) = \{\tau \in TIME \mid \tau \geq \tau_0\}$. Similarly, $(\tau_0, \tau_1]$ denotes a left-open and right-closed interval: $(\tau_0, \tau_1] = \{\tau \mid \tau_0 < \tau \leq \tau_1\}$ and $[\tau_0, \tau_1)$ denotes a left-closed and right-open interval: $[\tau_0, \tau_1) = \{\tau \mid \tau_0 \leq \tau < \tau_1\}$. The closed intervals will be used in the definition of a model, since we would like to observe the state and communication behavior at the starting and terminating points of a process.

Then a model, representing a real-time computation of a process, is defined as follows:

Definition 2.2.4 (Model) Let $\tau_0 \in TIME$, $\tau_1 \in TIME \cup \{\infty\}$, and $\tau_1 \geq \tau_0$. A *model* σ is a mapping $\sigma : [\tau_0, \tau_1] \rightarrow STATE \times \wp(COMM)$. Define $begin(\sigma) = \tau_0$ and $end(\sigma) = \tau_1$.

Consider a model σ and a point τ with $begin(\sigma) \leq \tau \leq end(\sigma)$. Then $\sigma(\tau) = (state, comm)$ with $state \in STATE$ and $comm \subseteq COMM$. Henceforth we refer to these two fields of $\sigma(\tau)$ by $\sigma(\tau).s$ and $\sigma(\tau).c$, respectively. Informally, if σ models a computation of a process S, $begin(\sigma)$ and $end(\sigma)$ denote, resp., the starting and terminating times of the computation of S ($end(\sigma) = \infty$ if S does not terminate). Furthermore, $\sigma(begin(\sigma)).s$ specifies the initial state of the computation, and if $end(\sigma) < \infty$

then $\sigma(end(\sigma)).s$ gives the final state. We will use σ^b to denote $\sigma(begin(\sigma))$, and if $end(\sigma) < \infty$, σ^e to denote $\sigma(end(\sigma))$. In general, $\sigma(\tau).s$ represents the values of variables. The set $\sigma(\tau).c$ might contain a communication record (c, ϑ), $c!$, or $c?$ with the following meaning, where $c \in CHAN$:

- $(c, \vartheta) \in \sigma(\tau).c$ iff value ϑ is being transmitted along channel c at time τ;
- $c! \in \sigma(\tau).c$ iff S is waiting to send along channel c at time τ;
- $c? \in \sigma(\tau).c$ iff S is waiting to receive along channel c at time τ.

To make the model convenient for sequential composition, the c-field at the last point is not used and then can have an arbitrary value. Only $\sigma^e.s$ is interesting for the specification and reasoning.

Define $DCHAN = CHAN \cup \{c? \mid c \in CHAN\} \cup \{c! \mid c \in CHAN\}$. Henceforth, we need the following definitions.

Definition 2.2.5 (Channels Occurring in a Model) The set of (directional) channels occurring in a model σ, denoted by $dch(\sigma)$, is defined as

$$\begin{aligned} dch(\sigma) = \ \bigcup_{begin(\sigma)\leq\tau<end(\sigma)} \ & \{c! \mid c! \in \sigma(\tau).c\} \cup \{c? \mid c? \in \sigma(\tau).c\} \cup \\ & \{c \mid \text{there exists a } \vartheta \text{ such that } (c, \vartheta) \in \sigma(\tau).c\} \end{aligned}$$

Definition 2.2.6 (Projection onto Channels) Let $cset \subseteq DCHAN$. Define the projection of a model σ onto $cset$, denoted by $[\sigma]_{cset}$, as follows: $begin([\sigma]_{cset}) = begin(\sigma)$, $end([\sigma]_{cset}) = end(\sigma)$, for any τ, $begin(\sigma) \leq \tau \leq end(\sigma)$, $[\sigma]_{cset}(\tau).s = \sigma(\tau).s$, and for any τ', $begin(\sigma) \leq \tau' < end(\sigma)$,

$$\begin{aligned} [\sigma]_{cset}(\tau').c \ = \ & \{c! \mid c! \in \sigma(\tau').c \wedge c! \in cset\} \cup \{c? \mid c? \in \sigma(\tau').c \wedge c? \in cset\} \cup \\ & \{(c, \vartheta) \mid (c, \vartheta) \in \sigma(\tau').c \wedge c \in cset\} \end{aligned}$$

Definition 2.2.7 (Projection onto Variables) Let $vset \subseteq VAR$. Define the projection of a model σ onto $vset$, denoted by $\sigma \downarrow vset$, as follows: $begin(\sigma \downarrow vset) = begin(\sigma)$, $end(\sigma \downarrow vset) = end(\sigma)$, for any τ, $begin(\sigma) \leq \tau < end(\sigma)$, $(\sigma \downarrow vset)(\tau).c = \sigma(\tau).c$, and for any τ', $begin(\sigma) \leq \tau' \leq end(\sigma)$, and any $x \in VAR$,

$$(\sigma \downarrow vset)(\tau').s(x) = \begin{cases} \sigma(\tau').s(x) & x \in vset \\ \sigma^b.s(x) & x \notin vset \end{cases}$$

Definition 2.2.8 (Concatenation) The *concatenation* of two models σ_1 and σ_2, denoted by $\sigma_1\sigma_2$, is a model σ such that

- if $end(\sigma_1) = \infty$, then $\sigma = \sigma_1$;

- if $end(\sigma_1) < \infty$, $end(\sigma_1) = begin(\sigma_2)$, and $\sigma_1^e.s = \sigma_2^b.s$, then σ has domain $[begin(\sigma_1), end(\sigma_2)]$ and is defined as follows:
$$\sigma(\tau) = \begin{cases} \sigma_1(\tau) & begin(\sigma_1) \leq \tau < end(\sigma_1) \\ \sigma_2(\tau) & begin(\sigma_2) \leq \tau \leq end(\sigma_2) \end{cases}$$
- otherwise undefined.

Definition 2.2.9 (Concatenation of Sets of Models) The *concatenation* of two sets of models Σ_1 and Σ_2 are defined as follows:

$SEQ(\Sigma_1, \Sigma_2) = \{\sigma_1\sigma_2 \mid \sigma_1 \in \Sigma_1 \text{ and } \sigma_2 \in \Sigma_2 \text{ such that } \sigma_1\sigma_2 \text{ is defined}\}$

It is easy to see that SEQ is associative, i.e.,
$SEQ(\Sigma_1, SEQ(\Sigma_2, \Sigma_3)) = SEQ(SEQ(\Sigma_1, \Sigma_2), \Sigma_3)$.
Henceforth we use $SEQ(\Sigma_1, \Sigma_2, \Sigma_3)$ to denote $SEQ(\Sigma_1, SEQ(\Sigma_2, \Sigma_3))$.

2.2.2 Formal Semantics

A good starting point for the development of a compositional proof system is the formulation of a compositional semantics. In such a semantics the meaning of a statement must be defined without any information about the environment in which it will be placed. Hence, the semantics of a statement in isolation must characterize all potential computations of the statement. When composing this statement with (part of) its environment, the semantic operators must remove the computations that are no longer possible. To be able to select the correct computations from the semantics, any dependency of an execution on the environment must be made explicit in the semantic model.

The evaluation of an expression e, denoted by $\mathcal{E}(e)$, is a function $\mathcal{E}(e) : STATE \rightarrow VAL$ defined by induction on the structure of e as follows:

- $\mathcal{E}(\vartheta)(s) = \vartheta$
- $\mathcal{E}(x)(s) = s(x)$
- $\mathcal{E}(e_1 + e_2)(s) = \mathcal{E}(e_1)(s) + \mathcal{E}(e_2)(s)$
- $\mathcal{E}(e_1 - e_2)(s) = \mathcal{E}(e_1)(s) - \mathcal{E}(e_2)(s)$
- $\mathcal{E}(e_1 \times e_2)(s) = \mathcal{E}(e_1)(s) \times \mathcal{E}(e_2)(s)$

The evaluation of a guard g, denoted by $\mathcal{G}(g)(s)$, is defined by induction on the structure of g as follows:

- $\mathcal{G}(e_1 = e_2)(s)$ iff $\mathcal{E}(e_1)(s) = \mathcal{E}(e_2)(s)$

- $\mathcal{G}(e_1 < e_2)(s)$ iff $\mathcal{E}(e_1)(s) < \mathcal{E}(e_2)(s)$
- $\mathcal{G}(\neg g)(s)$ iff not $\mathcal{G}(g)(s)$
- $\mathcal{G}(g_1 \vee g_2)(s)$ iff $\mathcal{G}(g_1)(s)$ or $\mathcal{G}(g_2)(s)$

The meaning of a process S, denoted by $\mathcal{M}(S)$, is a set of models representing all possible computations of S starting at any arbitrary time.

Skip

Statement **skip** terminates immediately without any state change or communication.
$\mathcal{M}(\mathbf{skip}) = \{\sigma \mid \mathit{begin}(\sigma) = \mathit{end}(\sigma)\ \}$

Assignment

An assignment $x := e$ terminates after K_a time units (recall that every assignment statement takes K_a time units to execute). All intermediate states before termination are the same as the initial state. The state at termination also equals the initial state except that the value of x is replaced by the value of e at the initial state. The c-field is empty during the execution period since the assignment does not (try to) communicate.
$\mathcal{M}(x := e) = \{\sigma \mid \mathit{end}(\sigma) = \mathit{begin}(\sigma) + K_a$, for any τ, $\mathit{begin}(\sigma) \leq \tau < \mathit{end}(\sigma)$,
$\sigma(\tau).s = \sigma^b.s$, $\sigma(\tau).c = \emptyset$, and $\sigma^e.s = (\sigma^b.s : x \mapsto \mathcal{E}(e)(\sigma^b.s))\}$

Delay

A **delay** e statement terminates after e time units if e is not negative. Otherwise it terminates immediately.
$\mathcal{M}(\mathbf{delay}\ e) = \{\sigma \mid \mathit{end}(\sigma) = \mathit{begin}(\sigma) + \mathit{max}(0, \mathcal{E}(e)(\sigma^b.s))$, for any τ,
$\mathit{begin}(\sigma) \leq \tau < \mathit{end}(\sigma)$, $\sigma(\tau).s = \sigma^b.s$, $\sigma(\tau).c = \emptyset$, and $\sigma^e.s = \sigma^b.s\}$

Output

In general, in the execution of an input or output statement, there are two periods: first there is a waiting period during which no communication partner is available (recall that communication is synchronous) and, secondly, when such a partner is available to communicate, there is a period (of K_c time units) during which the actual communication takes place. For an output statement $c!e$ these two periods are represented by two sets of models $\mathit{Wait}(c!)$ and $\mathit{Send}(c, e)$ defined below. Hence the semantics of $c!e$ is defined as

$\mathcal{M}(c!e) = \mathit{SEQ}(\mathit{Wait}(c!), \mathit{Send}(c, e))$ with

$Wait(c!) = \{\sigma \mid \text{for any } \tau,\ begin(\sigma) \leq \tau < end(\sigma),\ \sigma(\tau).s = \sigma^b.s,\ \sigma(\tau).c = \{c!\}, \text{ and}$
$\text{if } end(\sigma) < \infty, \text{ then } \sigma^e.s = \sigma^b.s\}$

$Send(c, e) = \{\sigma \mid end(\sigma) = begin(\sigma) + K_c, \text{ for any } \tau,\ begin(\sigma) \leq \tau < end(\sigma),$
$\sigma(\tau).s = \sigma^b.s,\ \sigma(\tau).c = \{(c, \mathcal{E}(e)(\sigma^b.s))\}, \text{ and } \sigma^e.s = \sigma^b.s\}$

Input

To represent all potential computations of an input statement $c?x$, its semantics should contain all possible models in which any possible value can be received for x. The value of x at the final state equals to the value in the communication record. Thus the semantics of $c?x$ is defined as

$\mathcal{M}(c?x) = SEQ(Wait(c?), Receive(c, x)),$

where $Wait(c?)$ is similar to $Wait(c!)$ and

$Receive(c, x) = \{\sigma \mid end(\sigma) = begin(\sigma) + K_c, \text{ there exists a } \vartheta \in VAL \text{ such that},$
$\text{for any } \tau,\ begin(\sigma) \leq \tau < end(\sigma),\ \sigma(\tau).s = \sigma^b.s,\ \sigma(\tau).c = \{(c, \vartheta)\},$
$\text{and } \sigma^e.s = (\sigma^b.s : x \mapsto \vartheta)\}$

Sequential Composition

Using the SEQ operator defined before, sequential composition is straightforward:

$\mathcal{M}(S_1; S_2) = SEQ(\mathcal{M}(S_1), \mathcal{M}(S_2))$

Since SEQ is associative, sequential composition is also associative. Thus we can write $S_1; S_2; S_3$ without causing ambiguity.

Guarded Command

For a guarded command G, first define

$$\bar{g} \equiv \begin{cases} \bigvee_{i=1}^{n} g_i & \text{if } G \equiv [\,[\!]_{i=1}^{n} g_i \rightarrow S_i] \\ \bigvee_{i=0}^{n} g_i & \text{if } G \equiv [\,[\!]_{i=1}^{n} g_i; c_i?x_i \rightarrow S_i \,[\!]\, g_0; \textbf{delay}\ e \rightarrow S_0] \end{cases}$$

Consider $G \equiv [\,[\!]_{i=1}^{n} g_i \rightarrow S_i]$. There are two possibilities: either none of the guards evaluates to true and the command terminates after K_g time units, or at least one of the guards yields true and then the corresponding statement S_i is executed. Recall that the evaluation of the guards takes K_g time units. In the semantics below this is represented by statement **delay** K_g.

$\mathcal{M}([\,[\!]_{i=1}^{n} g_i \rightarrow S_i]) = \{\sigma \mid \mathcal{G}(\neg \bar{g})(\sigma^b.s) \text{ and } \sigma \in \mathcal{M}(\textbf{delay}\ K_g)\} \cup$
$\{\sigma \mid \text{there exists a } k,\ 1 \leq k \leq n, \text{ such that } \mathcal{G}(g_k)(\sigma^b.s)$

$$\text{and } \sigma \in \mathcal{M}(\textbf{delay } K_g; S_k)\}$$

Next consider $G \equiv [[]_{i=1}^{n} g_i; c_i?x_i \rightarrow S_i [] g_0; \textbf{delay}\, e \rightarrow S_0]$.
There are four possibilities for an execution of G (see section 2.1). We first define two abbreviations:

$$\begin{aligned} Wait(G) = \{\sigma \mid\ & \mathcal{G}(\bar{g})(\sigma^b.s), \text{ for any } \tau,\ begin(\sigma) \leq \tau < end(\sigma),\ \sigma(\tau).s = \sigma^b.s, \\ & \sigma(\tau).c = \{c_i? \mid \mathcal{G}(g_i)(\sigma^b.s), 1 \leq i \leq n\ \}, \text{ and if } end(\sigma) < \infty \text{ then } \sigma^e.s = \sigma^b.s\} \end{aligned}$$

$$\begin{aligned} Comm(G) = \{\sigma \mid & \text{ there exists a } k,\ 1 \leq k \leq n, \text{ such that } \mathcal{G}(g_k)(\sigma^b.s) \text{ and} \\ & \sigma \in SEQ(Receive(c_k, x_k), \mathcal{M}(S_k))\ \} \end{aligned}$$

Using $Wait(G)$, we define the following extra abbreviations:

$$\begin{aligned} FinWait(G) = \{\sigma \mid\ & \mathcal{G}(g_0)(\sigma^b.s),\ end(\sigma) < begin(\sigma) + max(0, \mathcal{E}(e)(\sigma^b.s)), \text{ and} \\ & \sigma \in Wait(G)\ \} \end{aligned}$$

$$\begin{aligned} TimeOut(G) = \{\sigma \mid\ & \mathcal{G}(g_0)(\sigma^b.s),\ end(\sigma) = begin(\sigma) + max(0, \mathcal{E}(e)(\sigma^b.s)), \text{ and} \\ & \sigma \in Wait(G)\ \} \end{aligned}$$

$$AnyWait(G) = \{\sigma \mid \mathcal{G}(\neg g_0)(\sigma^b.s) \text{ and } \sigma \in Wait(G)\ \}$$

Then the semantics of G is defined as follows:

$$\begin{aligned} & \mathcal{M}([[]_{i=1}^{n} g_i; c_i?x_i \rightarrow S_i [] g_0; \textbf{delay}\, e \rightarrow S_0]) = \\ & \qquad \{\sigma \mid \mathcal{G}(\neg\bar{g})(\sigma^b.s) \text{ and } \sigma \in \mathcal{M}(\textbf{delay } K_g)\} \cup \\ & \qquad SEQ(\mathcal{M}(\textbf{delay } K_g), FinWait(G), Comm(G)) \cup \\ & \qquad SEQ(\mathcal{M}(\textbf{delay } K_g), TimeOut(G), \mathcal{M}(S_0)) \cup \\ & \qquad SEQ(\mathcal{M}(\textbf{delay } K_g), AnyWait(G), Comm(G)) \end{aligned}$$

Iteration

For a model in the semantics of the iteration statement $\star G$, we have the following possibilities:

- either it is the concatenation of a finite sequence of models from $\mathcal{M}(G)$ such that the last model corresponds to an execution where all guards evaluate to false or it represents a nonterminating computation of G,

- or it is the concatenation of an infinite sequence of models from $\mathcal{M}(G)$ that all represent terminating computations in which not all guards yield false.

This leads to the following definition:

$$\begin{aligned} \mathcal{M}(\star G) = \{\sigma \mid & \text{ there exist a } k \in I\!N, k \geq 1, \text{ and } \sigma_1, \ldots, \sigma_k \text{ such that } \sigma = \sigma_1 \cdots \sigma_k, \\ & \text{for any } i,\ 1 \leq i \leq k,\ \sigma_i \in \mathcal{M}(G), \text{ for any } j,\ 1 \leq j \leq k-1,\ end(\sigma_j) < \infty, \end{aligned}$$

$$\mathcal{G}(\bar{g})(\sigma_j^b.s), \text{ and if } end(\sigma_k) < \infty \text{ then } \mathcal{G}(\neg\bar{g})(\sigma_k^b.s) \text{ otherwise } \mathcal{G}(\bar{g})(\sigma_k^b.s) \}$$
$$\cup \{\sigma | \text{ there exists an infinite sequence of models } \sigma_1, \sigma_2, \ldots \text{ such that } \sigma = \sigma_1\sigma_2\cdots,$$
$$\text{for any } i \geq 1, \sigma_i \in \mathcal{M}(G), end(\sigma_i) < \infty, \text{ and } \mathcal{G}(\bar{g})(\sigma_i^b.s) \}$$

A slight apology should be made for the semantics of $\star G$. The semantics given above is not fully compositional, because it cannot be determined by the semantics of G alone. We still need to check if the guards of G are true.

Parallel Composition

The semantics of $S_1 \| S_2$ consists of all models σ such that there exist models $\sigma_1 \in \mathcal{M}(S_1)$ and $\sigma_2 \in \mathcal{M}(S_2)$ and the c-fields of σ is the point-wise union of the c-fields of σ_1 and σ_2, provided that the following requirements are fulfilled:

1. $end(\sigma) = max(end(\sigma_1), end(\sigma_2))$, to express that $S_1 \| S_2$ terminates when both processes have terminated.

2. Since communication is synchronous, S_1 and S_2 should communicate simultaneously on shared channels which connect them.

3. In our execution model we assume maximal parallelism and thus two processes should not be simultaneously waiting to send and waiting to receive on a shared channel. Formally, for any $c \in dch(S_1) \cap dch(S_2)$, and any τ, $begin(\sigma) \leq \tau < end(\sigma)$, we should have $\neg(c! \in \sigma(\tau).c \wedge c? \in \sigma(\tau).c)$.

For the s-fields of σ, recall that there are no shared variables, i.e., $var(S_1) \cap var(S_2) = \emptyset$. Hence the value of a variable x during the execution of $S_1 \| S_2$ can be obtained from the state of S_i if $x \in var(S_i)$, and from the initial state otherwise. This leads to the following definition for the semantics of parallel composition.

$$\begin{aligned}
\mathcal{M}(S_1 \| S_2) = \{\sigma \mid\ & dch(\sigma) \subseteq dch(S_1) \cup dch(S_2), \text{ for } i = 1, 2, \text{ there exist } \sigma_i \in \mathcal{M}(S_i) \\
& \text{such that} \\
& begin(\sigma) = begin(\sigma_1) = begin(\sigma_2),\ end(\sigma) = max(end(\sigma_1), end(\sigma_2)), \\
& [\sigma]_{dch(S_i)}(\tau).c = \begin{cases} \sigma_i(\tau).c & begin(\sigma_i) \leq \tau < end(\sigma_i) \\ \emptyset & end(\sigma_i) \leq \tau < end(\sigma) \end{cases} \\
& (\sigma \downarrow var(S_i))(\tau).s = \begin{cases} \sigma_i(\tau).s & begin(\sigma_i) \leq \tau \leq end(\sigma_i) \\ \sigma_i^e.s & end(\sigma_i) < \tau \leq end(\sigma) \end{cases} \\
& \text{for any } x \notin var(S_1) \cup var(S_2), \text{ any } \tau,\ begin(\sigma) \leq \tau \leq end(\sigma), \\
& \sigma(\tau).s(x) = \sigma_i^b.s(x), \\
& \text{and for any } c \in dch(S_1) \cap dch(S_2), \text{ any } \tau,\ begin(\sigma) \leq \tau < end(\sigma), \\
& \neg(c! \in \sigma(\tau).c\ \wedge\ c? \in \sigma(\tau).c)\}
\end{aligned}$$

We can prove that parallel composition is commutative and associative.

2.2.3 Properties of the Semantics

First we define a well-formedness property of a model.

Definition 2.2.10 (Well-Formedness) A model σ, defined in section 2.2.1, is *well-formed* iff for any $c \in CHAN$, any $\vartheta, \vartheta_1, \vartheta_2 \in VAL$, and any $\tau, begin(\sigma) \leq \tau < end(\sigma)$, the following formulae hold:

1. $\neg(c! \in \sigma(\tau).c \wedge c? \in \sigma(\tau).c)$,
   (*Minimal waiting*: it is not allowed to be simultaneously waiting to send and waiting to receive on a particular channel.)

2. $\neg[(c, \vartheta) \in \sigma(\tau).c \wedge c! \in \sigma(\tau).c] \ \wedge \ \neg[(c, \vartheta) \in \sigma(\tau).c \wedge c? \in \sigma(\tau).c]$, and
   (*Exclusion*: it is not allowed to be simultaneously communicating and waiting to communicate on a given channel.)

3. $(c, \vartheta_1) \in \sigma(\tau).c \wedge (c, \vartheta_2) \in \sigma(\tau).c \rightarrow \vartheta_1 = \vartheta_2$.
   (*Uniqueness*: at most one value is transmitted on a particular channel at any point of time.)

Then we have the following theorem.

Theorem 2.2.1 For any process S, if $\sigma \in \mathcal{M}(S)$ then

1. $dch(\sigma) \subseteq dch(S)$,

2. if $x \notin wvar(S)$,then for any τ, $begin(\sigma) \leq \tau \leq end(\sigma)$, $\sigma(\tau).s(x) = \sigma^b.s(x)$, and

3. σ is well-formed.

By induction on the structure of S and the definition of well-formedness, this theorem can be easily proved.

2.3 Specification Language

We define a specification language which is based on Explicit Clock Temporal Logic, i.e., linear time temporal logic augmented with a global clock variable denoted by T. Intuitively, T refers to the current point of time during an execution. We use *start* and *term* to express, respectively, the starting and terminating times of a computation ($term = \infty$ for a nonterminating computation). We also use $first(x)$ and $last(x)$ to refer to the value of variable x at the first and the last state of a computational model, respectively. If the computation does not terminate, then $last(x)$ has the initial value of x. Similar ideas have been used in, for instance, [Jon80] and [Jon90]. To specify the communication behavior of processes, we use a primitive $comm(c, vexp)$ to express a communication along channel c with value $vexp$. We also use $comm(c)$ to abstract from the value communicated. Furthermore, the specification language includes primitives $wait(c!)$ and $wait(c?)$ to denote that processes are waiting to communicate. Similar to the semantics, this is required to express maximal parallelism. By including the strong until operator, $\mathcal{U}$, from classical temporal logic we obtain the standard modal operators. In order to give compositional proof rules for sequential composition and iteration, we add the "chop" operator $\mathcal{C}$ and the "iterated chop" operator $\mathcal{C}^*$ from [BKP84].

In the specification language, there are two kinds of expressions, i.e., $vexp$ and $texp$, to express values of type VAL and $TIME \cup \{\infty\}$, respectively. A specification is represented by φ. The syntax of this specification language is given in Table 2.2, with $\vartheta \in VAL$, $x \in VAR$, $\hat{\tau} \in TIME \cup \{\infty\}$, and $c \in CHAN$.

Table 2.2: Syntax of the Specification Language in Chapter 2

Val Exp	$vexp ::=$	$\vartheta \mid x \mid first(x) \mid last(x) \mid max(vexp_1, vexp_2) \mid$
		$vexp_1 + vexp_2 \mid vexp_1 - vexp_2 \mid vexp_1 \times vexp_2$
Time Exp	$texp ::=$	$\hat{\tau} \mid T \mid start \mid term \mid vexp \mid$
		$texp_1 + texp_2 \mid texp_1 - texp_2 \mid texp_1 \times texp_2$
Specification	$\varphi ::=$	$texp_1 = texp_2 \mid texp_1 < texp_2 \mid$
		$comm(c, vexp) \mid comm(c) \mid wait(c!) \mid wait(c?) \mid$
		$\varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2 \mid \varphi_1 \,\mathcal{C}\, \varphi_2 \mid \varphi_1 \,\mathcal{C}^*\, \varphi_2$

Let $texp$ be any expression of type $TIME$ from the specification language. Define $var(texp)$ to be the set of all variables occurring in $texp$. Let φ be any specification. Define $dch(\varphi)$ to be the set of all directional channels, i.e., the set of c, $c!$, or $c?$, for $c \in CHAN$, occurring in φ, and $var(\varphi)$ to be the set of all variables occurring in φ.

The interpretation of specifications is defined over the computational model of section 2.2.1. First we define the value of expression $vexp$ at model σ and time $\tau \geq begin(\sigma)$, $\tau \in TIME$, denoted by $\mathcal{V}(vexp)(\sigma, \tau)$, as follows:

- $\mathcal{V}(\vartheta)(\sigma, \tau) = \vartheta$
- $\mathcal{V}(x)(\sigma, \tau) = \begin{cases} \sigma(\tau).s(x) & \text{if } \tau \leq end(\sigma) \\ \sigma^e.s(x) & \text{if } \tau > end(\sigma) \end{cases}$
- $\mathcal{V}(first(x))(\sigma, \tau) = \sigma^b.s(x)$
- $\mathcal{V}(last(x))(\sigma, \tau) = \begin{cases} \sigma^e.s(x) & \text{if } end(\sigma) < \infty \\ \sigma^b.s(x) & \text{if } end(\sigma) = \infty \end{cases}$
- $\mathcal{V}(max(vexp_1, vexp_2))(\sigma, \tau) = max(\mathcal{V}(vexp_1)(\sigma, \tau), \mathcal{V}(vexp_2)(\sigma, \tau))$

- $\mathcal{V}(vexp_1 \odot vexp_2)(\sigma, \tau) = \mathcal{V}(vexp_1)(\sigma, \tau) \odot \mathcal{V}(vexp_2)(\sigma, \tau)$, for $\odot \in \{+, -, \times\}$.

Next we define the value of time expression $texp$ at model σ and time $\tau \geq begin(\sigma)$, $\tau \in TIME$, denoted by $\mathcal{T}(texp)(\sigma, \tau)$, as follows:

- $\mathcal{T}(\hat{\tau})(\sigma, \tau) = \hat{\tau}$
- $\mathcal{T}(T)(\sigma, \tau) = \tau$
- $\mathcal{T}(start)(\sigma, \tau) = begin(\sigma)$
- $\mathcal{T}(term)(\sigma, \tau) = end(\sigma)$
- $\mathcal{T}(vexp)(\sigma, \tau) = \mathcal{V}(vexp)(\sigma, \tau)$
- $\mathcal{T}(texp_1 \odot texp_2)(\sigma, \tau) = \mathcal{T}(texp_1)(\sigma, \tau) \odot \mathcal{T}(texp_2)(\sigma, \tau)$, for $\odot \in \{+, -, \times\}$.

The interpretation of a specification φ at model σ and time $\tau \geq begin(\sigma)$, $\tau \in TIME$, is denoted by $\langle \sigma, \tau \rangle \models \varphi$ and defined by induction on the structure of φ.

- $\langle \sigma, \tau \rangle \models texp_1 = texp_2$ iff $\mathcal{T}(texp_1)(\sigma, \tau) = \mathcal{T}(texp_2)(\sigma, \tau)$.
- $\langle \sigma, \tau \rangle \models texp_1 < texp_2$ iff $\mathcal{T}(texp_1)(\sigma, \tau) < \mathcal{T}(texp_2)(\sigma, \tau)$.
- $\langle \sigma, \tau \rangle \models comm(c, vexp)$ iff $\tau < end(\sigma)$ and $(c, \mathcal{V}(vexp)(\sigma, \tau)) \in \sigma(\tau).c$.
- $\langle \sigma, \tau \rangle \models comm(c)$ iff $\tau < end(\sigma)$ and there exists a value $\vartheta \in VAL$ such that $(c, \vartheta) \in \sigma(\tau).c$.
- $\langle \sigma, \tau \rangle \models wait(c!)$ iff $\tau < end(\sigma)$ and $c! \in \sigma(\tau).c$.
- $\langle \sigma, \tau \rangle \models wait(c?)$ iff $\tau < end(\sigma)$ and $c? \in \sigma(\tau).c$.
- $\langle \sigma, \tau \rangle \models \varphi_1 \vee \varphi_2$ iff $\langle \sigma, \tau \rangle \models \varphi_1$ or $\langle \sigma, \tau \rangle \models \varphi_2$.
- $\langle \sigma, \tau \rangle \models \neg\varphi$ iff not $\langle \sigma, \tau \rangle \models \varphi$.
- $\langle \sigma, \tau \rangle \models \varphi_1 \; \mathcal{U} \; \varphi_2$ iff there exists a $\tau_2 \geq \tau$, such that $\langle \sigma, \tau_2 \rangle \models \varphi_2$, and for any $\tau_1, \tau \leq \tau_1 < \tau_2$, $\langle \sigma, \tau_1 \rangle \models \varphi_1$.
- $\langle \sigma, \tau \rangle \models \varphi_1 \; \mathcal{C} \; \varphi_2$ iff
  - either $\langle \sigma, \tau \rangle \models \varphi_1$ and $end(\sigma) = \infty$
  - or there exist models σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $\tau \leq end(\sigma_1) < \infty$, $\langle \sigma_1, \tau \rangle \models \varphi_1$, and $\langle \sigma_2, begin(\sigma_2) \rangle \models \varphi_2$.
- $\langle \sigma, \tau \rangle \models \varphi_1 \; \mathcal{C}^* \; \varphi_2$ iff

- either there exist a $k \geq 1$ and models $\sigma_1, \ldots, \sigma_k$ such that $\sigma = \sigma_1 \cdots \sigma_k$, $\langle \sigma_1, \tau \rangle \models \varphi_1$, $\tau \leq end(\sigma_1) < \infty$, for any j, $2 \leq j \leq k-1$, $\langle \sigma_j, begin(\sigma_j) \rangle \models \varphi_1$, $end(\sigma_j) < \infty$, and if $end(\sigma_k) < \infty$ then $\langle \sigma_k, begin(\sigma_k) \rangle \models \varphi_2$, otherwise $\langle \sigma_k, begin(\sigma_k) \rangle \models \varphi_1$,
- or there exist infinitely many models $\sigma_1, \sigma_2, \sigma_3, \ldots$ such that $\sigma = \sigma_1 \sigma_2 \sigma_3 \ldots$, $end(\sigma_1) \geq \tau$, for any $i \geq 1$, $end(\sigma_i) < \infty$, $\langle \sigma_1, \tau \rangle \models \varphi_1$, and for any $j \geq 2$, $\langle \sigma_j, begin(\sigma_j) \rangle \models \varphi_1$.

The substitution of an expression $vexp_1$ for a variable x in an expression $vexp_2$, denoted by $vexp_2[vexp_1/x]$, is defined as the expression obtained by replacing every occurrence of x in $vexp_2$ by $vexp_1$. This notation will be used in the axiom for assignment statement.

We also use the standard abbreviations such as $true \equiv 0 = 0$, $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$, $texp_1 \leq texp_2 \equiv (texp_1 = texp_2) \vee (texp_1 < texp_2)$, etc..

Furthermore we have the usual abbreviations from temporal logic:

- $\Diamond\varphi \equiv true \; \mathcal{U} \; \varphi$ (eventually φ will be true)
- $\Box\varphi \equiv \neg\Diamond\neg\varphi$ (henceforth φ will be true)
- $\varphi_1 \; \mathbf{U} \; \varphi_2 \equiv (\varphi_1 \; \mathcal{U} \; \varphi_2) \vee \Box\varphi_1$ (weak until: either eventually φ_2 will hold and until that point φ_1 holds continuously, or φ_1 holds henceforth)

Next we define validity of specifications and correctness formulae of the form S **sat** φ.

Definition 2.3.1 (Valid Specification) A specification φ is *valid*, denoted by $\models \varphi$, iff $\langle \sigma, begin(\sigma) \rangle \models \varphi$ for any model σ.

For instance, $\models T = start$, $\models x = first(x)$, and
$\models term < \infty \wedge \Box(T = term \rightarrow x = 5) \rightarrow last(x) = 5$.

Definition 2.3.2 (Satisfaction) A process S *satisfies* a specification φ, denoted by $\models S$ **sat** φ, iff $\langle \sigma, begin(\sigma) \rangle \models \varphi$ for any $\sigma \in \mathcal{M}(S)$.

We also say that S **sat** φ hlods if $\models S$ **sat** φ.

We give a few simple examples to illustrate our specification language. General safety properties can be specified, e.g.,

- Process S does not terminate: S **sat** $term = \infty$.
  Note that we could also use S **sat** $\Box\neg(T = term)$.
- S does not perform any communication along channel c: S **sat** $\Box\neg comm(c)$.

Some examples of real-time safety properties:

- If S starts its execution with $x = 0$, then S will terminate in less than 5 time units with $x = 8$:
  S **sat** $x = 0 \rightarrow (term < start + 5) \wedge (last(x) = 8)$.

- S waits to communicate on channel c and after communication on c it is waiting to send on channel d:
  S **sat** $(wait(c)\ \mathbf{U}\ (comm(c)\ \mathcal{U}\ T = term))\ \mathcal{C}\ \ wait(d!)$.

- During the execution of S, variable x has value 5 at 3 time units after the start of the execution, after 5 time units x has value 8 and y has value 9, and finally after 7 time units process S terminates with $x = 10$ and $y = 12$:
  S **sat** $\square\,[(T = start + 3 \rightarrow x = 5) \wedge (T = start + 5 \rightarrow x = 8 \wedge y = 9) \wedge$
  $(T = start + 7 \rightarrow x = 10 \wedge y = 12)] \wedge term = start + 7$.

Liveness properties can also be expressed:

- S terminates: S **sat** $term < \infty$. (Or, equivalently, S **sat** $\Diamond\,(T = term)$.)

- S either communicates along channel c infinitely often or eventually it waits forever to send on c: S **sat** $(\square\,\Diamond\, comm(c)) \ \vee\ (\Diamond\,\square\, wait(c!))$.

2.4 Proof System

In this section, we give a compositional proof system for the synchronous version of the programming and specification languages. This proof system will take all valid ECTL assertions as axioms. We start with axioms and rules which are generally applicable to any statement. Next we axiomatize the programming language by formulating axioms and rules for all atomic statements and compound programming constructs.
Let $vexp_1$ and $vexp_2$ be expressions of type VAL. The well-formedness property of the semantic models is axiomatized by the following axiom. For any finite $cset \subseteq DCHAN$,

Axiom 2.4.1 (Well-Formedness)
For any finite $cset \subseteq DCHAN$, S **sat** WF_{cset}, where

$$\begin{array}{lcl}
WF_{cset} & \equiv & \square\,(MinWait_{cset} \wedge Exclusion_{cset} \wedge Unique_{cset}) \\
MinWait_{cset} & \equiv & \bigwedge_{\{c!,c?\}\subseteq cset} \neg(wait(c!) \wedge wait(c?)) \\
Exclusion_{cset} & \equiv & \bigwedge_{\{c,c!\}\subseteq cset} \neg(comm(c) \wedge wait(c!)) \wedge \bigwedge_{\{c,c?\}\subseteq cset} \neg(comm(c) \wedge wait(c?)) \\
Unique_{cset} & \equiv & \bigwedge_{c\in cset} comm(c, vexp_1) \wedge comm(c, vexp_2) \rightarrow vexp_1 = vexp_2
\end{array}$$

For any finite $cset \subseteq DCHAN$ and $vset \subseteq VAR$, define
$empty(cset) \equiv \bigwedge_{c!\in cset} \neg wait(c!) \ \wedge\ \bigwedge_{c?\in cset} \neg wait(c?) \ \wedge\ \bigwedge_{c\in cset} \neg comm(c)$ and
$inv(vset) \equiv \bigwedge_{x\in vset} x = first(x)$.

The next general axiom expresses that a process does not (try to) communicate on channels that do not syntactically occur in the process.

Axiom 2.4.2 (Communication Invariance)
For any finite $cset \subseteq DCHAN$ with $cset \cap dch(S) = \emptyset$, S **sat** $\Box\, empty(cset)$.

Similarly, the proof system has an axiom to express that certain variables are not changed by a process.

Axiom 2.4.3 (Variable Invariance)
For any finite $vset \subseteq VAR$ with $vset \cap wvar(S) = \emptyset$, S **sat** $\Box\, inv(vset)$.

Furthermore, we have the usual conjunction rule and consequence rule.

Rule 2.4.1 (Conjunction) $$\frac{S \textbf{ sat } \varphi_1,\ S \textbf{ sat } \varphi_2}{S \textbf{ sat } \varphi_1 \wedge \varphi_2}$$

Rule 2.4.2 (Consequence) $$\frac{S \textbf{ sat } \varphi_1,\ \varphi_1 \rightarrow \varphi_2}{S \textbf{ sat } \varphi_2}$$

Next we give axioms for the five atomic statements. Statement **skip** terminates immediately.

Axiom 2.4.4 (Skip) **skip sat** $term = start$

The assignment axiom expresses that $x := e$ terminates after K_a time units and the final value of x equals the value of e at the initial state. If x occurs in the expression e, the initial value of x is needed to evaluate the value of e. We use $first(x)$ to record the initial value of x.

Axiom 2.4.5 (Assignment)

$$x := e \textbf{ sat } (x = first(x))\ \mathcal{U}\ (T = term = start + K_a \ \wedge\ x = e[first(x)/x])$$

Example 2.4.1 With this axiom and the consequence rule we can derive, for instance,
$x := x + 1$ **sat** $(last(x) = first(x) + 1) \wedge \Diamond\, (T = term = start + K_a)$. □

Example 2.4.2 We show that we can derive
$x := y + 4$ **sat** $y = 5 \rightarrow \Diamond\, (x = 9 \wedge T = term = start + K_a)$.
By the assignment axiom and the consequence rule we obtain
$x := y + 4$ **sat** $\Diamond\, (x = y + 4 \wedge T = term = start + K_a)$.
Since $y \notin wvar(x := y + 4)$, by the variable invariance axiom, we have
$x := y + 4$ **sat** $\Box\, (y = first(y))$.
Since $\models y = 5 \rightarrow \Box\, (first(y) = 5)$, by the assumption, we have

$\vdash y = 5 \rightarrow \Box(first(y) = 5)$. Then by the conjunction rule and consequence rule, we obtain

$$x := y + 4 \textbf{ sat } y = 5 \rightarrow \Box(y = 5).$$

Hence, by the conjunction rule and consequence rule again, we get

$$x := y + 4 \textbf{ sat } y = 5 \rightarrow \Diamond(x = 9 \wedge T = term = start + K_a). \qquad \Box$$

Statement **delay** e terminates after e time units if the value of e is not negative. Otherwise it terminates immediately like **skip**.

Axiom 2.4.6 (Delay) $\quad \textbf{delay } e \textbf{ sat } term = start + max(0, e)$

An output statement starts with waiting to send a message, and as soon as a communication partner is available the communication takes place during K_c time units. Note that we use a weak until operator in the axiom below to allow an infinite waiting period (i.e., deadlock) when no partner becomes available.

Axiom 2.4.7 (Output)

$$c!e \textbf{ sat } wait(c!) \textbf{ U } (T = term - K_c \wedge (comm(c, e) \; \mathcal{U} \; T = term))$$

Similarly, an input statement $c?x$ waits to receive a value along channel c. When the communication finishes the value received is assigned to variable x. Thus at the last state of the execution model x possesses that value.

Axiom 2.4.8 (Input)

$$\begin{aligned} c?x \textbf{ sat } & (x = first(x) \wedge wait(c?)) \textbf{ U } \\ & (T = term - K_c \wedge ((x = first(x) \wedge comm(c, last(x))) \; \mathcal{U} \; T = term)) \end{aligned}$$

Using the $\mathcal{C}$ operator we can easily formulate an inference rule for sequential composition.

Rule 2.4.3 (Sequential Composition) $\quad \dfrac{S_1 \textbf{ sat } \varphi_1, \; S_2 \textbf{ sat } \varphi_2}{S_1; S_2 \textbf{ sat } \varphi_1 \; \mathcal{C} \; \varphi_2}$

Example 2.4.3 Consider process $x := x + 1; x := x + 2$. By the assignment axiom and the consequence rule we have:

$$x := x + 1 \textbf{ sat } last(x) = first(x) + 1 \wedge term = start + K_a, \text{ and}$$
$$x := x + 2 \textbf{ sat } last(x) = first(x) + 2 \wedge term = start + K_a.$$

Then the sequential composition rule leads to

$$\begin{aligned} & x := x + 1; x := x + 2 \textbf{ sat } \\ & (last(x) = first(x) + 1 \wedge term = start + K_a) \; \mathcal{C} \\ & (last(x) = first(x) + 2 \wedge term = start + K_a). \end{aligned}$$

By the consequence rule, we obtain

$$x := x + 1; x := x + 2 \textbf{ sat } last(x) = first(x) + 3 \wedge term = start + 2K_a. \qquad \Box$$

Now consider a guarded command G. Recall that $\bar{g}$ is defined as (see section 2.2.2)

$$\bar{g} \equiv \begin{cases} \bigvee_{i=1}^{n} g_i & \text{if } G \equiv [[]_{i=1}^{n} g_i \rightarrow S_i] \\ \bigvee_{i=0}^{n} g_i & \text{if } G \equiv [[]_{i=1}^{n} g_i; c_i?x_i \rightarrow S_i \; [] \; g_0; \textbf{delay } e \rightarrow S_0] \end{cases}$$

First we give an axiom which expresses that if none of the guards evaluates to true then the guarded command terminates after K_g time units. Furthermore we express that there is no activity on the channels of G and no write-variable of G is changed during the evaluation of guards. Define $Eval \equiv term = start + K_g$.

Axiom 2.4.9 (Guarded Command Evaluation)

$$G \textbf{ sat } [(inv(wvar(G)) \wedge empty(dch(G))) \; \mathcal{U} \; (T = start + K_g \;\wedge\; inv(wvar(G)))] \wedge (\neg\bar{g} \rightarrow Eval)$$

Next consider a guarded command with purely boolean guards $G \equiv [[]_{i=1}^{n} g_i \rightarrow S_i]$. If at least one of the guards yields true then after the evaluation of the guards one of the statements S_i for which g_i evaluates to true is executed. This leads to the following rule.

Rule 2.4.4 (Guarded Command with Purely Boolean Guards)

$$\frac{S_i \textbf{ sat } \varphi_i, \quad \text{for } i = 1, \ldots, n}{[[]_{i=1}^{n} g_i \rightarrow S_i] \textbf{ sat } \bar{g} \rightarrow (Eval \; \mathcal{C} \; \bigvee_{i=1}^{n} g_i \wedge \varphi_i)}$$

Next we formulate a rule for $G \equiv [[]_{i=1}^{n} g_i; c_i?x_i \rightarrow S_i \; [] \; g_0; \textbf{delay } e \rightarrow S_0]$, using

$Wait \equiv inv(wvar(G)) \;\wedge\; empty(dch(G) \setminus \{c_1?, \ldots, c_n?\}) \;\wedge$
$\qquad (g_0 \rightarrow T < start + max(0, e)) \;\wedge\; \bigwedge_{i=1}^{n}(g_i \leftrightarrow wait(c_i?))$,

$InTime \equiv inv(wvar(G)) \;\wedge\; T = term \;\wedge\; (g_0 \rightarrow T < start + max(0, e))$,

$EndTime \equiv inv(wvar(G)) \;\wedge\; g_0 \;\wedge\; T = term = start + max(0, e)$,

$Comm \equiv (Wait \; \mathbf{U} \; InTime) \; \mathcal{C} \; \bigvee_{i=1}^{n} g_i \wedge \varphi_i \wedge comm(c_i)$, and

$TimeOut \equiv (Wait \; \mathcal{U} \; EndTime) \; \mathcal{C} \; \varphi_0$.

Rule 2.4.5 (Guarded Command with IO-Guards)

$$\frac{c_i?x_i; S_i \textbf{ sat } \varphi_i, \quad \text{for } i = 1, \ldots, n, \quad S_0 \textbf{ sat } \varphi_0}{\begin{array}{c}[[]_{i=1}^{n} g_i; c_i?x_i \rightarrow S_i \; [] \; g_0; \textbf{delay } e \rightarrow S_0] \textbf{ sat} \\ \bar{g} \rightarrow (Eval \; \mathcal{C} \; (Comm \vee TimeOut))\end{array}}$$

Observe that in the definition of $COMM$ we use $g_i \wedge \varphi_i \wedge comm(c_i)$, where φ_i is such that $c_i?x_i; S_i$ **sat** φ_i. In general, φ_i describes two parts of the computation: a possible waiting period for $c_i?x_i$ followed by a coomunication on channel c_i, and the execution of S_i. According to the definition of well-formedness, adding $comm(c_i)$ to φ_i excludes the possibility of waiting on c_i, and this is exactly what needed in the execution of the guarded command when the communication on c_i should start immediately.

The inference rule for an iterated guarded command is as follows.

Rule 2.4.6 (Iteration) $$\frac{G \ \textbf{sat} \ \varphi}{\star G \ \textbf{sat} \ (\bar{g} \wedge \varphi) \ \mathcal{C}^* \ (\neg \bar{g} \wedge \varphi)}$$

Next consider parallel composition of S_1 and S_2. Suppose we have deduced specifications φ_1 and φ_2 for, respectively, S_1 and S_2. If φ_1 and φ_2 do not contain *term*, then we have the following simple rule.

Rule 2.4.7 (Simple Parallel Composition)

$$\frac{S_1 \ \textbf{sat} \ \varphi_1, \ S_2 \ \textbf{sat} \ \varphi_2, \ \text{neither } \varphi_1 \text{ nor } \varphi_2 \text{ contain } term}{S_1 \| S_2 \ \textbf{sat} \ \ \varphi_1 \wedge \varphi_2}$$

provided $dch(\varphi_i) \subseteq dch(S_i)$ and $var(\varphi_i) \subseteq var(S_i)$, for $i = 1, 2$.

If one of φ_1 and φ_2 contains *term*, we have to take into account that the termination times of S_1 and S_2 are, in general, different. Observe that if S_1 terminates after (or at the same time as) S_2 then the model representing this computation satisfies $\varphi_1 \wedge (\varphi_2 \ \mathcal{C} \ true)$. Furthermore we have to express that the variables of S_2 are not changed and there is no activity on the channels of S_2 after the termination of S_2. Similarly, for S_1 and S_2 interchanged. Then it leads to the following general rule for parallel composition.

Rule 2.4.8 (General Parallel Composition)
Let $\psi_i \equiv \Box \, (inv(var(S_i)) \ \wedge \ empty(dch(S_i)))$, for $i = 1, 2$.

$$\frac{S_1 \ \textbf{sat} \ \varphi_1, \ S_2 \ \textbf{sat} \ \varphi_2}{S_1 \| S_2 \ \textbf{sat} \ \ (\varphi_1 \wedge (\varphi_2 \ \mathcal{C} \ \psi_2)) \ \vee \ (\varphi_2 \wedge (\varphi_1 \ \mathcal{C} \ \psi_1))}$$

provided $dch(\varphi_i) \subseteq dch(S_i)$ and $var(\varphi_i) \subseteq var(S_i)$, for $i = 1, 2$.

Example 2.4.4 Consider process $c!5 \ \| \ c?x$. Since we have assumed maximal parallelism, the communication takes place immediately and hence this process should satisfy $comm(c, 5) \ \mathcal{U} \ (T = term = start + K_c \wedge x = 5)$.

By the input axiom, output axiom, and the consequence rule, we obtain $c!5$ **sat** φ_1 and $c?x$ **sat** φ_2 with
$\varphi_1 \equiv wait(c!) \ \mathbf{U} \ (T = term - K_c \wedge (comm(c, 5) \ \mathcal{U} \ T = term))$ and
$\varphi_2 \equiv wait(c?) \ \mathbf{U} \ (T = term - K_c \wedge (comm(c, last(x)) \ \mathcal{U} \ T = term))$.

Suppose $\psi_1 \equiv \Box \, empty(\{c, c!\})$ and $\psi_2 \equiv \Box \, (inv(\{x\}) \wedge empty(\{c, c?\}))$.
Then the general parallel composition rule leads to
$c!5 \ \| \ c?x$ **sat** $(\varphi_1 \wedge (\varphi_2 \ \mathcal{C} \ \psi_2)) \vee (\varphi_2 \wedge (\varphi_1 \ \mathcal{C} \ \psi_1))$.
The well-formedness axiom and the conjunction rule allow us to add $MinWait_{\{c!, c?\}}$, $Exclusion_{\{c, c!, c?\}}$, and $Unique_{\{c\}}$ to $(\varphi_1 \wedge (\varphi_2 \ \mathcal{C} \ \psi_2)) \vee (\varphi_2 \wedge (\varphi_1 \ \mathcal{C} \ \psi_1))$.
Consider $\varphi_1 \wedge (\varphi_2 \ \mathcal{C} \ \psi_2) \wedge MinWait_{\{c!, c?\}} \wedge Exclusion_{\{c, c!, c?\}} \wedge Unique_{\{c\}}$.

It implies
$[wait(c!)\ \mathbf{U}\ (T = term - K_c \wedge (comm(c,5)\ \mathcal{U}\ T = term))] \wedge$
$[(wait(c?) \wedge \neg wait(c!) \wedge \neg comm(c))\ \mathbf{U}\ (comm(c, last(x)) \wedge \neg wait(c!))] \wedge Unique_{\{c\}}$,
which implies
$T = term - K_c \wedge (comm(c,5)\ \mathcal{U}\ T = term) \wedge last(x) = 5$.
Since $\models T = start$, the above formula implies
$comm(c,5)\ \mathcal{U}\ (T = term = start + K_c \wedge x = 5)$.
Similarly, we can prove that
$\varphi_2 \wedge (\varphi_1\ \ \mathcal{C}\ \ \psi_1) \wedge MinWait_{\{c!,c?\}} \wedge Exclusion_{\{c,c!,c?\}} \wedge Unique_{\{c\}} \rightarrow$
$comm(c,5)\ \mathcal{U}\ (T = term = start + K_c \wedge x = 5)$.
Then, using the consequence rule again, we obtain
$c!5 \parallel c?x$ **sat** $comm(c,5)\ \mathcal{U}\ (T = term = start + K_c \wedge x = 5)$. □

Example 2.4.5 Consider process $c!0; d!1 \parallel d?x; c?y$. Since this process leads to deadlock,
we should be able to prove $c!0; d!1 \parallel d?x; c?y$ **sat** $\Box(wait(c!) \wedge wait(d?))$.

By the output axiom, the communication invariance axiom, and the consequence rule, we have
$c!0$ **sat** $wait(c!)\ \mathbf{U}\ comm(c)$ and $c!0$ **sat** $\Box \neg comm(d)$.
Using the conjunction rule and the consequence rule, we obtain
$c!0$ **sat** $(wait(c!) \wedge \neg comm(d))\ \mathbf{U}\ (comm(c) \wedge \neg comm(d))$.
Since $((wait(c!) \wedge \neg comm(d))\ \mathbf{U}\ (comm(c) \wedge \neg comm(d)))\ \ \mathcal{C}\ \ true \rightarrow$
$(wait(c!) \wedge \neg comm(d))\ \mathbf{U}\ (comm(c) \wedge \neg comm(d))$,
the sequential composition rule and the consequence rule lead to
$c!0; d!1$ **sat** $(wait(c!) \wedge \neg comm(d))\ \mathbf{U}\ (comm(c) \wedge \neg comm(d))$.
Similarly, we have
$d?x; c?y$ **sat** $(wait(d?) \wedge \neg comm(c))\ \mathbf{U}\ (comm(d) \wedge \neg comm(c))$.
Using the simple parallel composition rule, we obtain
$c!0; d!1 \parallel d?x; c?y$ **sat** $((wait(c!) \wedge \neg comm(d))\ \mathbf{U}\ (comm(c) \wedge \neg comm(d))) \wedge$
$((wait(d?) \wedge \neg comm(c))\ \mathbf{U}\ (comm(d) \wedge \neg comm(c)))$.
Clearly this implies $\Box(wait(c!) \wedge wait(d?))$ and hence, by the consequence rule,
$c!0; d!1 \parallel d?x; c?y$ **sat** $\Box(wait(c!) \wedge wait(d?))$. □

2.5 Application

In this section we illustrate the use of our formalism by specifying and verifying a small part of an avionics system. Detailed specifications of the avionics system can be found in [PWT90]. Here we only consider the design of a reliable device.

A device is a component which receives a request from and sends data to its environment. A reliable device RD consists of a physical device PD and a handler H and is depicted by the following figure 2.1.

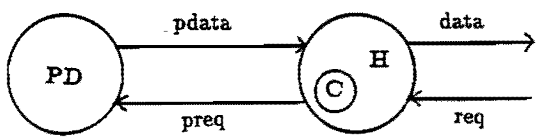

Fig. 2.1 Reliable Device

After receiving a request, the physical device PD either sends some data to its environment along channel *pdata* within a certain amount of time, or it fails to do so but will be ready for the next request on channel *preq* within some time bound. When the handler H receives a request from its environment along channel *req*, it will send a request to the physical device PD along channel *preq* and then wait for PD to send data on channel *pdata*. Then there are two possibilities:

- If PD functions correctly, it will be ready to send some data to H on channel *pdata* within a certain amount of time. After H has received the data, it will send the data to its environment on channel *data*.
- If PD does not function correctly, H will stop waiting after a certain period of time and an approximation of the data will be computed by a component C inside the handler. Then the approximated data will be sent to the environment along channel *data*.

Given a physical device, the problem is to construct a handler such that the composition of the physical device and the handler is a reliable device. We will design a handler H such that the parallel composition of PD and H, $PD \| H$, behaves like RD, i.e., satisfies the given specification of RD.

In this example, we make the following assumptions.

- We focus on the communication behavior of the system and not on how data is produced. Thus we abstract from whether data is precise or approximated and ignore the data when a communication takes place. Hence data will not appear in any specification or process.
- As in the rest of this chapter, communications are synchronous along unidirectional channels and a communication takes K_c time units.

- Component C will take $D_C \geq 0$ time units to compute an approximation of the data.

The specification of the physical device PD is given informally as follows.

1. Initially, PD is waiting to receive a request along channel *preq*.

2. When PD receives a request on channel *preq*, it takes $D_{PD} \geq 0$ time units to process the request. Then either it is ready to send data on channel *pdata* and after having sent data on *pdata* it is again ready for another request on channel *preq*, or it is not ready for sending on *pdata* but it will be ready for another request on *preq* within $D_{PQ} \geq 0$ time units.

The implementation of PD may be in hardware or in software. Since our method is compositional, only the specification of PD is used to construct the reliable device. The formal specification of PD is given as $SPEC_{PD}$ in the following way.

$$
\begin{aligned}
SPEC_{PD} \equiv\ & ([wait(preq?)\ \mathbf{U}\ (comm(preq)\ \mathcal{U}\ T = term)]\ \mathcal{C} \\
& [term = start + D_{PD}]\ \mathcal{C} \\
& [(wait(pdata!)\ \mathbf{U}\ (comm(pdata)\ \mathcal{U}\ T = term)) \vee \\
& (\neg comm(pdata!)\ \mathbf{U}\ T = term \leq start + D_{PQ})])\ \mathcal{C}^*\ false.
\end{aligned}
$$

The specification of the reliable device RD is informally stated as follows.

1. Initially, RD is ready to receive a request from the environment along channel *req* within $D_{RQ} \geq 0$ time units.

2. When RD receives a request on channel *req*, it will be ready to send the data to the environment through channel *data* within $D_{RD} \geq 0$ time units.

3. When RD has sent the data through channel *data*, it will again be ready to accept the next request on channel *req* within D_{RQ} time units.

The formal specification of RD is defined as $SPEC_{RD}$ as follows.

$$
\begin{aligned}
SPEC_{RD} \equiv\ & ([term \leq start + D_{RQ}]\ \mathcal{C} \\
& [wait(req?)\ \mathbf{U}\ (comm(req)\ \mathcal{U}\ T = term)]\ \mathcal{C} \\
& [term \leq start + D_{RD}]\ \mathcal{C} \\
& [wait(data!)\ \mathbf{U}\ (comm(data)\ \mathcal{U}\ T = term)])\ \mathcal{C}^*\ false.
\end{aligned}
$$

Our aim is to find a handler H such that $PD \| H$ **sat** $SPEC_{RD}$. After having examined the requirement of RD and the specification of PD, we propose the following specification for H.

1. Initially, H should be ready to receive a request from the environment along channel *req* within D_{RQ} time units.

2. When H receives a request on channel req, it is immediately ready to send a request to PD on channel $preq$. After the communication on $preq$ finishes, H is allowed to wait $D_0 \geq 0$ time units before it is ready to receive on channel $pdata$ for at most D_1 time units. If a communication on $pdata$ starts in less than D_1 time units, then after this communication H is ready to send on channel $data$. If no communication occurs on $pdata$ in less than D_1 time units, H starts to compute an approximation of the data by means of the component C and then is ready to send the data on channel $data$.

3. When H has sent the data along channel $data$, it will again be ready to accept the next request on channel req within D_{RQ} time units.

The values of the constants D_0 and D_1 will be determined later. These informal descriptions can be formalized in our specification language as $SPEC_H$.

$$
\begin{aligned}
SPEC_H \equiv ([term \leq start + D_{RQ}]\ \mathcal{C}\ & \\
& [wait(req?)\ \mathbf{U}\ (comm(req)\ \mathcal{U}\ T = term)]\ \mathcal{C} \\
& [wait(preq!)\ \mathbf{U}\ (comm(preq)\ \mathcal{U}\ T = term)]\ \mathcal{C} \\
& [term = start + D_0]\ \mathcal{C} \\
& [(wait(pdata?)\ \mathcal{U}\ (comm(pdata)\ \mathcal{U}\ T = term < start + D_1 + K_c)) \vee \\
& ((wait(pdata?)\ \mathcal{U}\ T = term = start + D_1)\ \mathcal{C}\ (term = start + D_C))]\ \mathcal{C} \\
& [wait(data!)\ \mathbf{U}\ (comm(data)\ \mathcal{U}\ T = term)])\ \mathcal{C}^*\ false.
\end{aligned}
$$

Then the handler H is specified by H **sat** $SPEC_H$. For the physical device PD we have, by assumption, PD **sat** $SPEC_{PD}$. To show that $PD \| H$ **sat** $SPEC_{RD}$, we apply the parallel composition rule. Observe that although $SPEC_{PD}$ and $SPEC_H$ contain $term$, we have $SPEC_{PD}\ \mathcal{C}\ \psi \leftrightarrow SPEC_{PD}$ and $SPEC_H\ \mathcal{C}\ \psi \leftrightarrow SPEC_H$, for any formula ψ. Then by the general parallel composition rule, we obtain $PD \| H$ **sat** $SPEC_{PD} \wedge SPEC_H$. Let

$cset \equiv \{preq?, preq!, preq, pdata?, pdata!, pdata, req?, req, data!, data\}$ and $WFD \equiv WF_{cset}$. By the well-formedness axiom, we have $PD \| H$ **sat** WFD. Using the conjunction rule, we obtain $PD \| H$ **sat** $SPEC_{PD} \wedge SPEC_H \wedge WFD$. If we can prove $SPEC_{PD} \wedge SPEC_H \wedge WFD \rightarrow SPEC_{RD}$, then by the consequence rule, we obtain $PD \| H$ **sat** $SPEC_{RD}$. Hence we have to prove $SPEC_{PD} \wedge SPEC_H \wedge WFD \rightarrow SPEC_{RD}$.

By comparing $SPEC_H$ with $SPEC_{RD}$, we see that the waiting time of H on channel $pdata$ has an upper bound of $D_1 + max(K_c, D_c)$. It remains to determine an upper bound on the waiting time of H on channel $preq$. Therefore we make the following observations.

1. For the first communication on $preq$ H does not need to wait for PD since PD is initially ready for $preq$.

2. Let t_{PD} denote the maximal amount of time for PD to be ready to receive along *preq* after a communication on *preq* completes. Let t_H denote the minimal amount of time for H to be ready to send along *preq* after a communication on *preq* finishes. We will determine t_{PD} and t_H and then use them to derive an upper bound on the waiting time.of H on *preq*. After a communication on *preq* ends, there are two possibilities for PD:

   - PD functions correctly, i.e. after D_{PD} time units it is ready to send on *pdata*. In this case, we should require

     $$D_{PD} < D_0 + D_1, \tag{1}$$

     i.e. H has to wait long enough to receive the data from *pdata*. If this requirement is not satisfied, H will stop waiting for PD on *pdata* and start component C to compute approximated data before PD is ready to send on *pdata*. Then after a next communication on *req* H will start waiting to send on *preq* whereas PD is still waiting to send on *pdata*. Hence this leads to a deadlock.

     After a communication on *preq*, H is ready to receive on *pdata* in D_0 time units. Thus, assuming (1), PD will start the communication on *pdata* after $max(D_{PD}, D_0)$ and then be ready for the next request on *preq*. Hence $t_{PD} = max(D_{PD}, D_0) + K_c$.

     Also H communicates on *pdata* after $max(D_{PD}, D_0)$ waiting time and then is ready to send on *data*. After the communications on *data* and *req* H is again ready for *preq*. Thus $t_H = max(D_{PD}, D_0) + 3K_c$.

     Obviously $t_{PD} < t_H$. Thus PD is ready for *preq* earlier than H is and then H does not have to wait for PD on *preq*. Hence after a *req* communication, H immediately sends along *preq* and the sending takes K_c time units. Next, as above, a communication along *pdata* starts after $max(D_{PD}, D_0)$, which also takes K_c time units, and then H is ready to send on *data*.

     Thus in this case we obtain $SPEC_{RD}$ provided

     $$max(D_{PD}, D_0) + 2K_c \leq D_{RD}. \tag{2}$$

   - Or PD does not function correctly, i.e. after D_{PD} it is not ready for *pdata* but it will be ready for the next request on *preq* within D_{PQ} time units. In this case, we have $t_{PD} = D_{PD} + D_{PQ}$.

     Regarding H, after it has waited $D_0 + D_1$ time units for *pdata* it starts to compute approximated data by component C (which takes D_C time units) and then is ready for channel *data*. Then we have $t_H = D_0 + D_1 + D_C + 2K_c$.

     – If $t_{PD} \leq t_H$, i.e. $D_{PD} + D_{PQ} \leq D_0 + D_1 + D_C + 2K_c$, then H does not have to wait for PD on *preq*. In this case $SPEC_{PD} \wedge SPEC_H \wedge WFD$

leads to

$$\begin{array}{l}([term \leq start + D_{RQ}]\ \mathcal{C}\\ [wait(req?)\ \mathbf{U}\ (comm(req)\ \mathcal{U}\ T = term)]\ \mathcal{C}\\ [term = start + K_c]\ \mathcal{C}\\ [term = start + D_0]\ \mathcal{C}\\ [term = start + D_1 + D_C]\ \mathcal{C}\\ [wait(data!)\ \mathbf{U}\ (comm(data)\ \mathcal{U}\ T = term)])\ \mathcal{C}^* false\end{array}$$

Hence, to obtain $SPEC_{RD}$, we require $K_c + D_0 + D_1 + D_C \leq D_{RD}$, i.e.,
$K_c + D_0 + D_1 \leq D_{RD}$. (3)

- If $t_{PD} > t_H$, i.e. $D_{PD} + D_{PQ} > D_0 + D_1 + D_C + 2K_c$, then H has to wait at most $t_{PD} - t_H$ time units for PD on *preq*. Thus $SPEC_{PD} \wedge SPEC_H \wedge WFD$ leads to

$$\begin{array}{l}([term \leq start + D_{RQ}]\ \mathcal{C}\\ [wait(req?)\ \mathbf{U}\ (comm(req)\ \mathcal{U}\ T = term)]\ \mathcal{C}\\ [term = start + t_{PD} - t_H + K_c]\ \mathcal{C}\\ [term = start + D_0]\ \mathcal{C}\\ [term = start + D_1 + D_C]\ \mathcal{C}\\ [wait(data!)\ \mathbf{U}\ (comm(data)\ \mathcal{U}\ T = term)])\ \mathcal{C}^*\ false\end{array}$$

Therefore we have to require $t_{PD} - t_H + K_c + D_0 + D_1 + D_C \leq D_{RD}$, i.e.,
$D_{PD} + D_{PQ} - K_c \leq D_{RD}$. (4)

Conditions (1), (2), (3), and (4) are the restrictions on the parameters to achieve the required implication. By these restrictions, we only know the relation between D_0 and D_1. When we implement H below, we obtain the value of D_0 and then the value of D_1 is determined as well.

Now we implement H in our programming language. We propose the following process H.

$H ::= \star[\ req? \rightarrow preq!; [\ pdata? \rightarrow data! [\!] \mathbf{delay} D_1 \rightarrow C; data!\ ]\ ]$
where process C is such that C **sat** $term = start + D_C$.

We show that H **sat** $SPEC_H$. By the proof system, we can derive that H **sat** φ_H with

$$\begin{array}{l}\varphi_H \equiv ([term = start + K_g]\ \mathcal{C}\\ \quad [wait(req?)\ \mathbf{U}\ (T = term - K_c \wedge (comm(req)\ \mathcal{U}\ T = term))]\ \mathcal{C}\\ \quad [wait(preq!)\ \mathbf{U}\ (T = term - K_c \wedge (comm(preq)\ \mathcal{U}\ T = term))]\ \mathcal{C}\\ \quad [term = start + K_g]\ \mathcal{C}\\ \quad [(wait(pdata?)\ \mathcal{U}\ (T = term - K_c \wedge (comm(pdata)\ \mathcal{U}\\ \qquad T = term < start + D_1 + K_c))) \vee\\ \quad ((wait(pdata?)\ \mathcal{U}\ T = term = start + D_1)\ \mathcal{C}\ (term = start + D_c))]\ \mathcal{C}\\ \quad [wait(data!)\ \mathbf{U}\ (T = term - K_c \wedge (comm(data)\ \mathcal{U}\ T = term)])\ \mathcal{C}^*\ false\end{array}$$

By comparing $SPEC_H$ and φ_H, we can easily derive $\varphi_H \rightarrow SPEC_H$, i.e., H **sat** $SPEC_H$ and then process H is a correct implementation of the handler H, provided

$D_{RQ} \geq K_g$ (5)

and $D_0 = K_g$. Combining the conditions (1) through (4), we see that (1) and (3) are equivalent to the following condition on D_1:

$D_{PD} - K_g < D_1 \leq D_{RD} - K_c - K_g$. (6)

We show that $(D_{PD} - K_g, D_{RD} - K_c - K_g]$ is not an empty interval, i.e., D_1 can be found. We only have to prove that $D_{PD} < D_{RD} - K_c$. Recall $D_0 = K_g$. If $D_{PD} \geq D_0$, by (2), we have $D_{PD} + 2K_c \leq D_{RD}$ and then, since $K_c > 0$, $D_{PD} + K_c < D_{RD}$. If $D_{PD} < D_0$, by (2) again, we obtain $K_g + 2K_c \leq D_{RD}$, i.e. $D_{PD} + K_c < D_{RD}$. Thus the condition (6) for D_1 is reasonable.

Furthermore, by $D_0 = K_g$, the condition (2) can be replaced by the following (2'):

$max(D_{PD}, K_g) + 2K_c \leq D_{RD}$. (2')

Hence the final restrictions on the parameters are (2'), (4), (5), and (6).

2.6 Soundness and Completeness

In this section, we consider the soundness and completeness of the proof system in section 2.4. For the soundness of our proof system, we must show that every formula S **sat** φ derivable in the proof system is indeed valid. We first give a few lemmas which will be used to prove the soundness. The proofs of these lemmas can be found in Appendix A.

Lemma 2.6.1 For any expression e from the programming language, any model σ, and any $\tau \geq begin(\sigma)$, $\mathcal{E}(e)(\sigma(\tau).s) = \mathcal{V}(e)(\sigma, \tau)$.

Lemma 2.6.2 For any boolean guard g from the programming language, any model σ, and any $\tau \geq begin(\sigma)$, $\mathcal{G}(g)(\sigma(\tau).s)$ iff $\langle \sigma, \tau \rangle \models g$.

Lemma 2.6.3 For any expression $vexp$ of type VAL, any model σ, any $cset \subseteq DCHAN$, and any $\tau \geq begin(\sigma)$, $\mathcal{V}(vexp)(\sigma, \tau) = \mathcal{V}(vexp)([\sigma]_{cset}, \tau)$.

Lemma 2.6.4 For any expression $vexp$ of type VAL, any model σ, any $vset \subseteq VAR$, and any $\tau \geq begin(\sigma)$, if $var(vexp) \subseteq vset$, then $\mathcal{V}(vexp)(\sigma, \tau) = \mathcal{V}(vexp)(\sigma \downarrow vset, \tau)$.

Lemma 2.6.5 For any expression $texp$ of type $TIME$, any model σ, any $cset \subseteq DCHAN$, and any $\tau \geq begin(\sigma)$, $\mathcal{T}(texp)(\sigma, \tau) = \mathcal{T}(texp)([\sigma]_{cset}, \tau)$.

Lemma 2.6.6 For any expression $texp$ of type $TIME$, any model σ, any $vset \subseteq VAR$, and any $\tau \geq begin(\sigma)$, if $var(texp) \subseteq vset$, then $\mathcal{T}(texp)(\sigma, \tau) = \mathcal{T}(texp)(\sigma \downarrow vset, \tau)$.

Lemma 2.6.7 For any $cset \subseteq DCHAN$ and any specification φ, if $dch(\varphi) \subseteq cset$, then for any model σ and any $\tau \geq begin(\sigma)$, $\langle \sigma, \tau \rangle \models \varphi$ iff $\langle [\sigma]_{cset}, \tau \rangle \models \varphi$.

Lemma 2.6.8 For any $vset \subseteq VAR$ and any specification φ, if $var(\varphi) \subseteq vset$, then for any model σ and any $\tau \geq begin(\sigma)$, $\langle \sigma, \tau \rangle \models \varphi$ iff $\langle \sigma \downarrow vset, \tau \rangle \models \varphi$.

Given these lemmas, we have the following soundness theorem.

Theorem 2.6.1 (Soundness) The proof system in section 2.4 is sound.

To prove this theorem, we have to show that all axioms are valid and all inference rules preserve validity, i.e., if the hypotheses of any rule are valid, so is the conclusion. For most axioms and inference rules, soundness follows directly from the definitions of semantics and given lemmas. The detailed proofs can be found in Appendix B.

We would also like the proof system to be *complete*, i.e. if S **sat** φ is valid then it is derivable from our proof system. Observe that the consequence rule relies on implications that are formulae in Explicit Clock Temporal Logic (ECTL), and hence the completeness of our proof system also requires that every valid ECTL formula is provable. Since proof systems for ECTL are beyond the scope of this thesis, we prove *relative completeness*: Every valid specification is derivable in our proof system, assuming that any valid ECTL formula can be proved.

We first give some lemmas which will be used in the completeness proof. The proofs of these lemmas can be found in Appendix A.

Lemma 2.6.9 For any model σ and any $cset \subseteq DCHAN$, $dch(\sigma) \subseteq cset$ iff $\sigma = [\sigma]_{cset}$.

Lemma 2.6.10 For any model σ and any $cset_1, cset_2 \subseteq DCHAN$, if $\langle \sigma, begin(\sigma) \rangle \models \Box\, empty(cset_2 \setminus cset_1)$, then $[\sigma]_{cset_1 \cup cset_2} = [\sigma]_{cset_1}$.

Lemma 2.6.11 For any model σ and any $vset_1, vset_2 \subseteq VAR$, if $\langle \sigma, begin(\sigma) \rangle \models \Box\, inv(vset_2 \setminus vset_1)$, then $\sigma \downarrow (vset_1 \cup vset_2) = \sigma \downarrow vset_1$.

Lemma 2.6.12 For any model σ, if $dch(\sigma) \subseteq cset$ and $\langle \sigma, begin(\sigma) \rangle \models WF_{cset}$, then σ is well-formed.

In order to prove the relative completeness of our system, we define a property of specifications called *preciseness.*

Definition 2.6.1 (Invariant Variable) A variable x is *invariant* with respect to a model σ iff for all τ, $begin(\sigma) \leq \tau \leq end(\sigma)$, $\sigma(\tau).s(x) = \sigma^b.s(x)$.

Definition 2.6.2 (Preciseness) A specification φ is *precise* for a statement S of the programming language in section 2.1 iff

1. S **sat** φ holds, i.e., $\langle \sigma, begin(\sigma)\rangle \models \varphi$, for any $\sigma \in \mathcal{M}(S)$;

2. If σ is a well-formed model, $dch(\sigma) \subseteq dch(S)$, for any variable $x \notin wvar(S)$, x is invariant with respect to σ, and $\langle \sigma, begin(\sigma)\rangle \models \varphi$, then $\sigma \in \mathcal{M}(S)$; and

3. $dch(\varphi) = dch(S)$ and $var(\varphi) = var(S)$.

A precise specification φ for S thus characterizes all possible computations of S: φ is valid for S, and any "reasonable" computation satisfying φ is a possible computation of S.

We first prove that for any statement S a precise specification can be derived from the axioms and inference rules (Theorem 2.6.2). We then show (in Theorem 2.6.3) that any specification φ_2 which is valid for S can be derived from a precise specification φ_1 for S and three predicates. Hence, relative completeness follows directly (Theorem 2.6.4).

Theorem 2.6.2 If S is a statement from the programming language in section 2.1, then a precise specification for S can be derived by using the proof system in section 2.4.

The proof of this theorem can be found in Appendix C.

Theorem 2.6.3 If φ_1 is precise for S and φ_2 is valid for S, then
$\models [\varphi_1 \wedge WF_{dch(\varphi_1)} \wedge \Box\,[empty(dch(\varphi_2) \setminus dch(\varphi_1)) \wedge inv(var(\varphi_2) \setminus var(\varphi_1))]] \rightarrow \varphi_2$.

Proof: Let φ_1 be precise for S and φ_2 be valid for S. Consider a model σ. Assume that $\langle \sigma, begin(\sigma)\rangle \models \varphi_1 \wedge WF_{dch(\varphi_1)} \wedge \Box\,[empty(dch(\varphi_2) \setminus dch(\varphi_1)) \wedge inv(var(\varphi_2) \setminus var(\varphi_1))]$ holds. We show $\langle \sigma, begin(\sigma)\rangle \models \varphi_2$.
By $\langle \sigma, begin(\sigma)\rangle \models \varphi_1$, lemma 2.6.7 leads to $\langle [\sigma]_{dch(\varphi_1)}, begin(\sigma)\rangle \models \varphi_1$. By lemma 2.6.8, $\langle [\sigma]_{dch(\varphi_1)} \downarrow var(\varphi_1), begin(\sigma)\rangle \models \varphi_1$. From $\langle \sigma, begin(\sigma)\rangle \models WF_{dch(\varphi_1)}$, by lemma 2.6.7, we obtain $\langle [\sigma]_{dch(\varphi_1)}, begin(\sigma)\rangle \models WF_{dch(\varphi_1)}$. Then, by lemma 2.6.12, $[\sigma]_{dch(\varphi_1)}$ is well-formed. By definition, $[\sigma]_{dch(\varphi_1)} \downarrow var(\varphi_1)$ is also well-formed. Since φ_1 is precise for S, we have $dch(\varphi_1) = dch(S)$ and $var(\varphi_1) = var(S)$. By the definition of projection onto variables, any variable $x \notin wvar(S)$ is invariant with respect to $[\sigma]_{dch(\varphi_1)} \downarrow var(\varphi_1)$. Hence by the definition of preciseness, $[\sigma]_{dch(\varphi_1)} \downarrow var(\varphi_1) \in \mathcal{M}(S)$.
From $\langle \sigma, begin(\sigma)\rangle \models \Box\, empty(dch(\varphi_2) \setminus dch(\varphi_1))$, lemma 2.6.10 leads to $[\sigma]_{dch(\varphi_1) \cup dch(\varphi_2)} = [\sigma]_{dch(\varphi_1)}$. Since $\langle \sigma, begin(\sigma)\rangle \models \Box\, inv(var(\varphi_2) \setminus var(\varphi_1))$, lemma 2.6.11 leads to $\sigma \downarrow (var(\varphi_1) \cup var(\varphi_2)) = \sigma \downarrow var(\varphi_1)$. Thus we obtain $[\sigma]_{dch(\varphi_1) \cup dch(\varphi_2)} \downarrow (var(\varphi_1) \cup var(\varphi_2)) = [\sigma]_{dch(\varphi_1)} \downarrow var(\varphi_1)$. Therefore we have

$[\sigma]_{dch(\varphi_1)\cup dch(\varphi_2)} \downarrow (var(\varphi_1) \cup var(\varphi_2)) \in \mathcal{M}(S)$. Since φ_2 is valid for S, we obtain $\langle [\sigma]_{dch(\varphi_1)\cup dch(\varphi_2)} \downarrow (var(\varphi_1) \cup var(\varphi_2)), begin(\sigma)\rangle \models \varphi_2$. From $var(\varphi_2) \subseteq var(\varphi_1) \cup var(\varphi_2)$, lemma 2.6.8 leads to $\langle [\sigma]_{dch(\varphi_1)\cup dch(\varphi_2)}, begin(\sigma)\rangle \models \varphi_2$. By $dch(\varphi_2) \subseteq (dch(\varphi_1) \cup dch(\varphi_2))$, lemma 2.6.7 leads to $\langle \sigma, begin(\sigma)\rangle \models \varphi_2$. Hence this theorem holds. □

Theorem 2.6.4 (Relative Completeness) The proof system in section 2.4 is relatively complete.

Proof: For any process S, assume that specification φ is valid for S. We prove that S **sat** φ is derivable in the proof system in section 2.4. By theorem 2.6.2, we have S **sat** φ_1 where φ_1 is a precise specification for S. By the well-formedness axiom, we obtain S **sat** $WF_{dch(\varphi_1)}$. Since $dch(\varphi_1) = dch(S)$, we have $[dch(\varphi) \setminus dch(\varphi_1)] \cap dch(S) = \emptyset$. Then by the communication invariance axiom, we obtain S **sat** $\Box\, empty(dch(\varphi) \setminus dch(\varphi_1))$. From $var(\varphi_1) = var(S)$, we have $[var(\varphi) \setminus var(\varphi_1)] \cap var(S) = \emptyset$ and thus $[var(\varphi) \setminus var(\varphi_1)] \cap wvar(S) = \emptyset$. By the variable invariance axiom, we obtain S **sat** $\Box\, inv(var(\varphi) \setminus var(\varphi_1))$. Then the conjunction rule and the consequence rule lead to S **sat** $\varphi_1 \wedge WF_{dch(\varphi_1)} \wedge \Box\,[empty(dch(\varphi) \setminus dch(\varphi_1)) \wedge inv(var(\varphi) \setminus var(\varphi_1))]$. By theorem 2.6.3, $[\varphi_1 \wedge WF_{dch(\varphi_1)} \wedge \Box\,[empty(dch(\varphi) \setminus dch(\varphi_1)) \wedge inv(var(\varphi) \setminus var(\varphi_1))]] \rightarrow \varphi$ is valid and, by our relative completeness assumption, provable. Hence, by the consequence rule, S **sat** φ is derivable in the proof system. □

Chapter 3

Asynchronous Communication

In this chapter, we study a verification theory for asynchronously communicating real-time systems. In section 3.1, we define the asynchronous version of our programming language in which parallel processes communicate through asynchronous message passing. A compositional semantics is given in section 3.2. The asynchronous version of the specification language is presented in section 3.3. A compositional proof system is shown in section 3.4. The soundness and completeness issues are discussed in section 3.5.

3.1 Real-Time Programming Language

3.1.1 Syntax and Informal Semantics

Consider a real-time programming language in which parallel processes communicate by sending and receiving messages along channels. A channel connects exactly two processes. Communication is asynchronous, that is, a sender does not synchronize with a receiver but sends its message immediately. Similar to the programming language in chapter 2, a real-time statement **delay** e is added to suspend execution for a certain period of time. Such a delay-statement may also occur in a guard of a guarded command. Parallel processes do not share variables. Nested parallelism is allowed.

Similar to chapter 2, let VAR be a nonempty set of variables, $CHAN$ be a nonempty set of channel names, and VAL be a nonempty domain of values. The syntax of the real-time programming language is given in table 3.1, with $c, c_i \in CHAN$, $x, x_i \in VAR$, $n \in I\!N$, and $n \geq 1$, where $I\!N$ denotes the set of all natural numbers.

Notice that this programming language is similar to the programming language in chapter 2 section 2.1, except three statements involving communication. We give the informal meaning of these three statements as follows:

<u>Atomic statements</u>

Table 3.1: Syntax of the Programming Language in Chapter 3

Expression	$e ::=$	$\vartheta \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2$
Guard	$g ::=$	$e_1 = e_2 \mid e_1 < e_2 \mid \neg g \mid g_1 \vee g_2$
Statement	$S ::=$	$\mathbf{skip} \mid x := e \mid \mathbf{delay}\ e \mid c!!e \mid c??x \mid$
		$S_1; S_2 \mid G \mid \star G \mid S_1 \| S_2$
Guarded Command	$G ::=$	$[[]_{i=1}^{n} g_i \rightarrow S_i] \mid [[]_{i=1}^{n} g_i; c_i??x_i \rightarrow S_i [] g_0; \mathbf{delay}\ e \rightarrow S_0]$

- $c!!e$ sends the value of e to the buffer of channel c. We assume that there is an (unbounded) buffer for every channel. Since the communication is asynchronous, $c!!e$ never waits for its communication partner.

- $c??x$ reads a value from the buffer of channel c and assigns it to variable x. If the buffer is empty, $c??x$ has to wait until a message arrives.

Compound statements

- The execution of a guarded command $[[]_{i=1}^{n} g_i; c_i??x_i \rightarrow S_i [] g_0; \mathbf{delay}\ e \rightarrow S_0]$ is similar to the execution of $[[]_{i=1}^{n} g_i; c_i?x_i \rightarrow S_i [] g_0; \mathbf{delay}\ e \rightarrow S_0]$ from chapter 2, except that the communication in the guards here is asynchronous.

Similar to chapter 2, any statement in this programming language is called a *process*. A write-variable is a variable which occurs in a receive statement (i.e. $c??x$) or on the left hand side of an assignment. Let S be any statement. We also use $var(S)$ and $wvar(S)$ to denote the set of variables and write-variables occurring in S, respectively. We define $ch(S)$ as the set of all channel names occurring in S, $ich(S)$ as the set of all input channel names occurring in S, and $och(S)$ as the set of all output channel names appearing in S. Notice that $ich(S) \cup och(S) = ch(S)$ and $ich(S) \cap och(S)$ denotes the set of internal channels. For instance, $ch(c!!5) = och(c!!5) = \{c\}$, $ich(c!!5) = \emptyset$, $ich(c!!3; d??x \| c??y) = \{c, d\}$, and $och(c!!3; d??x \| c??y) = \{c\}$.

3.1.2 Basic Assumptions

Similar to chapter 2, we assume that there is no overhead for compound statements and that a **delay** e statement takes exactly e time units if the value of e is not negative. We also assume given positive parameters K_a and K_g such that each assignment takes K_a time units and the evaluation of the guards in a guarded command takes K_g time units. The new assumption here is that we assume a positive parameter K_c such that each sending takes K_c time units and each reading takes K_c time units. It is possible to generalize these assumptions, for instance, sending and reading take different times.

In this chapter we also use the *maximal parallelism* model to represent the situation that each parallel process runs at its own processor. Hence any action is executed as soon as possible. A process only waits when it tries to receive a message from a channel but the buffer for that channel is empty.

3.2 Compositional Semantics

In this section, we give a compositional semantics for the programming language defined in section 3.1. First we define a computational model in section 3.2.1. Then we describe the formal semantics in section 3.2.2.

3.2.1 Computational Model

Similar to chapter 2, the timing behavior of a process is expressed from the viewpoint of an external observer with his own clock. Thus we will use the same time domain *TIME* as defined in chapter 2, i.e., $TIME = \{\tau \in I\!R \mid \tau \geq 0\}$. We will also use the notations defined there, for instance, $[\tau_0, \tau_1]$, denoting a closed interval of time points, $(\tau_0, \tau_1]$, representing a left-open and right-closed interval, and so on.

Next we define a model representing a real-time computation of a process.

Definition 3.2.1 (Model) Let $\tau_0 \in TIME$, $\tau_1 \in TIME \cup \{\infty\}$, and $\tau_1 \geq \tau_0$. A *model* σ is a mapping $\sigma : [\tau_0, \tau_1] \rightarrow STATE \times \wp(COMM) \times \wp(COMM)$, where $STATE = \{s \mid s : VAR \rightarrow VAL\}$ and $COMM = \{(c, \vartheta) \mid c \in CHAN \text{ and } \vartheta \in VAL\}$. Define $begin(\sigma) = \tau_0$ and $end(\sigma) = \tau_1$. The set of all models is denoted by MOD.

Consider a model σ and a $\tau \in [begin(\sigma), end(\sigma)]$. Then we have $\sigma(\tau) = (s, S, R)$ with $s \in STATE$, $S \subseteq COMM$, and $R \subseteq COMM$. Henceforth we refer to the three fields of $\sigma(\tau)$ by $\sigma(\tau).s$, $\sigma(\tau).S$, and $\sigma(\tau).R$, respectively. Informally, if σ models a computation of a process P, $begin(\sigma)$ and $end(\sigma)$ denote, resp., the starting and terminating times of this computation ($end(\sigma) = \infty$ if P does not terminate). Furthermore, $\sigma(begin(\sigma)).s$ specifies the initial state of the computation, and if $end(\sigma) < \infty$ then $\sigma(end(\sigma)).s$ gives the final state. We will use σ^b to denote $\sigma(begin(\sigma))$ and, if $end(\sigma) < \infty$, σ^e to denote $\sigma(end(\sigma))$. In general, $\sigma(\tau).s$ represents the values of variables. For a channel c and a value $\vartheta \in VAL$, a record (c, ϑ) has the following meaning:

- $(c, \vartheta) \in \sigma(\tau).S$ iff process P or the environment of P has sent value ϑ along c at time τ;
- $(c, \vartheta) \in \sigma(\tau).R$ iff process P has read value ϑ from (the buffer of) channel c at time τ.

Note that, using the syntax of process P, we can observe if a message has been sent by P itself or by its environment. For instance, if $P \equiv c!!5$ and σ represents an execution of P, we are sure that if $(c, 5)$ is in some S-field of σ, value 5 is sent by P itself, since it is assumed that each channel connects exactly two processes. On the other hand, if $P \equiv c??x$ and $(c, 5)$ occurs in some S-field of σ, value 5 is sent by the environment of P.

In the description of the semantics we use the following definitions.

The definition about the *variant* of a state s is the same as the one in chapter 2.

Definition 3.2.2 (Input Channels Occurring in a Model) The set of input channels occurring in a model σ, denoted by $ich(\sigma)$, is defined as

$ich(\sigma) = \bigcup_{begin(\sigma) \le \tau \le end(\sigma)} \{c \mid \text{ there exists a } \vartheta \in VAL \text{ such that } (c, \vartheta) \in \sigma(\tau).R\}$

Definition 3.2.3 (Prefix of a Model) A model σ_1 is a prefix of model σ_2, denoted by $\sigma_1 \preceq \sigma_2$, iff $begin(\sigma_1) = begin(\sigma_2)$, $end(\sigma_1) \le end(\sigma_2)$, and for any $\tau \in [begin(\sigma_1), end(\sigma_1)]$, $\sigma_1(\tau) = \sigma_2(\tau)$. Define $\sigma_1 \prec \sigma_2$ as $\sigma_1 \preceq \sigma_2 \wedge end(\sigma_1) < end(\sigma_2)$.

Definition 3.2.4 (Concatenation of Models) The *concatenation* of two models σ_1 and σ_2, denoted by $\sigma_1\sigma_2$, is a model σ defined as follows:

- if $end(\sigma_1) = \infty$, then $\sigma = \sigma_1$;
- if $end(\sigma_1) < \infty$, $end(\sigma_1) = begin(\sigma_2)$, and $\sigma_1^e.s = \sigma_2^b.s$, then σ has domain $[begin(\sigma_1), end(\sigma_2)]$ and is defined by $\sigma(\tau) = \begin{cases} \sigma_1(\tau) & \tau \in [begin(\sigma_1), end(\sigma_1)] \\ \sigma_2(\tau) & \tau \in (begin(\sigma_2), end(\sigma_2)] \end{cases}$
- otherwise σ is undefined.

Definition 3.2.5 (Sequence) A sequence q is a finite or infinite list of values. If it is infinite, it takes the form of $\langle \vartheta_1, \vartheta_2, \ldots \rangle$ with $\vartheta_i \in VAL$, for any $i \ge 1$, and its length $|q|$ is ∞. If it is finite, it has the form of $\langle \vartheta_1, \ldots, \vartheta_n \rangle$ for some $n \ge 0$, $n \in I\!N$, with $\vartheta_i \in VAL$, for any i, $1 \le i \le n$, and its length $|q|$ is n. If $n = 0$, it is an empty sequence and denoted by $\langle\rangle$. The set of all sequences is denoted by QUE.

For any nonempty sequence q, $First(q)$ gives the first element of q. For any two sequences q_1 and q_2, $q_1 \cdot q_2$ is the concatenation of q_1 and q_2. If q_2 is a prefix of q_1, $q_1 - q_2$ results in a sequence obtained by removing all elements of q_2 from q_1, otherwise $q_1 - q_2$ is undefined.

Definition 3.2.6 (Buffer) A buffer is represented by a mapping which assigns to each channel a sequence representing the messages in the buffer of the channel.
Define $BUF = \{b \mid b : CHAN \to QUE\}$ as the set of all buffers.

Thus $b(c)$ specifies a sequence which represents the messages in the buffer of channel c.

Next we define the sequence of messages being sent along channel c, by a process or an environment, after a model σ, denoted by $BufS(\sigma)(c)$, as follows.

- $BufS(\sigma)(c)$ records every value ϑ for which there exists a $\tau \in [begin(\sigma), end(\sigma)]$ such that $(c, \vartheta) \in \sigma(\tau).S$.

- $BufS(\sigma)(c)$ is time-ordered, that is, if there exist τ_1 and τ_2 such that $\tau_1 < \tau_2$, $(c, \vartheta_1) \in \sigma(\tau_1).S$, and $(c, \vartheta_2) \in \sigma(\tau_2).S$, then ϑ_1 appears before ϑ_2 in $BufS(\sigma)(c)$.

We can similarly define $BufR(\sigma)(c)$ as the sequence of values being read by a process along channel c after the computation of σ, namely replacing $\sigma(\tau).S$ by $\sigma(\tau).R$ in the corresponding places in the definition of $BufS(\sigma)(c)$.

In the semantics, we assign a set of models to each statement, representing all possible computations of that statement starting with an initial buffer. To compute the resulting buffer after a computation σ with initial buffer b, we give the following definition.

Definition 3.2.7 (Buffer of a Model) For any $\sigma \in MOD$, any $c \in CHAN$, and any $b \in BUF$, the buffer of channel c after a computation σ starting with initial buffer b, denoted by $Buf(b, \sigma)(c)$, is defined as $Buf(b, \sigma)(c) = (b(c) \cdot BufS(\sigma)(c)) - BufR(\sigma)(c)$.

Thus $Buf(b, \sigma)(c)$ representes the sequence of values which are left in the buffer of c after the execution of σ which starts with initial buffer b. The semantics of our programming language will be such that, for any channel c and any σ from the semantics of any statement S starting with any initial buffer b, the sequence of messages being read from c is a prefix of the sequence of messages being stored at the buffer of channel c, i.e., $Buf(b, \sigma)(c) \in QUE$ and thus $Buf(b, \sigma) \in BUF$.

We will use $Buf(b, \sigma_1\sigma_2 \cdots \sigma_n)$ to denote $Buf(Buf(\cdots(Buf(b, \sigma_1), \sigma_2), \cdots), \sigma_n)$.

Definition 3.2.8 (Concatenation) For any $F_1, F_2 \in BUF \rightarrow \wp(MOD)$, we define $CON(F_1, F_2) \in BUF \rightarrow \wp(MOD)$ by
$CON(F_1, F_2)(b) = \{\sigma_1\sigma_2 \mid \sigma_1 \in F_1(b),\ \sigma_2 \in F_2(Buf(b, \sigma_1)), \text{ and } Buf(b, \sigma_1) \in BUF\}$.

It is not difficult to see that CON is associative, i.e.,
$CON(F_1, CON(F_2, F_3))(b) = CON(CON(F_1, F_2), F_3)(b)$.
Henceforth, we use $CON(F_1, F_2, F_3)(b)$ to denote $CON(F_1, CON(F_2, F_3))(b)$.

3.2.2 Formal Semantics

The meaning of a process S, denoted by $\mathcal{M}(S)$, associates to each element $b \in BUF$, a set of models representing all possible computations of S starting at an arbitrary time where the initial contents of the buffer of each channel c is given by $b(c)$. For any process S and a buffer $b \in BUF$, we define $\mathcal{M}(S)(b)$ by induction on the structure of S.

The evaluation of an expression e from the programming language in section 3.1 is a function $\mathcal{E}(e) : STATE \rightarrow VAL$, which is defined similarly as in chapter 2 section 2.2.2. The evaluation of a guard g from the language at a state s, denoted by $\mathcal{G}(g)(s)$, is also defined similarly as in chapter 2 section 2.2.2.

Before giving the semantics, we need to make a general assumption about the S-fields of any model. Since the S-fields of a model contain all the values sent to a process, especially by its environment, we do not describe those S-fields in the semantics of the process. Instead, they only need to obey the following assumption.

General Assumption

For any model σ, any $c \in CHAN$, any τ, $begin(\sigma) \leq \tau \leq end(\sigma)$, and any $\vartheta_1, \vartheta_2 \in VAL$, the following holds:

$$(c, \vartheta_1) \in \sigma(\tau).S \wedge (c, \vartheta_2) \in \sigma(\tau).S \rightarrow \vartheta_1 = \vartheta_2.$$

Informally, this means that there can be at most one value being sent along a channel at any time point. This assumption will be used in, for instance, a theorem concerning the relative completeness of a proof system for this asynchronous version of the programming language.

We first define a predicate $Idle(\sigma)$, which expresses that all states are equal to the initial state and no message has been read during the execution of σ:

Definition 3.2.9 For any model σ, $Idle(\sigma)$ iff for any $\tau \in [begin(\sigma), end(\sigma)]$, $\sigma(\tau).s = \sigma^b.s$ and $\sigma(\tau).R = \emptyset$.

Skip

Statement **skip** terminates immediately without any state change or communication. The S-fields of any model of this statement indicate the messages sent by its environment and thus obey the general assumption.

$\mathcal{M}(\mathbf{skip})(b) = \{\sigma \mid begin(\sigma) = end(\sigma) \text{ and } Idle(\sigma)\}$

Assignment

Statement $x := e$ assigns the value of e to variable x and terminates after K_a time units. All intermediate states before termination are the same as the initial one. The state at termination also equals to the initial state except that the value of x is replaced by the value of e evaluated at the initial state. The R-fields of any model of this statement are empty during the execution period since this statement does not receive messages. But the S-fields show the messages sent by the environment and thus also obey the general assumption.

$$\mathcal{M}(x := e)(b) = \{\sigma \mid end(\sigma) = begin(\sigma) + K_a, \text{ for any } \sigma' \prec \sigma, Idle(\sigma'), \sigma^e.R = \emptyset, \text{ and } \sigma^e.s = (\sigma^b.s : x \mapsto \mathcal{E}(e)(\sigma^b.s))\}$$

Delay

$$\mathcal{M}(\textbf{delay } e)(b) = \{\sigma \mid end(\sigma) = begin(\sigma) + max(0, \mathcal{E}(e)(\sigma^b.s)) \text{ and } Idle(\sigma)\}$$

Send

Statement $c!!e$ sends the value of e to the buffer of channel c. This is represented by a record (c, ϑ_0), where ϑ_0 is the value of e, in the S-field at termination. But before that point, there should be no record (c, ϑ), for any $\vartheta \in VAL$, in any S-field, because c is an output channel of the statement itself and thus the environment cannot send any message along c.

In order to express that no message should be sent along a set of channels during a computation, we define the following predicate.

Definition 3.2.10 For any model σ and any $cset \subseteq CHAN$, $Nomsg(\sigma, cset)$ iff for any $c \in cset$, any $\tau \in [begin(\sigma), end(\sigma)]$, and any $\vartheta \in VAL$, $(c, \vartheta) \notin \sigma(\tau).S$.

Furthermore, it is possible that the environment of $c!!e$ sends some value along another channel $d \not\equiv c$ during the execution of $c!!e$. Thus we need the following definition, which expresses that the projection of a model σ onto a set of channel names $cset$ at S-fields is the same as σ except that the new S-fields contain only those records for which the channel name belongs to $cset$.

Definition 3.2.11 (Projection onto Channels at S-Fields) Let $cset \subseteq CHAN$. Define the projection of a model σ onto $cset$ at S-fields, denoted by $[\sigma]^S_{cset}$, as follows:
$begin([\sigma]^S_{cset}) = begin(\sigma)$, $end([\sigma]^S_{cset}) = end(\sigma)$,
for any $\tau \in [begin(\sigma), end(\sigma)]$, $[\sigma]^S_{cset}(\tau).s = \sigma(\tau).s$, $[\sigma]^S_{cset}(\tau).R = \sigma(\tau).R$, and $[\sigma]^S_{cset}(\tau).S = \{(c, \vartheta) \mid (c, \vartheta) \in \sigma(\tau).S \text{ and } c \in cset\}$.

The semantics of $c!!e$ is then defined as:

$$\mathcal{M}(c!!e)(b) = \{\sigma \mid end(\sigma) = begin(\sigma) + K_c, \text{ for any } \sigma' \prec \sigma, Idle(\sigma'), Nomsg(\sigma', \{c\}), \sigma^e.s = \sigma^b.s, \sigma^e.R = \emptyset, \text{ and } ([\sigma]^S_{\{c\}})^e.S = \{(c, \mathcal{E}(e)(\sigma^b.s))\}\ \}$$

Receive

During the execution of a receive statement $c??x$ there are generally two periods: first there is a waiting period during which the initial buffer of c is empty and no message has been sent by its environment along channel c. Next, when the initial buffer of c is not empty or some message has been sent by the environment along channel c, there is a period of K_c time units during which the actual reading takes place. When the reading finishes, x gets the first value from the buffer of channel c. Let

$$WRead(c??x)(b) = \{\sigma \mid Idle(\sigma), \text{ for any } \sigma' \prec \sigma,\ Buf(b,\sigma')(c) = \langle\rangle, \text{ and } \\ \text{if } end(\sigma) < \infty \text{ then } Buf(b,\sigma)(c) \neq \langle\rangle\}$$

and

$$Read(c??x)(b) = \{\sigma \mid end(\sigma) = begin(\sigma) + K_c, \text{ for any } \sigma' \prec \sigma,\ Idle(\sigma'), \\ \sigma^e.R = \{(c, First(b(c)))\}, \text{ and } \sigma^e.s = (\sigma^b.s : x \mapsto First(b(c)))\}$$

Then the semantics for $c??x$ is defined as:

$$\mathcal{M}(c??x)(b) = CON(WRead(c??x), Read(c??x))(b)$$

Sequential Composition

To give the correct semantics of $S_1; S_2$, the models of S_1 and S_2 should agree with each other such that, if c is an output channel of S_1 but not an output channel of S_2, then (c, ϑ), for any $\vartheta \in VAL$, should not be in any S-field of the model of S_2, because c is an output channel of $S_1; S_2$ and thus the environment of $S_1; S_2$ cannot send any message along c. If c is an output channel of S_2 but not an output channel of S_1, a similar reasoning holds. Let

$$Agree(\sigma_1, \sigma_2, S_1, S_2) \equiv Nomsg(\sigma_1, och(S_2) \setminus och(S_1)) \ \wedge\ Nomsg(\sigma_2, och(S_1) \setminus och(S_2)).$$

The semantics of sequential composition is then defined as:

$$\mathcal{M}(S_1; S_2)(b) = \\ \{\sigma_1\sigma_2 \mid \sigma_1 \in \mathcal{M}(S_1)(b), \sigma_2 \in \mathcal{M}(S_2)(Buf(b, \sigma_1)), \text{ and } Agree(\sigma_1, \sigma_2, S_1, S_2)\}$$

Guarded Command

Define $G_1 \equiv [\|_{i=1}^n g_i \rightarrow S_i]$, $G_2 \equiv [\|_{i=1}^n g_i; c_i??x_i \rightarrow S_i \| \textbf{delay } e \rightarrow S_0]$, $\bar{g} \equiv \bigvee_{i=1}^n g_i$ for G_1, $\bar{g} \equiv \bigvee_{i=0}^n g_i$ for G_2, and $\bar{c} \equiv \{c_1, \ldots, c_n\}$ for G_2.

Consider G_1 first. There are two possibilities for the execution of G_1: either none of the boolean guards evaluates to true and then this command terminates after evaluation, or at least one guard g_i yields true and then the corresponding statement S_i is executed.

Recall that the evaluation of guards takes K_g time units. During the evaluation

period, the S-fields of any model of G_i, for $i = 1, 2$, should not contain any (c, ϑ) with $c \in och(G_i)$ and $\vartheta \in VAL$, because the environment of G_i cannot send any message to $och(G_i)$ and G_i itself has not yet sent values to $och(G_i)$. For $i = 1, 2$, define
$Eval(G_i)(b) = \{\sigma \mid end(\sigma) = begin(\sigma) + K_g, Idle(\sigma), \text{ and } Nomsg(\sigma, och(G_i))\}$.
Then the semantics for G_1 is given as follows.

$$\begin{aligned}\mathcal{M}([\|_{i=1}^n g_i \rightarrow S_i])(b) = \{\sigma \mid \mathcal{G}(\neg\bar{g})(\sigma^b.s) \text{ and } \sigma \in Eval(G_1)(b)\} \cup \\ \{\sigma_1\sigma_2 \mid \text{there exists a } k,\ 1 \leq k \leq n, \text{ such that } \mathcal{G}(g_k)(\sigma_1^b.s), \\ \sigma_1 \in Eval(G_1)(b),\ \sigma_2 \in \mathcal{M}(S_k)(Buf(b, \sigma_1)), \\ \text{and } Nomsg(\sigma_2, och(G_1) \setminus och(S_k))\}\end{aligned}$$

During an execution of a guarded command $[\|_{i=1}^n g_i; c_i??x_i \rightarrow S_i \| g_0; \textbf{delay } e \rightarrow S_0]$, first the guards g_i, for $i = 0, 1, ..., n$, are evaluated. Then,

- if none of the g_i evaluates to true, then the command terminates;
- if g_0 evaluates to true, e is positive, and at least one of the $c_i??x_i$ for which g_i evaluate to true can start reading messages in less than e time units, then one of the first possible $c_i??x_i$ and its corresponding S_i are executed;
- if g_0 evaluates to true and either e is not positive or none of the $c_i??x_i$ for which g_i are true can start reading in less than e time units, then S_0 is executed;
- if g_0 evaluates to false, then the command waits until one of the $c_i??x_i$ for which g_i are true can read messages. Then one of the first possible $c_i??x_i$ and its corresponding S_i are executed.

To give the semantics for G_2, we first define two abbreviations:

$$\begin{aligned}Wait(G_2)(b) = \{\sigma \mid \mathcal{G}(\bar{g})(\sigma^b.s), Idle(\sigma), Nomsg(\sigma, och(G_2)), \text{ for any } \sigma' \prec \sigma, \text{ any } i, \\ 1 \leq i \leq n, \text{ either } \mathcal{G}(\neg g_i)(\sigma^b.s) \text{ or } Buf(b, \sigma')(c_i) = \langle\rangle, \\ \text{and if } end(\sigma) < \infty \text{ then there exists a } k,\ 1 \leq k \leq n, \text{ such that} \\ \mathcal{G}(g_k)(\sigma^b.s) \text{ and } Buf(b, \sigma)(c_k) \neq \langle\rangle\}\end{aligned}$$

$$\begin{aligned}Comm(G_2)(b) = \{\sigma \mid \text{there exists a } k,\ 1 \leq k \leq n, \text{ such that } \mathcal{G}(g_k)(\sigma^b.s), \\ \sigma \in \mathcal{M}(c_k??x_k; S_k)(b), \text{ and } Nomsg(\sigma, och(G_2) \setminus och(S_k))\}\end{aligned}$$

Notice that $Wait(G_2)(b)$ is similar to $WRead(c??x)(b)$.
Using $Wait(G_2)(b)$, we define the following additional abbreviations:

$$\begin{aligned}FinWait(G_2)(b) = \{\sigma \mid \mathcal{G}(g_0)(\sigma^b.s), end(\sigma) < begin(\sigma) + max(0, \mathcal{E}(e)(\sigma^b.s)), \\ \text{and } \sigma \in Wait(G_2)(b)\}\end{aligned}$$

$$\begin{aligned}TimeOut(G_2)(b) = \{\sigma_1\sigma_2 \mid \mathcal{G}(g_0)(\sigma_1^b.s), end(\sigma_1) = begin(\sigma_1) + max(0, \mathcal{E}(e)(\sigma_1^b.s)), Idle(\sigma_1), \\ Nomsg(\sigma_1, och(G_2)), \text{ for any } c_i \in \bar{c},\ Buf(b, \sigma_1)(c_i) = \langle\rangle,\end{aligned}$$

$$\sigma_2 \in \mathcal{M}(S_0)(Buf(b,\sigma_1)), \text{ and } Nomsg(\sigma_2, och(G_2) \setminus och(S_0))\}$$

$$AnyWait(G_2)(b) = \{\sigma \mid \mathcal{G}(\neg g_0)(\sigma^b.s) \text{ and } \sigma \in Wait(G_2)(b)\}$$

Then the semantics for G_2 is given as follows.

$$\begin{aligned}
&\mathcal{M}([[]_{i=1}^n g_i; c_i??x_i \to S_i \; [] \; g_0; \mathbf{delay}\; e \to S_0])(b) = \\
&\quad \{\sigma \mid \mathcal{G}(\neg\bar{g})(\sigma^b.s) \text{ and } \sigma \in Eval(G_2)(b)\} \cup \\
&\quad CON(Eval(G_2), FinWait(G_2), Comm(G_2))(b) \cup \\
&\quad CON(Eval(G_2), TimeOut(G_2))(b) \cup \\
&\quad CON(Eval(G_2), AnyWait(G_2), Comm(G_2))(b)
\end{aligned}$$

Iteration

For a model in the semantics of $\star G$ starting with a buffer b, there are two possibilities:

- either it is a concatenation of a finite sequence of models from $\mathcal{M}(G)(b_i)$, for some b_i, such that each model corresponds to an execution of G starting with b_i and either the last model represents a nonterminating computation of G or all boolean guards evaluate to false at the initial state of the last model,

- or it is a concatenation of an infinite sequence of models from $\mathcal{M}(G)(b_i)$, for some b_i, such that each model represents a terminating computation of G starting with b_i and not all boolean guards yield false at the initial state of each model.

Thus we have the following semantics for $\star G$.

$$\begin{aligned}
\mathcal{M}(\star G)(b) = \{\sigma \mid{} & \text{there exist a } k \in I\!N,\ k \geq 1, \text{ and } \sigma_1, \ldots, \sigma_k \text{ such that } \sigma = \sigma_1 \cdots \sigma_k, \\
& \sigma_1 \in \mathcal{M}(G)(b), \text{ for any } i,\ 2 \leq i \leq k,\ \sigma_i \in \mathcal{M}(G)(Buf(b, \sigma_1 \cdots \sigma_{i-1})), \\
& \text{for any } j,\ 1 \leq j \leq k-1,\ end(\sigma_j) < \infty, \mathcal{G}(\bar{g})(\sigma_j^b.s), \text{ and} \\
& \text{if } end(\sigma_k) < \infty \text{ then } \mathcal{G}(\neg\bar{g})(\sigma_k^b.s) \text{ otherwise } \mathcal{G}(\bar{g})(\sigma_k^b.s)\} \\
\cup \{\sigma \mid{} & \text{there exists an infinite sequence of models } \sigma_1, \sigma_2, \ldots, \text{ such that} \\
& \sigma = \sigma_1\sigma_2\cdots, \sigma_1 \in \mathcal{M}(G)(b), \text{ for any } i \geq 2, \\
& \sigma_i \in \mathcal{M}(G)(Buf(b, \sigma_1 \cdots \sigma_{i-1})), \text{ for any } j \geq 1, \\
& end(\sigma_j) < \infty, \text{ and } \mathcal{G}(\bar{g})(\sigma_j^b.s)\}
\end{aligned}$$

Parallel Composition

In order to define the semantics of parallel composition, we first need a few definitions. The first definition expresses that the projection of a model σ onto a set of channel names *cset* at R-fields is the same as σ except that the new R-fields contain only those records for which the channel name belongs to *cset*.

Definition 3.2.12 (Projection onto Channels at R-Fields) Let $cset \subseteq CHAN$. Define the projection of a model σ onto $cset$ at R-fields, denoted by $[\sigma]^R_{cset}$, as follows:
$begin([\sigma]^R_{cset}) = begin(\sigma)$, $end([\sigma]^R_{cset}) = end(\sigma)$,
for any $\tau \in [begin(\sigma), end(\sigma)]$, $[\sigma]^R_{cset}(\tau).s = \sigma(\tau).s$, $[\sigma]^R_{cset}(\tau).S = \sigma(\tau).S$, and
$[\sigma]^R_{cset}(\tau).R = \{(c, \vartheta) \mid (c, \vartheta) \in \sigma(\tau).R \text{ and } c \in cset\}$.

The projection of a model σ onto a set of variables $vset$ is the same as σ except that if a variable does not belong to $vset$ then its value at all states is the same as its initial value in σ.

Definition 3.2.13 (Projection onto Variables) Let $vset \subseteq VAR$. Define the projection of a model σ onto $vset$, denoted by $\sigma \downarrow vset$, as follows:
$begin(\sigma \downarrow vset) = begin(\sigma)$, $end(\sigma \downarrow vset) = end(\sigma)$, for any $\tau \in [begin(\sigma), end(\sigma)]$,
$(\sigma \downarrow vset)(\tau).S = \sigma(\tau).S$, $(\sigma \downarrow vset)(\tau).R = \sigma(\tau).R$, and for any $x \in VAR$,

$$(\sigma \downarrow vset)(\tau).s(x) = \begin{cases} \sigma(\tau).s(x) & x \in vset \\ \sigma^b.s(x) & x \notin vset \end{cases}$$

The semantics of $S_1 \| S_2$ consists of all models σ for which there exist models $\sigma_1 \in \mathcal{M}(S_1)$ and $\sigma_2 \in \mathcal{M}(S_2)$ such that

- the S-fields of σ are the same as those of σ_1 and σ_2 because the S-fields contain the messages that have been sent in the whole system;
- the R-fields of the projection of σ onto $ich(S_i)$ at R-fields should be the same as the corresponding R-fields of σ_i;
- the value of a variable x during the execution of $S_1 \| S_2$ is obtained from the state of σ_i if $x \in var(S_i)$, and from the initial state otherwise, since $var(S_1) \cap var(S_2) = \emptyset$;
- if S_1 terminates before S_2, the S-fields of σ_2 should not contain any (c, ϑ) with $c \in och(S_1)$ and $\vartheta \in VAL$ after S_1 has terminated, because $c \in och(S_1)$ implies $c \notin och(S_2)$ and the environment of $S_1 \| S_2$ cannot send any message to c either. Similarly, for S_1 and S_2 interchanged. To express this property, we have the following predicate $Cons$.

Definition 3.2.14 For any statements S_1, S_2, and any models σ_1, σ_2,
$Cons(\sigma_1, \sigma_2, S_1, S_2)$ iff

- if $end(\sigma_1) \leq end(\sigma_2)$, then for any $c \in och(S_1)$, any $\vartheta \in VAL$, and any $\tau \in (end(\sigma_1), end(\sigma_2)]$, $(c, \vartheta) \notin \sigma_2(\tau).S$;
- if $end(\sigma_2) < end(\sigma_1)$, then for any $c \in och(S_2)$, any $\vartheta \in VAL$, and any $\tau \in (end(\sigma_2), end(\sigma_1)]$, $(c, \vartheta) \notin \sigma_1(\tau).S$.

The initial buffers of joint channels of S_1 and S_2 should not contain any message. Thus, given any initial buffer b,

- if there exists a $c \in ch(S_1) \cap ch(S_2)$ with $b(c) \neq \langle\rangle$, then $\mathcal{M}(S_1 \| S_2)(b) = \emptyset$;
- otherwise $\mathcal{M}(S_1 \| S_2)(b) =$
  $\{\sigma \mid ich(\sigma) \subseteq ich(S_1) \cup ich(S_2)$, for $i = 1, 2$, there exist $\sigma_i \in \mathcal{M}(S_i)(b)$ such that
  $begin(\sigma) = begin(\sigma_i)$, $end(\sigma) = max(end(\sigma_1), end(\sigma_2))$,
  for any $\tau_1 \in [begin(\sigma_i), end(\sigma_i)]$, $\sigma(\tau_1).S = \sigma_i(\tau_1).S$,
  $[\sigma]^R_{ich(S_i)}(\tau_1).R = \sigma_i(\tau_1).R$, $(\sigma \downarrow var(S_i))(\tau_1).s = \sigma_i(\tau_1).s$,
  for any $\tau_2 \in (end(\sigma_i), end(\sigma)]$, $[\sigma]^R_{ich(S_i)}(\tau_2).R = \emptyset$, $(\sigma \downarrow var(S_i))(\tau_2).s = \sigma_i^e.s$,
  for any $x \notin var(S_1) \cup var(S_2)$ and any $\tau \in [begin(\sigma), end(\sigma)]$,
  $\sigma(\tau).s(x) = \sigma^b.s(x) = \sigma_i^b.s(x)$,
  for any $c \in ch(S_1) \cap ch(S_2)$, $b(c) = \langle\rangle$, and $Cons(\sigma_1, \sigma_2, S_1, S_2)\}$

Similar to chapter 2, we also define a so-called well-formedness property of the semantics.

Definition 3.2.15 (Well-Formedness) A model σ, defined in section 3.2.1, is *well-formed* iff for any $c \in CHAN$, any τ, $begin(\sigma) \leq \tau \leq end(\sigma)$, and any $\vartheta_1, \vartheta_2 \in VAL$, the following holds:

- $(c, \vartheta_1) \in \sigma(\tau).R \wedge (c, \vartheta_2) \in \sigma(\tau).R \rightarrow \vartheta_1 = \vartheta_2$.
  (Uniqueness: at most one value is received on a channel at any time point.)

And then we also have the following theorem.

Theorem 3.2.1 For any process S and any buffer b, if $\sigma \in \mathcal{M}(S)(b)$, then

- $ich(\sigma) \subseteq ich(S)$,
- if $x \notin wvar(S)$, then for any τ, $begin(\sigma) \leq \tau \leq end(\sigma)$, $\sigma(\tau).s(x) = \sigma^b.s(x)$, and
- σ is well-formed.

This theorem can be easily proved, by induction on the structure of S.

3.3 Specification Language

We define a specification language which is based on Explicit Clock Temporal Logic, i.e., ordinary linear time temporal logic augmented with a global clock variable denoted by T. Intuitively, T refers to the current point of time during an execution. We use *start* and *term* to express the starting and terminating times of a computation respectively (*term* $= \infty$ for a nonterminating computation). We also use $first(x)$ and $init(c)$ to refer to the value of x at the first state of a computation and the initial buffer of channel c, respectively. Notice that $last(x)$ (from the specification language in chapter 2) is not needed here. To specify the communication behavior of processes, it is sufficient to use two primitives $send(c, vexp)$ and $receive(c, vexp)$, which express sending and receiving of expression $vexp$ along channel c, respectively. To abstract from values, we also use $send(c)$ and $receive(c)$. Similar to chapter 2, this specification language include the strong until operator, $\mathcal{U}$, the "chop" operator $\mathcal{C}$, and the "iterated chop" operator $\mathcal{C}^*$.

In this specification language, there are three kinds of expressions, i.e., $qexp$, $vexp$, and $texp$, to express values of type QUE, VAL, and $TIME \cup \{\infty\}$, respectively. A specification is denoted by φ. The syntax of this language is given in tabel 3.2, with $w \in QUE$, $c \in CHAN$, $\vartheta \in VAL$, $x \in VAR$, and $\hat{\tau} \in TIME \cup \{\infty\}$.

Table 3.2: Syntax of the Specification Language in Chapter 3

Que Exp	$qexp ::=$	$w \mid init(c)$
Val Exp	$vexp ::=$	$\vartheta \mid x \mid first(x) \mid first(qexp) \mid max(vexp_1, vexp_2) \mid$ $vexp_1 + vexp_2 \mid vexp_1 - vexp_2 \mid vexp_1 \times vexp_2$
Time Exp	$texp ::=$	$\hat{\tau} \mid T \mid start \mid term \mid vexp \mid$ $texp_1 + texp_2 \mid texp_1 - texp_2 \mid texp_1 \times texp_2$
Specification	$\varphi ::=$	$qexp_1 = qexp_2 \mid texp_1 = texp_2 \mid texp_1 < texp_2 \mid$ $send(c, vexp) \mid send(c) \mid receive(c, vexp) \mid receive(c) \mid$ $\varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \varphi_1\ \mathcal{U}\ \varphi_2 \mid \varphi_1\ \mathcal{C}\ \varphi_2 \mid \varphi_1\ \mathcal{C}^*\ \varphi_2$

Let exp be any expression from this specification language, i.e., exp can be some $qexp$ or $texp$. Define the input channels of exp, denoted by $ich(exp)$, to be the set of all channel names occurring in $init(c)$ in exp. Define the variables of exp, denoted by $var(exp)$, to be the set of all variables occurring in exp. Let φ be any specification. We define $ich(\varphi)$ to be the set of all channel names occurring in $init(c)$, $receive(c)$, or $receive(c, vexp)$ in φ, for some $vexp$. We also define $var(\varphi)$ to be the set of all variables occurring in φ.

Next we give the interpretation of this specification language. We first define the value of a sequence expression $qexp$ at model σ, initial buffer b, and time $\tau \geq begin(\sigma)$, $\tau \in TIME$, denoted by $\mathcal{Q}(qexp)(\sigma, b, \tau)$, as follows.

- $\mathcal{Q}(w)(\sigma, b, \tau) = w$
- $\mathcal{Q}(init(c))(\sigma, b, \tau) = b(c)$

The value of expression $vexp$ at model σ, initial buffer b, and time $\tau \geq begin(\sigma)$, $\tau \in TIME$, denoted by $\mathcal{V}(vexp)(\sigma, b, \tau)$, is defined as follows.

- $\mathcal{V}(\vartheta)(\sigma, b, \tau) = \vartheta$
- $\mathcal{V}(x)(\sigma, b, \tau) = \begin{cases} \sigma(\tau).s(x) & \text{if } \tau \leq end(\sigma) \\ \sigma^e.s(x) & \text{if } \tau > end(\sigma) \end{cases}$
- $\mathcal{V}(first(x))(\sigma, b, \tau) = \sigma^b.s(x)$
- $\mathcal{V}(first(qexp))(\sigma, b, \tau) = First(\mathcal{Q}(qexp)(\sigma, b, \tau))$
- $\mathcal{V}(max(vexp_1, vexp_2))(\sigma, b, \tau) = max(\mathcal{V}(vexp_1)(\sigma, b, \tau), \mathcal{V}(vexp_2)(\sigma, b, \tau))$
- $\mathcal{V}(vexp_1 \odot vexp_2)(\sigma, b, \tau) = \mathcal{V}(vexp_1)(\sigma, b, \tau) \odot \mathcal{V}(vexp_2)(\sigma, b, \tau)$, for $\odot \in \{+, -, \times\}$.

The value of a time expression $texp$ at model σ, initial buffer b, and time $\tau \geq begin(\sigma)$, $\tau \in TIME$, denoted by $\mathcal{T}(texp)(\sigma, b, \tau)$, is defined as follows.

- $\mathcal{T}(\hat{\tau})(\sigma, b, \tau) = \hat{\tau}$
- $\mathcal{T}(T)(\sigma, b, \tau) = \tau$
- $\mathcal{T}(start)(\sigma, b, \tau) = begin(\sigma)$
- $\mathcal{T}(term)(\sigma, b, \tau) = end(\sigma)$
- $\mathcal{T}(vexp)(\sigma, b, \tau) = \mathcal{V}(vexp)(\sigma, b, \tau)$
- $\mathcal{T}(vexp_1 \odot vexp_2)(\sigma, b, \tau) = \mathcal{T}(vexp_1)(\sigma, b, \tau) \odot \mathcal{T}(vexp_2)(\sigma, b, \tau)$, for $\odot \in \{+, -, \times\}$.

The interpretation of a specification φ at model σ, initial buffer b, and time $\tau \geq begin(\sigma)$, $\tau \in TIME$, denoted by $\langle \sigma, b, \tau \rangle \models \varphi$, is defined by induction on the structure of φ.

- $\langle \sigma, b, \tau \rangle \models qexp_1 = qexp_2$ iff $\mathcal{Q}(qexp_1)(\sigma, b, \tau) = \mathcal{Q}(qexp_2)(\sigma, b, \tau)$.
- $\langle \sigma, b, \tau \rangle \models texp_1 = texp_2$ iff $\mathcal{T}(texp_1)(\sigma, b, \tau) = \mathcal{T}(texp_2)(\sigma, b, \tau)$.
- $\langle \sigma, b, \tau \rangle \models texp_1 < texp_2$ iff $\mathcal{T}(texp_1)(\sigma, b, \tau) < \mathcal{T}(texp_2)(\sigma, b, \tau)$.
- $\langle \sigma, b, \tau \rangle \models send(c, vexp)$ iff $\tau \leq end(\sigma)$ and $(c, \mathcal{V}(vexp)(\sigma, b, \tau)) \in \sigma(\tau).S$.
- $\langle \sigma, b, \tau \rangle \models send(c)$ iff $\tau \leq end(\sigma)$ and there exists a $\vartheta \in VAL$ such that $(c, \vartheta) \in \sigma(\tau).S$.
- $\langle \sigma, b, \tau \rangle \models receive(c, vexp)$ iff $\tau \leq end(\sigma)$ and $(c, \mathcal{V}(vexp)(\sigma, b, \tau)) \in \sigma(\tau).R$.
- $\langle \sigma, b, \tau \rangle \models receive(c)$ iff $\tau \leq end(\sigma)$ and there exists a $\vartheta \in VAL$ such that $(c, \vartheta) \in \sigma(\tau).R$.

- $\langle\sigma, b, \tau\rangle \models \varphi_1 \vee \varphi_2$ iff $\langle\sigma, b, \tau\rangle \models \varphi_1$ or $\langle\sigma, b, \tau\rangle \models \varphi_2$.
- $\langle\sigma, b, \tau\rangle \models \neg\varphi$ iff not $\langle\sigma, b, \tau\rangle \models \varphi$.
- $\langle\sigma, b, \tau\rangle \models \varphi_1 \ \mathcal{U} \ \varphi_2$ iff there exists a $\tau_2 \geq \tau$, such that $\langle\sigma, b, \tau_2\rangle \models \varphi_2$, and for all $\tau_1, \tau \leq \tau_1 < \tau_2$, $\langle\sigma, b, \tau_1\rangle \models \varphi_1$.
- $\langle\sigma, b, \tau\rangle \models \varphi_1 \ \mathcal{C} \ \varphi_2$ iff
  - either $\langle\sigma, b, \tau\rangle \models \varphi_1$ and $end(\sigma) = \infty$,
  - or there exist models σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $\tau \leq end(\sigma_1) < \infty$, $\langle\sigma_1, b, \tau\rangle \models \varphi_1$, and $\langle\sigma_2, Buf(b, \sigma_1), begin(\sigma_2)\rangle \models \varphi_2$.
- $\langle\sigma, b, \tau\rangle \models \varphi_1 \ \mathcal{C}^* \ \varphi_2$ iff
  - either there exist a $k \geq 1$ and models $\sigma_1, \ldots, \sigma_k$ such that $\sigma = \sigma_1 \cdots \sigma_k$, $\tau \leq end(\sigma_1) < \infty$, $\langle\sigma_1, b, \tau\rangle \models \varphi_1$, for all i, $2 \leq i \leq k-1$, $end(\sigma_i) < \infty$, $\langle\sigma_i, b_i, begin(\sigma_i)\rangle \models \varphi_1$, if $end(\sigma_k) < \infty$ then $\langle\sigma_k, b_k, begin(\sigma_k)\rangle \models \varphi_2$, otherwise $\langle\sigma_k, b_k, begin(\sigma_k)\rangle \models \varphi_1$, and for all j, $2 \leq j \leq k$, $b_j = Buf(b, \sigma_1 \cdots \sigma_{j-1})$,
  - or there exist infinite models $\sigma_1, \sigma_2, \sigma_3, \ldots$ such that $\sigma = \sigma_1\sigma_2\sigma_3\ldots$, $end(\sigma_1) \geq \tau$, $\langle\sigma_1, b, \tau\rangle \models \varphi_1$, for all $i \geq 2$, $\langle\sigma_i, b_i, begin(\sigma_i)\rangle \models \varphi_1$ with $b_i = Buf(b, \sigma_1 \cdots \sigma_{i-1})$, and for all $j \geq 1$, $end(\sigma_j) < \infty$.

The substitution of an expression $vexp_1$ for a variable x in an expression $vexp_2$, denoted by $vexp_2[vexp_1/x]$, is defined as the expression obtained by replacing every occurrence of x in $vexp_2$ by $vexp_1$.

Moreover, we have the usual abbreviations from temporal logic, i.e., $\Diamond\varphi$, $\Box\varphi$, and $\varphi_1 \ \mathbf{U} \ \varphi_2$. Their definitions can be found in chapter 2 section 2.3.

Definition 3.3.1 (Valid Specification) A specification φ is *valid*, denoted by $\models \varphi$, iff $\langle\sigma, b, begin(\sigma)\rangle \models \varphi$ for any buffer b and any model σ.

To express that every computation of a process S satisfies an ECTL specification φ, we use a correctness formula of the form S **sat** φ.

Definition 3.3.2 (Satisfaction) A process S *satisfies* a specification φ, denoted by $\models S$ **sat** φ, iff $\langle\sigma, b, begin(\sigma)\rangle \models \varphi$ for any buffer b and any model $\sigma \in \mathcal{M}(S)(b)$.

The following are some examples of correctness formulae in this specification language.

- S never receives any message from channel c and never terminates:
  S **sat** $(\Box \neg receive(c)) \wedge term = \infty$.

- If S starts its execution with $x = 0$, S will eventually terminate and x will have value 10 at termination:
  S **sat** $first(x) = 0 \rightarrow \Diamond(T = term \wedge x = 10)$.

- If the initial buffer of channel c is empty and no message will be sent to channel c, then S never receives any message from c:
  S **sat** $(init(c) = \langle\rangle \wedge \Box\neg send(c)) \rightarrow \Box\neg receive(c)$.

- If the initial buffer of c is not empty, then S will eventually receive the first value of the buffer for channel c:
  S **sat** $init(c) \neq \langle\rangle \rightarrow \Diamond receive(c, first(init(c)))$.

3.4 Proof System

In this section, we give a compositional proof system for our programming language in section 3.1. Similarly to chapter 2, this proof system will include all valid assertions of ECTL as axioms. We first formulate some general axioms and then give axioms and rules for each statement from the programming language.

For any finite $cset \subseteq CHAN$ and finite $vset \subseteq VAR$, define
$norecv(cset) \equiv \bigwedge_{c \in cset} \neg receive(c)$, $nosend(cset) \equiv \bigwedge_{c \in cset} \neg send(c)$, and
$inv(vset) \equiv \bigwedge_{x \in vset} x = first(x)$.

The first axiom axiomatizes the well-formedness property of the semantics.

Axiom 3.4.1 (Well-Formedness)
For any finite $cset \subseteq CHAN$, S **sat** WF^A_{cset}, where

$WF^A_{cset} \equiv \bigwedge_{c \in cset} receive(c, vexp_1) \wedge receive(c, vexp_2) \rightarrow vexp_1 = vexp_2$.

The next axiom expresses that if a channel is not an input channel of statement S, S will never receive a message along that channel.

Axiom 3.4.2 (Receiving Invariance)
For any finite $cset \subseteq CHAN$ with $cset \cap ich(S) = \emptyset$, S **sat** $\Box norecv(cset)$.

The variable invariance axiom, the conjunction rule, and the consequence rule defined in chapter 2 are also included in the proof system.

The axioms for skip, assignment, and delay statements are the same as defined in chapter 2.

Statement $c!!e$ sends the value of e along channel c without waiting for its communication partner.

Axiom 3.4.3 (Send) $c!!e$ **sat** $\neg send(c) \;\mathcal{U}\; (T = term = start + K_c \wedge send(c, e))$

Statement $c??x$ reads the first value of the sequence of messages in the buffer of channel c. If there is no message available, it has to wait until a message arrives.
Let ψ be any specification. Define $Await(\psi) \equiv (\neg\psi)\ \mathbf{U}\ (\psi \wedge T = term)$.
We formulate an axiom for $c??x$ by using

$$WRecv(c??x) \equiv \Box\,[x = first(x) \wedge \neg receive(c)] \wedge Await[init(c) \neq \langle\rangle \vee send(c)]$$

and

$$Recv(c??x) \equiv [x = first(x) \wedge \neg receive(c)]\ \mathcal{U}$$
$$[T = term = start + K_c \wedge receive(c, x) \wedge x = first(init(c))]$$

Axiom 3.4.4 (Receive) $\quad c??x$ **sat** $WRecv(c??x)\ \mathcal{C}\ Recv(c??x)$

Sequential composition $S_1; S_2$ expresses a sequential execution of S_1 followed by S_2.
Let $\psi_1 \equiv \Box\, nosend(och(S_2) \setminus och(S_1))$ and $\psi_2 \equiv \Box\, nosend(och(S_1) \setminus och(S_2))$.
Then we have the following rule for sequential composition.

Rule 3.4.1 (Sequential Composition) $\quad \dfrac{S_1 \textbf{ sat } \varphi_1,\ S_2 \textbf{ sat } \varphi_2}{S_1; S_2 \textbf{ sat } (\varphi_1 \wedge \psi_1)\ \mathcal{C}\ (\varphi_2 \wedge \psi_2)}$

Recall that we have the following abbreviations (see section 3.2.2):

$G_1 \equiv [[]_{i=1}^{n} g_i \to S_i]$, $G_2 \equiv [[]_{i=1}^{n} g_i; c_i??x_i \to S_i\, []\, \textbf{delay}\ e \to S_0]$,

$\bar{g} \equiv \bigvee_{i=1}^{n} g_i$ for G_1, $\bar{g} \equiv \bigvee_{i=0}^{n} g_i$ for G_2, $\bar{c} \equiv \{c_i \mid g_i\}$ for G_2.

To axiomatize guarded commands, we define some additional abbreviations:

$Quiet(G_i) \equiv inv(wvar(G_i)) \wedge norecv(ich(G_i)) \wedge nosend(och(G_i))$, for $i = 1, 2$,

$Quiet(G_2 \setminus j) \equiv inv(wvar(G_2) \setminus \{x_j\}) \wedge norecv(ich(G_2) \setminus \{c_j\}) \wedge nosend(och(G_2))$,
for $j = 1, ..., n$,

and
$Eval \equiv term = start + K_g$.

First we give an axiom for the evaluation of guarded commands G_1 and G_2.

Axiom 3.4.5 (Guarded Command Evaluation) $\quad$ For $i = 1, 2$,

$$G_i \textbf{ sat } [Quiet(G_i)\ \mathcal{U}\ (T = start + K_g \wedge Quiet(G_i))] \wedge\ [\neg\bar{g} \to Eval]$$

Next we formulate a rule for G_1, by using
$Exec \equiv \bigvee_{i=1}^{n} g_i \wedge \varphi_i \wedge \Box\, nosend(och(G_1) \setminus och(S_i))$

Rule 3.4.2 (Guarded Command with Purely Boolean Guards)

$$\frac{S_i \ \textbf{sat} \ \varphi_i, \ \text{for } i = 1, \ldots, n}{[\|_{i=1}^{n} g_i \rightarrow S_i] \ \textbf{sat} \ \bar{g} \rightarrow (Eval \ \mathcal{C} \ Exec)}$$

For G_2, we use the following additional abbreviations:

$Wait \equiv \bar{g} \wedge Await[\bigvee_{1 \leq i \leq n} g_i \wedge (init(c_i) \neq \langle\rangle \vee send(c_i))] \wedge \ \Box\, Quiet(G_2)$

$Comm \equiv \bigvee_{i=1}^{n} g_i \wedge \varphi_i \wedge \Box\, nosend(och(G_2) \setminus och(S_i))$

$FinComm \equiv (g_0 \wedge term < start + max(0, e) \wedge Wait) \ \mathcal{C} \ Comm$

$TimeOut \equiv [g_0 \wedge \Box\, (\bigwedge_{c_i \in \bar{c}} init(c_i) = \langle\rangle \wedge \neg send(c_i)) \wedge term = start + max(0, e) \wedge$
$\Box\, Quiet(G_2)] \ \ \mathcal{C} \ \ [\varphi_0 \wedge \Box\, nosend(och(G_2) \setminus och(S_0))]$

$AnyComm \equiv (\neg g_0 \wedge Wait) \ \mathcal{C} \ Comm$

Rule 3.4.3 (Guarded Command with IO-Guards)

$$\frac{c_i??x_i; S_i \ \textbf{sat} \ \varphi_i, \quad \text{for } i = 1, \ldots, n, \quad S_0 \ \textbf{sat} \ \varphi_0}{\begin{array}{c}[\|_{i=1}^{n} g_i; c_i??x_i \rightarrow S_i \| g_0; \textbf{delay} \ e \rightarrow S_0] \ \textbf{sat} \\ \bar{g} \rightarrow (Eval \ \mathcal{C} \ (FinComm \vee Timeout \vee AnyComm))\end{array}}$$

Statement $\star G$ denotes repeated execution of G if one of those g_i in G is true. Its execution can be expressed by using the $\mathcal{C}^*$ operator.

Rule 3.4.4 (Iteration) $$\frac{G \ \textbf{sat} \ \varphi}{\star G \ \textbf{sat} \ (\bar{g} \wedge \varphi) \ \mathcal{C}^* \ (\neg \bar{g} \wedge \varphi)}$$

Next consider parallel composition of S_1 and S_2. Suppose we have specifications φ_1 and φ_2 for, respectively, S_1 and S_2. If S_1 terminates after (or at the same time as) S_2 then the model representing this computation of $S_1 \| S_2$ satisfies $\varphi_1 \wedge (\varphi_2 \ \mathcal{C} \ true)$. Furthermore we have to express that the variables of S_2 are not changed and there is no activity on the channels of S_2 after the termination of S_2. Similarly, for S_1 and S_2 interchanged.
Let $IBuf \equiv \bigwedge_{c \in ch(S_1) \cap ch(S_2)} init(c) = \langle\rangle$ and
$\psi_i \equiv \Box\, [inv(var(S_i)) \wedge norecv(ich(S_i)) \wedge nosend(och(S_i))]$, for $i = 1, 2$.

The parallel composition rule is formulated as follows.

Rule 3.4.5 (Parallel Composition)

$$\frac{S_1 \ \textbf{sat} \ \varphi_1, S_2 \ \textbf{sat} \ \varphi_2}{S_1 \| S_2 \ \textbf{sat} \ IBuf \wedge [(\varphi_1 \wedge (\varphi_2 \ \mathcal{C} \ \psi_2)) \ \vee \ (\varphi_2 \wedge (\varphi_1 \ \mathcal{C} \ \psi_1))]}$$

provided $ich(\varphi_i) \subseteq ich(S_i)$ and $var(\varphi_i) \subseteq var(S_i)$, for $i = 1, 2$.

Example 3.4.1 We prove that
$c??x \| c!!5$ **sat** $term = start + 2K_c \wedge \Box(T = term \to x = 5)$.

By the receive axiom, we have $c??x$ **sat** φ_1 with
$\varphi_1 \equiv WRecv(c??x)\ \mathcal{C}\ Recv(c??x)$, where
$WRecv(c??x) \equiv \Box[x = first(x) \wedge \neg receive(c)] \wedge Await[init(c) \neq \langle\rangle \vee send(c)]$ and
$Recv(c??x) \equiv [x = first(x) \wedge \neg receive(c)]\ \mathcal{U}$
$[T = term = start + K_c \wedge receive(c, x) \wedge x = first(init(c))]$.
By the send axiom, we have $c!!5$ **sat** φ_2 with
$\varphi_2 \equiv \neg send(c)\ \mathcal{U}\ (T = term = start + K_c \wedge send(c, 5))$.

Since $ich(\varphi_1) \subseteq ich(c??x)$, $ich(\varphi_2) \subseteq ich(c!!5)$, $var(\varphi_1) \subseteq var(c??x)$, and $var(\varphi_2) \subseteq var(c!!5)$, by the parallel composition rule, we have
$c??x \| c!!5$ **sat** $IBuf \wedge [(\varphi_1 \wedge (\varphi_2\ \mathcal{C}\ \psi_2)) \vee (\varphi_2 \wedge (\varphi_1\ \mathcal{C}\ \psi_1))]$
where
$IBuf \equiv init(c) = \langle\rangle$,
$\psi_1 \equiv \Box[inv(\{x\}) \wedge norecv(\{c\})]$, and
$\psi_2 \equiv \Box\, nosend(\{c\})$.
Observe that,
$IBuf \wedge \varphi_1 \wedge (\varphi_2\ \mathcal{C}\ \psi_2)$ is equivalent to
$init(c) = \langle\rangle \wedge [WRecv(c??x)\ \mathcal{C}\ Recv(c??x)] \wedge$
$[(\neg send(c)\ \mathcal{U}\ T = start + K_c \wedge send(c, 5))\ \mathcal{C}\ \Box\, nosend(\{c\})]$,
which implies
$[(\neg send(c) \wedge init(c) = \langle\rangle)\ \mathcal{U}\ (T = term = start + K_c \wedge send(c, 5))]\ \mathcal{C}$
$[(x = first(x) \wedge \neg receive(c))\ \mathcal{U}\ (T = term = start + K_c \wedge receive(c, x) \wedge$
$x = first(init(c)))]$,
and this leads to
$term = start + 2K_c \wedge \Box(T = term \to x = 5)$.
Furthermore, we have that,
$IBuf \wedge \varphi_2 \wedge (\varphi_1\ \mathcal{C}\ \psi_1)$ implies
$[\neg send(c)\ \mathcal{U}\ (T = term = start + K_c \wedge send(c, 5))] \wedge$
$[WRecv(c??x)\ \mathcal{C}\ Recv(c??x)\ \mathcal{C}\ \Box\, norecv(\{c\})]$,
which implies
$term = start + K_c\ \wedge [\Diamond(T = term = start + K_c \wedge send(c, 5))\ \mathcal{C}$
$\Diamond(T = term = start + K_c \wedge receive(c, x))\ \mathcal{C}\ \Box\, norecv(\{c\})]$,
and this leads to
$term = start + K_c \wedge term \geq start + 2K_c$,
which leads to $false$.
Combining these two cases, we obtain

$IBuf \wedge [(\varphi_1 \wedge (\varphi_2\, C\, \psi_2)) \vee (\varphi_2 \wedge (\varphi_1\, C\, \psi_1))] \rightsquigarrow term = start + 2K_c \wedge \Box\, (T = term \rightarrow x = 5)$.
Hence, by the consequence rule,
$c??x \| c!!5$ **sat** $term = start + 2K_c \wedge \Box\, (T = term \rightarrow x = 5)$. □

3.5 Soundness and Completeness

In this section, we discuss the soundness and completeness of the proof system in section 3.4. Regarding the soundness of the proof system, we must show that every formula S **sat** φ derivable in the proof system is indeed valid. We first give some lemmas which will be used to prove the soundness. These lemmas can be proved similarly as in Appendix A for those lemmas in chapter 2 section 2.6. The proofs for some new or modified lemmas can be found in Appendix D.

Lemma 3.5.1 For any expression e from the programming language, any model σ, any buffer b, and any $\tau \geq begin(\sigma)$, $\mathcal{E}(e)(\sigma(\tau).s) = \mathcal{V}(e)(\sigma, b, \tau)$.

Lemma 3.5.2 For any boolean guard g from the programming language, any model σ, any buffer b, and any $\tau \geq begin(\sigma)$, $\mathcal{G}(g)(\sigma(\tau).s)$ iff $\langle \sigma, b, \tau \rangle \models g$.

Lemma 3.5.3 For any expression $qexp$ of type QUE, any $cset \subseteq CHAN$, and any buffers b_1 and b_2, if $ich(qexp) \subseteq cset$ and for any $c \in cset$, $b_1(c) = b_2(c)$, then for any model σ and any $\tau \geq begin(\sigma)$, $\mathcal{Q}(qexp)(\sigma, b_1, \tau) = \mathcal{Q}(qexp)(\sigma, b_2, \tau)$.

Lemma 3.5.4 For any expression $qexp$ of type QUE, any model σ, any buffer b, any $cset \subseteq CHAN$, and any $\tau \geq begin(\sigma)$, $\mathcal{Q}(qexp)(\sigma, b, \tau) = \mathcal{Q}(qexp)([\sigma]^R_{cset}, b, \tau)$.

Lemma 3.5.5 For any expression $qexp$ of type QUE, any model σ, any buffer b, any $vset \subseteq VAR$, and any $\tau \geq begin(\sigma)$, $\mathcal{Q}(qexp)(\sigma, b, \tau) = \mathcal{Q}(qexp)(\sigma \downarrow vset, b, \tau)$.

Lemma 3.5.6 For any expression $vexp$ of type VAL, any $cset \subseteq CHAN$, and any buffers b_1 and b_2, if $ich(vexp) \subseteq cset$ and for any $c \in cset$, $b_1(c) = b_2(c)$, then for any model σ and any $\tau \geq begin(\sigma)$, $\mathcal{V}(vexp)(\sigma, b_1, \tau) = \mathcal{V}(vexp)(\sigma, b_2, \tau)$.

Lemma 3.5.7 For any expression $vexp$ of type VAL, any model σ, any buffer b, any $cset \subseteq CHAN$, and any $\tau \geq begin(\sigma)$, $\mathcal{V}(vexp)(\sigma, b, \tau) = \mathcal{V}(vexp)([\sigma]^R_{cset}, b, \tau)$.

Lemma 3.5.8 For any expression $vexp$ of type VAL, any model σ, any buffer b, any $vset \subseteq VAR$, and any $\tau \geq begin(\sigma)$, if $var(vexp) \subseteq vset$, then $\mathcal{V}(vexp)(\sigma, b, \tau) = \mathcal{V}(vexp)(\sigma \downarrow vset, b, \tau)$.

Lemma 3.5.9 For any expression $texp$ of type $TIME$, any $cset \subseteq CHAN$, and any buffers b_1 and b_2, if $ich(vexp) \subseteq cset$ and for any $c \in cset$, $b_1(c) = b_2(c)$, then for any model σ and any $\tau \geq begin(\sigma)$, $\mathcal{T}(texp)(\sigma, b_1, \tau) = \mathcal{T}(texp)(\sigma, b_2, \tau)$.

Lemma 3.5.10 For any expression $texp$ of type $TIME$, any model σ, any buffer b, any $cset \subseteq CHAN$, and any $\tau \geq begin(\sigma)$, $\mathcal{T}(texp)(\sigma, b, \tau) = \mathcal{T}(texp)([\sigma]^R_{cset}, b, \tau)$.

Lemma 3.5.11 For any expression $texp$ of type $TIME$, any model σ, any buffer b, any $vset \subseteq VAR$, and any $\tau \geq begin(\sigma)$, if $var(texp) \subseteq vset$, then $\mathcal{T}(texp)(\sigma, b, \tau) = \mathcal{T}(texp)(\sigma \downarrow vset, b, \tau)$.

Lemma 3.5.12 For any specification φ, any $cset \subseteq CHAN$, and any buffers b_1 and b_2, if $ich(\varphi) \subseteq cset$ and for any $c \in cset$, $b_1(c) = b_2(c)$, then for any model σ and any $\tau \geq begin(\sigma)$, $\langle \sigma, b_1, \tau \rangle \models \varphi$ iff $\langle \sigma, b_2, \tau \rangle \models \varphi$.

Lemma 3.5.13 For any $cset \subseteq CHAN$ and any specification φ, if $ich(\varphi) \subseteq cset$, then for any model σ, any buffer b, and any $\tau \geq begin(\sigma)$, $\langle \sigma, b, \tau \rangle \models \varphi$ iff $\langle [\sigma]^R_{cset}, b, \tau \rangle \models \varphi$.

Lemma 3.5.14 For any $vset \subseteq VAR$ and any specification φ, if $var(\varphi) \subseteq vset$, then for any model σ, any buffer b, and any $\tau \geq begin(\sigma)$, $\langle \sigma, b, \tau \rangle \models \varphi$ iff $\langle \sigma \downarrow vset, b, \tau \rangle \models \varphi$.

For the soundness of this proof system, we have the following theorem.

Theorem 3.5.1 (Soundness) The proof system in section 3.4 is sound.

To formally prove this theorem, we have to show that all axioms are valid and all inference rules preserve validity. For most axioms and inference rules, the soundness can be proved similarly as in Appendix B for the proof system in chapter 2, i.e., by following the definitions of the semantics and given lemmas. In Appendix E, we only give the soundness proofs for receiving invariance, send, receive, sequential composition, and parallel composition.

Similarly to chapter 2, we only prove the relative completeness of the proof system in section 3.4, i.e., every valid specification is derivable in the proof system, provided that any valid ECTL formula is provable.

We give a few lemmas which will be used for the completeness proof. These lemmas can be proved similarly as in Appendix A for lemmas from chapter 2.

Lemma 3.5.15 For any model σ and any $cset \subseteq DCHAN$, $ich(\sigma) \subseteq cset$ iff $\sigma = [\sigma]^R_{cset}$.

Lemma 3.5.16 For any model σ, any buffer b, and any $cset_1, cset_2 \subseteq DCHAN$, if $\langle \sigma, b, begin(\sigma) \rangle \models \Box\, norecv(cset_2 \setminus cset_1)$, then $[\sigma]^R_{cset_1 \cup cset_2} = [\sigma]^R_{cset_1}$.

Lemma 3.5.17 For any model σ, any buffer b, and any $vset_1, vset_2 \subseteq VAR$, if $\langle \sigma, b, begin(\sigma)\rangle \models \Box\, inv(vset_2 \setminus vset_1)$, then $\sigma \downarrow (vset_1 \cup vset_2) = \sigma \downarrow vset_1$.

Lemma 3.5.18 For any model σ, any buffer b, if $ich(\sigma) \subseteq cset$ and $\langle \sigma, b, begin(\sigma)\rangle \models WF^A_{cset}$, then σ is well-formed.

Similar to chapter 2, we prove the relative completeness by using a property of specifications called *preciseness*.

Definition 3.5.1 (Invariant Variable) A variable x is *invariant* with respect to a model σ iff for any τ, $begin(\sigma) \leq \tau \leq end(\sigma)$, $\sigma(\tau).s(x) = \sigma^b.s(x)$.

Notice that although this definition is the same as definition 2.6.1, they refer to different computational models.

Definition 3.5.2 (Preciseness) A specification φ is *precise* for a statement S of the programming language in section 3.1 iff

1. S **sat** φ holds, i.e., $\langle \sigma, b, begin(\sigma)\rangle \models \varphi$, for any buffer b and any $\sigma \in \mathcal{M}(S)(b)$;
2. For any buffer b and any well-formed model σ, if $ich(\sigma) \subseteq ich(S)$, any variable $x \notin wvar(S)$ is invariant with respect to σ, and $\langle \sigma, b, begin(\sigma)\rangle \models \varphi$, then $\sigma \in \mathcal{M}(S)(b)$; and
3. $ich(\varphi) = ich(S)$ and $var(\varphi) = var(S)$.

A precise specification φ for S thus characterizes all possible computations of S: φ is valid for S, and any "reasonable" computation satisfying φ is a possible computation of S.

In Theorem 3.5.2, we first show that for any statement S a precise specification can be derived from the proof system. Then, in Theorem 3.5.3, we prove that any specification φ_2 which is valid for S can be derived from a precise specification φ_1 for S and two other predicates. Hence, in Theorem 3.5.4, relative completeness is proved easily.

Theorem 3.5.2 If S is a statement from section 3.1, then a precise specification for S can be derived by using the proof system in section 3.4.

This theorem can be proved similarly as in Appendix C for theorem 2.6.2. In Appendix F we give a precise specification for each statement from section 3.1.

Theorem 3.5.3 If φ_1 is precise for S and φ_2 is valid for S, then $\models [\varphi_1 \wedge WF^A_{ich(\varphi_1)} \wedge \Box\, [norecv(ich(\varphi_2) \setminus ich(\varphi_1)) \wedge inv(var(\varphi_2) \setminus var(\varphi_1))]] \to \varphi_2$.

Proof: Let φ_1 be precise for S and φ_2 be valid for S. Consider a model σ and a buffer b. Assume that $\langle \sigma, b, begin(\sigma)\rangle \models \varphi_1 \wedge \Box\, [norecv(ich(\varphi_2) \setminus ich(\varphi_1)) \wedge inv(var(\varphi_2) \setminus var(\varphi_1))]$ holds. We prove $\langle \sigma, b, begin(\sigma)\rangle \models \varphi_2$.

By $\langle \sigma, b, begin(\sigma)\rangle \models \varphi_1$, lemma 3.5.13 leads to $\langle [\sigma]^R_{ich(\varphi_1)}, b, begin(\sigma)\rangle \models \varphi_1$. By lemma 3.5.14, $\langle [\sigma]^R_{ich(\varphi_1)} \downarrow var(\varphi_1), b, begin(\sigma)\rangle \models \varphi_1$. From $\langle \sigma, b, begin(\sigma)\rangle \models WF^A_{ich(\varphi_1)}$, by lemma 3.5.13, we have $\langle [\sigma]^R_{ich(\varphi_1)}, b, begin(\sigma)\rangle \models WF^A_{ich(\varphi_1)}$. By lemma 3.5.18, $[\sigma]^R_{ich(\varphi_1)}$ is well-formed. Then by definition, $[\sigma]^R_{ich(\varphi_1)} \downarrow var(\varphi_1)$ is also well-formed. Since φ_1 is precise for S, we have $ich(\varphi_1) = ich(S)$ and $var(\varphi_1) = var(S)$. By the definition of projection onto variables, any variable $x \notin wvar(S)$ is invariant with respect to $[\sigma]^R_{ich(\varphi_1)} \downarrow var(\varphi_1)$. Hence by the definition of preciseness, $[\sigma]^R_{ich(\varphi_1)} \downarrow var(\varphi_1) \in \mathcal{M}(S)$.
From $\langle \sigma, b, begin(\sigma)\rangle \models \Box\, norecv(ich(\varphi_2) \setminus ich(\varphi_1))$, lemma 3.5.16 leads to
$[\sigma]^R_{ich(\varphi_1)\cup ich(\varphi_2)} = [\sigma]^R_{ich(\varphi_1)}$. Since $\langle \sigma, b, begin(\sigma)\rangle \models \Box\, inv(var(\varphi_2) \setminus var(\varphi_1))$, lemma 3.5.17 leads to $\sigma \downarrow (var(\varphi_1) \cup var(\varphi_2)) = \sigma \downarrow var(\varphi_1)$. Thus we obtain
$[\sigma]^R_{ich(\varphi_1)\cup ich(\varphi_2)} \downarrow (var(\varphi_1) \cup var(\varphi_2)) = [\sigma]^R_{ich(\varphi_1)} \downarrow var(\varphi_1)$. Therefore we have
$[\sigma]^R_{ich(\varphi_1)\cup ich(\varphi_2)} \downarrow (var(\varphi_1) \cup var(\varphi_2)) \in \mathcal{M}(S)$. Since φ_2 is valid for S, we obtain
$\langle [\sigma]^R_{ich(\varphi_1)\cup ich(\varphi_2)} \downarrow (var(\varphi_1) \cup var(\varphi_2)), b, begin(\sigma)\rangle \models \varphi_2$. From $var(\varphi_2) \subseteq var(\varphi_1) \cup var(\varphi_2)$, lemma 3.5.14 leads to $\langle [\sigma]^R_{ich(\varphi_1)\cup ich(\varphi_2)}, b, begin(\sigma)\rangle \models \varphi_2$. By $ich(\varphi_2) \subseteq (ich(\varphi_1) \cup ich(\varphi_2))$, lemma 3.5.13 leads to $\langle \sigma, b, begin(\sigma)\rangle \models \varphi_2$. Hence this theorem holds. □

Theorem 3.5.4 (Relative Completeness) The proof system in section 3.4 is relatively complete.

Proof: For any process S, assume that specification φ is valid for S. We prove that S **sat** φ is derivable in the proof system in section 3.4. By theorem 3.5.2, we have S **sat** φ_1 where φ_1 is a precise specification for S. By the axiom 3.4.1, we have S **sat** $WF^A_{ich(\varphi_1)}$. Since $ich(\varphi_1) = ich(S)$, we have $[ich(\varphi) \setminus ich(\varphi_1)] \cap ich(S) = \emptyset$. Then by the receiving invariance axiom, we obtain S **sat** $\Box\, norecv(ich(\varphi) \setminus ich(\varphi_1))$. From $var(\varphi_1) = var(S)$, we have $[var(\varphi) \setminus var(\varphi_1)] \cap var(S) = \emptyset$ and thus $[var(\varphi) \setminus var(\varphi_1)] \cap wvar(S) = \emptyset$. By the variable invariance axiom, we obtain
S **sat** $\Box\, inv(var(\varphi) \setminus var(\varphi_1))$. Then the conjunction rule and the consequence rule lead to S **sat** $\varphi_1 \wedge WF^A_{ich(\varphi_1)} \wedge \Box\, [norecv(ich(\varphi) \setminus ich(\varphi_1)) \wedge inv(var(\varphi) \setminus var(\varphi_1))]$. By theorem 3.5.3, $[\varphi_1 \wedge WF^A_{ich(\varphi_1)} \wedge \Box\, [norecv(ich(\varphi) \setminus ich(\varphi_1)) \wedge inv(var(\varphi) \setminus var(\varphi_1))]] \rightarrow \varphi$ is valid and, by our relative completeness assumption, provable. Hence, by the consequence rule, S **sat** φ is derivable in the proof system in section 3.4. □

Chapter 4

Atomic Broadcast Protocol

4.1 Introduction

Computing systems are composed of hardware and software components which can fail. Component failures can lead to unanticipated behaviour and unavailability of service. To achieve a high availability of a service despite the presence of faults, a key idea is to implement the service by replicating a server process on all processors [Cri90]. Replication of service state information among group members enables the group to provide the service even when some of its members fail, since the remaining members have enough information about the service state to continue to provide it. To maintain the consistency of these replicated global states, any state update must be broadcast to all correct servers such that all these servers observe the same sequence of state updates. Thus a communication service is needed so that client processes can use it to deliver updates to their peers. This communication service is called *atomic* or *reliable* broadcast. We will refer to it as *atomic broadcast.* There are two sets of atomic broadcast protocols: *synchronous* ones, such as [BD85,CASD85], and [Cri90], and *asynchronous* ones, such as [BJ87] and [CM84].

Synchronous atomic broadcast protocols assume that the underlying communication delays between correct processors are bounded. Given this assumption, local clocks of correct processors can be synchronized [CAS86]. Then the properties of synchronous atomic broadcast protocols are described in terms of local clocks as follows [CASD85, CASD89]:

- Termination: every update whose broadcast is initiated by a correct processor at time T on its clock is delivered by all correct processors at time $T + \Delta$ on their own clocks, where Δ is a positive constant and is called the *broadcast termination time.*

- Atomicity: if a correct processor delivers an update at time U on its clock, then that update was initiated by some processor and is delivered by each correct processor at time U on its clock.

- Order: all correct processors deliver their updates in the same order.

Synchronous atomic broadcast protocols provide an upper bound for the broadcast termination time. Thus they can be used in real-time applications where deadlines must always be met, even in the presence of faults. On the other hand, asynchronous broadcast protocols do not assume bounded message transmission delays between correct processors. Thus they cannot guarantee a bound for the broadcast termination time. Therefore asynchronous atomic broadcast protocols are not suitable for critical real-time applications.

We are interested in the formal specification and verification of real-time and fault-tolerant systems. Since atomic broadcast service is one of the fundamental issues in fault-tolerance, we choose an atomic broadcast protocol as our case study.

An informal description of an atomic broadcast protocol, an implementation, and an informal proof which shows that the implementation indeed satisfies the requirement of this protocol are presented in [CASD85,CASD89]. In these papers, there is a series of protocols each of which tolerates omission failures, timing failures, and authentication-detectable byzantine failures. As a starting point of verifying real-time and fault-tolerant systems, we choose a fairly simple protocol which tolerates omission failures. Henceforth, we use the term *atomic broadcast protocol* to refer to this protocol. We will follow the ideas of [CASD89] as closely as possible and compare our results with it in section 4.8.

The atomic broadcast service is implemented by replicating a server process on all distributed processors in a network. Thus any client process on any processor can use this service. We allow more than one client process located on one processor. Assume that there are n processors in the network. Pairs of processors are connected by links which are point-to-point, bi-directional, communication channels. A processor (link) is correct if and only if it behaves as specified. In the atomic broadcast protocol, it is assumed that only omission failures occur on processors and links. When a processor suffers an omission failure, it cannot send messages to other processors. When a link suffers an omission failure, the messages traveling along this link may be lost. But those messages received by a processor are correct in time and contents. It is also assumed that the duration of message transmission between correct processors takes finite time and local clocks of correct processors are approximately synchronized. To send an update to its peers, a client process initiates the atomic broadcast server process located on the same processor to atomically broadcast that update. After such a request, each server process will deliver that update to the client processes located on the same processor. To achieve the order property of the service, there is a priority ordering among all processors. If two updates are initiated at different clock times, they will be delivered according to the ordering of their initiation times. If they are initiated at the same clock time on different processors, they will be delivered according to the priority of their initiation processors. The configuration of the service is illustrated in the following figure 4.1.

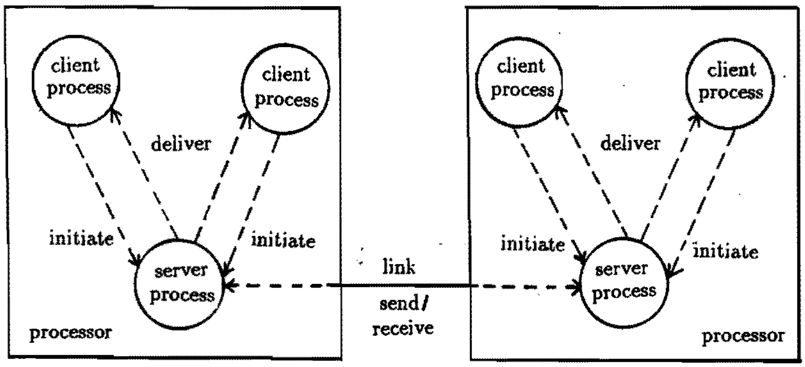

Fig. 4.1 Atomic Broadcast Service Configuration

In general, to formally verify a system, we need a proof theory which consists of axioms and rules about the system components. To be able to abstract from implementation details, it is often convenient to have a compositional verification method. Compositionality enables us to verify a system by using only specifications of its components without knowing any internal information of those components. In particular, if the system is composed of parallel components, the proof method should contain a *parallel composition rule*. Let $S(p)$ denote the atomic broadcast server process running on processor p, φ denote a specification written in a specification language based on first-order logic, and $S(p)$ **sat** φ denote that server process $S(p)$ satisfies specification φ. The parallel composition rule states that if server process $S(p_i)$ satisfies specification φ_i and φ_i only refers to the interface of p_i, i.e., φ_i and φ_j do not interfere with each other, for any $i, j = 1, 2, \ldots, n$ and $i \neq j$, then parallel execution of $S(p_i)$ satisfies the conjunction of the φ_i. This rule can be formalized as follows.

Parallel Composition Rule

$$\frac{S(p_i)\ \textbf{sat}\ \varphi_i,\ \varphi_i \text{ only refers to the interface of } p_i, \text{ for } i = 1, 2, \ldots, n}{S(p_1) \| \cdots \| S(p_n)\ \textbf{sat}\ \bigwedge_{i=1}^{n} \varphi_i}$$

To prove that a component satisfies a weaker specification, we need a *consequence rule*. Namely, if process S satisfies φ and φ implies ψ, then S also satisfies ψ.

Consequence Rule $$\frac{S\ \textbf{sat}\ \varphi,\ \varphi \rightarrow \psi}{S\ \textbf{sat}\ \psi}$$

Another useful rule is the *conjunction rule*, which shows that if process S satisfies φ_1 and φ_2, then S also satisfies $\varphi_1 \wedge \varphi_2$.

Conjunction Rule $$\frac{S \ \mathbf{sat}\ \varphi_1,\ S\ \mathbf{sat}\ \varphi_2}{S\ \mathbf{sat}\ \varphi_1 \wedge \varphi_2}$$

Recall that local clocks of correct processors are approximately synchronized. We show that the verification of the protocol can be done compositionally by using specifications in which timing is expressed by local clock values as follows.

- In section 4.2, we specify the properties of the atomic broadcast protocol in a specification language based on first-order logic. We call this the *top-level specification* and denote it by ABS. Thus our aim is to prove $S(p_1)\| \cdots \|S(p_n)$ **sat** ABS.

- In section 4.3, we axiomatize the required assumptions about the service configuration, including underlying communication mechanism, clock synchronization assumption, and failure assumptions. We denote the conjunction of all these axioms by AX.

- In section 4.4, we define the properties of the atomic broadcast server process running on processor p. We call this the *server process specification* and denote it by $Spec(p)$. The specification $Spec(p)$ should only refer to the interface of processor p. We assume $S(p)$ **sat** $Spec(p)$.

- By the parallel composition rule, we obtain $S(p_1)\| \cdots \|S(p_n)$ **sat** $\bigwedge_{i=1}^{n} Spec(p_i)$. Since $S(p_1)\| \cdots \|S(p_n)$ also satisfies AX, we prove, in section 4.5, 4.6, and 4.7, that
  $\bigwedge_{i=1}^{n} Spec(p_i) \ \wedge \ AX \ \rightarrow \ ABS$.
  Hence the consequence rule leads to $S(p_1)\| \cdots \|S(p_n)$ **sat** ABS.

- We compare our results with [CASD89] in section 4.8.

4.2 Top-Level Specification

We formalize the top-level requirements of the atomic broadcast protocol in this section.

Let P be a set of processor names and L a set of link names. We assume that all processors and links have unique names. We use $p, q, r, s, \ldots$ to denote elements of P and $l, l_1, \ldots$ to denote elements of L. Let G be the network of processors and links, i.e., $G = P \cup L$.

We assume that all real times range over a dense time domain called $RTIME$ and the standard arithmetic operators $+$, $-$, $\times$, and $\leq$ are defined on $RTIME$. We use lower case letters, e.g. t, u, v, …, to denote variables ranging over $RTIME$.

Each processor has access to a local clock. We denote by C_p a function which represents the value of the local clock of processor p, i.e., $C_p(t)$ is the value of the local clock of p at real time t. Let all clock values range over a domain called $CVAL$. We assume that, for any $T \in CVAL$, $T \geq 0$. Similarly, the standard arithmetic operators $+$, $-$, $\times$, and $\leq$ are defined on $CVAL$. We use capital letters, e.g. T, U, V, ..., to denote variables ranging over $CVAL$. We also use $[U, V]$, $[U, V)$, $(U, V]$, and (U, V) to express, respectively, closed, half-open, and open intervals of clock values.

The atomic broadcast service is implemented by a group of server processes replicated on all processors in the network. When a client process initiates a server process running on processor p by sending a request of broadcasting update σ, we call p the initiator of σ, i.e., we interpret it as *p initiates σ*. Similarly, when the server process delivers an update σ to client processes, we interpret it as *p delivers σ to client processes.*

To formally describe the properties of the atomic broadcast protocol, we define the following primitives:

- $correct(p)$ **at** t: processor p is correct at real time t, i.e., no omission failure occurs on p at real time t.

- $correct(l)$ **at** t: link l is correct at real time t, i.e., no omission failure occurs on l at real time t.

- $initiate(p, \sigma)$ **at** t: processor p finishes with receiving a request of broadcasting update σ from a client process located on p at real time t, i.e., p initiates σ at real time t.

- $deliver(p, \sigma)$ **at** t: processor p starts to send update σ to client processes located on p at real time t.

Henceforth, we use the following abbreviations:

- $correct(p) \equiv \forall t : correct(p)$ **at** t

- $correct(l) \equiv \forall t : correct(l)$ **at** t

In [CASD89], local clock values are used to express and reason about the properties of the protocol. We would also like to use local clock values to describe and verify the protocol. For any primitive φ **at** t, we define the following abbreviations:

- $\varphi\ \mathbf{at_p}\ T \equiv \exists t : \varphi\ \mathbf{at}\ t \wedge C_p(t) = T$

- $\varphi\ \mathbf{by_p}\ T \equiv \exists T_0 : \varphi\ \mathbf{at_p}\ T_0 \wedge T_0 \leq T$

- $\varphi\ \mathbf{before_p}\ T \equiv \exists T_0 : \varphi\ \mathbf{at_p}\ T_0 \wedge T_0 < T$

- $\varphi\ \mathbf{in_p}\ I \equiv \exists T \in I : \varphi\ \mathbf{at_p}\ T$, where $I \subseteq CVAL$.

In [CASD89], assumptions about the system are simplified. For instance, it is assumed that message processing time on a correct processor is zero. In this paper, we will take all possible times spent by a correct processor into account. Then the termination and atomicity properties can only be described by using an upper bound and an interval, respectively, instead of precise time points as in [CASD89].

4.2.1 Termination

The property of termination is stated as follows: every update whose broadcast is initiated by a correct processor s at clock value T will be delivered at all correct processors by clock value $T + D_1$ on their own clocks, where D_1 is a positive constant and is also the broadcast termination time.

In this paper, we take the convention that any free variable occurring in a formula is universally, outermostly, quantified. Thus the termination property is formally expressed as follows:

$$TERM \equiv correct(s) \land correct(q) \land initiate(s, \sigma)\ \mathbf{at_s}\ T \to deliver(q, \sigma)\ \mathbf{by_q}\ T + D_1$$

4.2.2 Atomicity

The atomicity property is described as follows: if a correct processor p delivers an update at clock value U, then that update was initiated by some processor s at some local time T and is delivered by all correct processors at some local clock value between $U - D_2$ and $U + D_2$, where D_2 is a positive constant and indicates the difference of delivery times of an update by two correct processors.

This property is formalized as follows:

$$\begin{aligned} ATOM \equiv\ & correct(p) \land correct(q) \land deliver(p, \sigma)\ \mathbf{at_p}\ U \to \\ & \exists s, T : initiate(s, \sigma)\ \mathbf{at_s}\ T \land deliver(q, \sigma)\ \mathbf{in_q}\ [U - D_2, U + D_2] \end{aligned}$$

The atomicity property claims that if any correct processor delivers an update σ at time U on its clock, then every correct processor will deliver that update at more or less the same time on its own clock, while the initiator of that update might happen to be correct at the initiation time . This is the difference with the termination property.

4.2.3 Order

The property of order is expressed in [CASD89] as follows: all correct processors deliver their updates in the same order.

Intuitively, we understand the order property as follows. Let U be any clock value. If a sequence of updates delivered by processor p before local time U is $\langle\sigma_1,\ldots,\sigma_k\rangle$, then there should exist a clock value V such that $\langle\sigma_1,\ldots,\sigma_k\rangle$ has also been delivered by any other processor q before local time V. Notice that U and V can be different. Furthermore, there is no reason to exclude the possibility that more than one update is delivered at the same time by a processor. Therefore the *set* of sequences of updates should include all possible sequences of updates in which those updates which are delivered at the same time are interleaved.

We define the following abbreviation:

- $\neg deliver(p)$ $\mathbf{in_p}$ $I \equiv \neg\exists\sigma : deliver(p,\sigma)$ $\mathbf{in_p}$ I.

Let $I\!N$ denote the set of all natural numbers (including 0). Let $I\!N^+ = I\!N \setminus \{0\}$. We define $List(p,U)$ to be the set of all possible sequences of updates delivered by p before local time U as follows.

Definition 4.2.1 For any processor p and any clock value $U \in CVAL$, define
$List(p,U) = \{\langle\sigma_1,\sigma_2,\ldots,\sigma_k\rangle \mid$ there exist $k \in I\!N^+$, $U_1,U_2,\ldots,U_k \in CVAL$ such that
$U_1 \leq U_2 \leq \ldots \leq U_k < U$, $deliver(p,\sigma_i)$ $\mathbf{at_p}$ U_i,
for all $i = 1,2,\ldots,k$, $\neg deliver(p)$ $\mathbf{in_p}$ (U_j,U_{j+1}),
for all $j = 1,2,\ldots,k-1$, and $\neg deliver(p)$ $\mathbf{in_p}$ $[0,U_1).\}$

If we can prove that, for any two correct processors p and q and any clock value U, there exists a clock value V such that $List(p,U)$ is a subset of $List(q,V)$, then symmetrically we can also prove that for any U there exists a V such that $List(q,U) \subseteq List(p,V)$. Hence p and q deliver their updates in the same order. Then the order property is formalized as follows:

$ORDER \equiv correct(p) \wedge correct(q) \rightarrow \forall U \exists V : List(p,U) \subseteq List(q,V)$

Notice that, by the definition of $ORDER$, if p delivers σ_1 and σ_2 at some clock value U_1, then q also delivers σ_1 and σ_2 at some clock value V_1, although U_1 and V_1 can be different.

The top-level specification of the protocol is the conjunction of these three properties. Recall that ABS denotes the top-level specification of the atomic broadcast protocol. Thus,

$ABS \equiv TERM \wedge ATOM \wedge ORDER.$

4.3 System Assumptions

In this section, we axiomatize the assumptions about the system.

4.3.1 Processors and Links

We define the following primitive for a link l.

- $link(l, p, q)$: l is a physical communication channel between p and q.

Definition 4.3.1 Define $Link(p)$ as the set of links each of which connects p with another processor: $Link(p) = \{l \mid \exists q : link(l, p, q)\}$.

For any p, q, and l, if $l \in Link(p)$, $l \in Link(q)$, and $p \not\equiv q$, then p and q are connected by l. This is expressed by the following axiom.

Axiom 4.3.1 (Link) $l \in Link(p) \wedge l \in Link(q) \wedge p \not\equiv q \rightarrow link(l, p, q)$

We also assume that a link connects at most two processors.

Axiom 4.3.2 (Point-to-Point) $link(l, p, q) \wedge link(l, p, r) \rightarrow q \equiv r$

Let $FP = \{p \mid \neg correct(p)\}$ and $FL = \{l \mid \neg correct(l)\}$. Define $F = FP \cup FL$. Thus F denotes the set of processors and links which are not always correct, i.e., they experience omission failures during an execution of the protocol. We assume that during any protocol execution there can be at most m processors that suffer omission failures, where $m \in I\!N$.

One important assumption about the network is that during any execution of the protocol all correct processors remain connected via correct links. Otherwise bounded communication delays between correct processors cannot be guaranteed and thus the protocol cannot provide any upper bound for the broadcast termination time. Recall that G is the set of all processors and links, i.e., $G = P \cup L$. Then $G \setminus F = \{p \mid correct(p)\} \cup \{l \mid correct(l)\}$ and it denotes the set of correct processors and links. $G \setminus F$ can be considered as a graph in which processors are vertices and links are edges. Thus we have the following standard definitions (see, e.g. [Gou88]) with $p, q \in G \setminus F$:

Definition 4.3.2

- A $p-q$ *walk* in $G \setminus F$ is a finite alternating sequence of correct processors and links that begins with p and ends with q and in which each link connects the processor that precedes it in the sequence and the processor that follows it in the sequence.

- A $p - q$ *path* in $G \setminus F$ is a $p - q$ walk in which no processor is repeated.

- The *length* of a path is the number of links in that path.

- The *distance* between p and q, denoted by $d(p, q)$, is the minimum of all lengths of $p - q$ paths in $G \setminus F$. If there is no path between p and q, then $d(p, q)$ is ∞.

- $G \setminus F$ is *connected* if and only if there exists a path in $G \setminus F$ between any two processors in $G \setminus F$.
- When $G \setminus F$ is connected, the *diameter* of $G \setminus F$ is the longest distance between any two processors in $G \setminus F$, i.e., $max(\{d(p,q) \mid p,q \in G \setminus F\})$.

Now we can give the axiom for connectivity.

Axiom 4.3.3 (Connectivity) $G \setminus F$ is connected.

Given axiom 4.3.3, we assume that the diameter of $G \setminus F$ is d.

4.3.2 Bounded Communication

We define two primitives:

- *send*(p, m, l) **at** t: processor p starts to send message m along link l at real time t.
- *receive*(p, m, l) **at** t: processor p finishes with receiving message m along link l at real time t.

The abbreviations defined in section 4.2 also hold for these two primitives.

Two processors connected by a link are called neighbors. When *send*(p, m, l) **at** t or *receive*(p, m, l) **at** t holds, l must be a link connecting p and one of its neighbors. This is expressed in terms of clock values by the following axiom.

Axiom 4.3.4 (Neighbor)

$$send(p, m, l)\ \mathbf{at_q}\ T\ \vee\ receive(p, m, l)\ \mathbf{at_q}\ T \rightarrow l \in Link(p)$$

Two processors can send messages to each other if they are connected by a link. The communication between two processors is synchronous in the sense that the duration of the transmission of a message is bounded by two positive constants γ and δ with $\gamma, \delta \in CVAL$ and $\gamma \leq \delta$. Let p and q be two correct processors connected by a correct link l. Let r be any correct processor to be used as reference. If p sends message m along link l at clock value U according to the clock of r, then q will receive m along l at some clock value in the interval $[U + \gamma, U + \delta]$ according to the clock of r.

Axiom 4.3.5 (Bounded Communication)

$$correct(p) \wedge correct(q) \wedge link(l, p, q) \wedge correct(l) \wedge correct(r) \wedge send(p, m, l)\ \mathbf{at_r}\ U \rightarrow$$
$$receive(q, m, l)\ \mathbf{in_r}\ [U + \gamma, U + \delta]$$

This axiom implicitly implies that the local clock function C_p for correct processor p should be monotonic.

Given bounded communication, the clocks of correct processors can be assumed approximately synchronized.

4.3.3 Clock Synchronization

We assume that when processors are correct their clocks are approximately synchronized within a sufficiently small, positive, constant ϵ.

Axiom 4.3.6 (Clock Synchronization)

$$correct(p) \textbf{ at } t \wedge correct(q) \textbf{ at } t \rightarrow |C_p(t) - C_q(t)| < \epsilon$$

It is trival to derive the following lemma.

Lemma 4.3.1 (Clock Synchronization)

$$correct(p) \wedge correct(q) \rightarrow |C_p(t) - C_q(t)| < \epsilon$$

Given axiom 4.3.6 and lemma 4.3.1, we can easily prove the following lemmas.

Lemma 4.3.2 For any primitive φ **at** t,

$$correct(p) \;\wedge\; correct(q) \;\wedge\; \varphi \textbf{ in}_\textbf{p} \, [U, V] \;\rightarrow\; \varphi \textbf{ in}_\textbf{q} \, (U - \epsilon, V + \epsilon).$$

Proof: Assume that the premise of this lemma holds. From φ $\textbf{in}_\textbf{p}$ $[U, V]$, by definition, there exists a T such that φ $\textbf{at}_\textbf{p}$ $T \wedge T \in [U, V]$. Let t be such that $C_p(t) = T$. Then we have φ **at** $t \wedge C_p(t) \in [U, V]$. In terms of the clock of q, we obtain φ $\textbf{at}_\textbf{q}$ $C_q(t)$. Since $correct(p)$ and $correct(q)$ hold, by the synchronization lemma 4.3.1, we have $|C_q(t) - C_p(t)| < \epsilon$, i.e., $C_p(t) - \epsilon < C_q(t) < C_p(t) + \epsilon$. Thus we obtain $U - \epsilon < C_q(t) < V + \epsilon$, i.e., $C_q(t) \in (U - \epsilon, V + \epsilon)$. Therefore we obtain φ $\textbf{in}_\textbf{q}$ $(U - \epsilon, V + \epsilon)$. Hence this lemma holds. □

Lemma 4.3.3 For any primitive φ **at** t,

$$correct(r) \;\wedge\; correct(p) \textbf{ at}_\textbf{p} \, T \;\wedge\; \varphi \textbf{ at}_\textbf{p} \, T \;\rightarrow\; \varphi \textbf{ in}_\textbf{r} \, (T - \epsilon, T + \epsilon).$$

Proof: Assume that the premise of this lemma holds. Let t be such that $C_p(t) = T$. Then by assumption, we have φ **at** t. In terms of the clock of r, we have φ $\textbf{at}_\textbf{r}$ $C_r(t)$. From $correct(p)$ $\textbf{at}_\textbf{p}$ T, we obtain $correct(p)$ **at** t. Since $correct(r)$ holds, by the synchronization axiom 4.3.6, we have $|C_r(t) - C_p(t)| < \epsilon$, i.e., $C_p(t) - \epsilon < C_r(t) < C_p(t) + \epsilon$. Then we obtain $C_r(t) \in (T - \epsilon, T + \epsilon)$. Therefore we have φ $\textbf{in}_\textbf{r}$ $(T - \epsilon, T + \epsilon)$. Hence this lemma holds. □

Lemma 4.3.4 For any primitive φ **at** t,

$$correct(r) \;\wedge\; correct(p) \textbf{ at}_\textbf{r} \, T \;\wedge\; \varphi \textbf{ at}_\textbf{r} \, T \;\rightarrow\; \varphi \textbf{ in}_\textbf{p} \, (T - \epsilon, T + \epsilon).$$

This lemma can be proved similarly as lemma 4.3.3.

The bounded communication property is also expressed in terms of local clock values in the next lemma, which can be proved by using axiom 4.3.5 and lemma 4.3.2.

Lemma 4.3.5 (Bounded Communication)

$$correct(p) \wedge correct(q) \wedge link(l, p, q) \wedge correct(l) \wedge send(p, m, l) \textbf{ at}_\textbf{p} \, U \rightarrow$$
$$receive(q, m, l) \textbf{ in}_\textbf{q} \, (U + \gamma - \epsilon, U + \delta + \epsilon)$$

4.3.4 Failure Assumptions

The atomic broadcast protocol verified in this paper tolerates only omission failures. When a processor suffers an omission failure, it cannot send out messages. More precisely, if a processor p is not correct at real time t, then p is not able to send any message m along any link l at time t. This is also called the *fail silence* property of processors. We express this property in terms of clock values by the following axiom.

Axiom 4.3.7 (Fail Silence) $\neg correct(p)\ \mathbf{at_q}\ T \rightarrow \neg send(p,m,l)\ \mathbf{at_q}\ T$

When a link suffers an omission failure, the messages entrusted on that link may be lost. But if a message has been received by a processor along a (faulty) link, then that message should have been correctly transmitted by that (faulty) link, i.e., that message is not corrupted, there are no timing errors on the message sending and receiving, etc.. Therefore, if a processor q receives a message m along link l at clock value V and q is correct at V according to the clock of any correct processor r, then there exists another processor p which has sent that message earlier along l at some time between $[V-\delta, V-\gamma]$ according to the clock of r.

Axiom 4.3.8 (Only Omission Failure)

$$correct(r) \wedge correct(q)\ \mathbf{at_r}\ V \wedge receive(q,m,l)\ \mathbf{at_r}\ V \rightarrow$$
$$\exists p \not\equiv q : send(p,m,l)\ \mathbf{in_r}\ [V-\delta, V-\gamma]$$

We can also express this property in terms of local clock values on p and q.

Lemma 4.3.6 (Only Omission Failure)

$$correct(q)\ \mathbf{at_q}\ V \wedge receive(q,m,l)\ \mathbf{at_q}\ V \rightarrow$$
$$\exists p \not\equiv q : [send(p,m,l)\ \mathbf{in_p}\ (V-\delta-2\epsilon, V-\gamma+2\epsilon) \wedge$$
$$(correct(q) \rightarrow send(p,m,l)\ \mathbf{in_p}\ (V-\delta-\epsilon, V-\gamma+\epsilon))]$$

Proof: Assume that the premise of the lemma holds. Consider any correct processor r. From $receive(q,m,l)\ \mathbf{at_q}\ V$, since $correct(q)\ \mathbf{at_q}\ V$ holds, by lemma 4.3.3, we obtain $receive(q,m,l)\ \mathbf{in_r}\ (V-\epsilon, V+\epsilon)$. By definition, there exists a $V_1 \in (V-\epsilon, V+\epsilon)$ such that $receive(q,m,l)\ \mathbf{at_r}\ V_1$ holds. Then by the only omission failure axiom 4.3.8, we have $\exists p \not\equiv q : send(p,m,l)\ \mathbf{in_r}\ [V_1-\delta, V_1-\gamma]$. There must also exist a $V_2 \in [V_1-\delta, V_1-\gamma]$ such that $\exists p \not\equiv q : send(p,m,l)\ \mathbf{at_r}\ V_2$. Then by the fail silence axiom 4.3.7, we have $correct(p)\ \mathbf{at_r}\ V_2$. Thus by lemma 4.3.4, we obtain $\exists p \not\equiv q : send(p,m,l)\ \mathbf{in_p}\ (V_2 - \epsilon, V_2+\epsilon)$, i.e., $\exists p \not\equiv q : send(p,m,l)\ \mathbf{in_p}\ (V-\delta-2\epsilon, V-\gamma+2\epsilon)$.

If $correct(q)$ holds, by the only omission failure axiom 4.3.8, we have $\exists p \not\equiv q : send(p,m,l)\ \mathbf{in_q}\ [V-\delta, V-\gamma]$. Then there exists a $V_3 \in [V-\delta, V-\gamma]$ such that $\exists p \not\equiv q : send(p,m,l)\ \mathbf{at_q}\ V_3$. By the fail silence axiom 4.3.7, we obtain $correct(p)\ \mathbf{at_q}\ V_3$.

Then by lemma 4.3.4, we have $\exists p \not\equiv q : send(p, m, l)$ $\mathbf{in_p}$ $(V_3 - \epsilon, V_3 + \epsilon)$, i.e.,
$\exists p \not\equiv q : send(p, m, l)$ $\mathbf{in_p}$ $(V - \delta - \epsilon, V - \gamma + \epsilon)$.
Hence this lemma holds. □

So far, we have given the required assumptions for the system.

4.4 Server Process Specification

For any processor p, we characterize the atomic broadcast server process running on p, i.e., $S(p)$, by the following requirements.

- Initiation requirement.

  When p initiates an update σ at clock time T, it will send message $< T, p, \sigma >$ to all its neighbors immediately. When p has waited long enough to be sure that all correct processors have received that message, p will convey $< T, p, \sigma >$ to client processes.

  Notice that, in the top-level specification, only delivery of updates is important and thus primitive $deliver(p, \sigma)$ **at** t is used. In the server process specification, information about the initiation time T and the initiator s of an update σ is needed to implement the top-level specification. Therefore we define another primitive $convey(p, < T, s, \sigma >)$ **at** t as follows:

  - $convey(p, < T, s, \sigma >)$ **at** t: processor p starts to send message $< T, s, \sigma >$ to client processes located on p at real time t.

  Then the relation between $deliver(p, \sigma)$ **at** t and $convey(p, < T, s, \sigma >)$ **at** t is clear:

  - $deliver(p, \sigma)$ **at** $t \equiv \exists s, T : convey(p, < T, s, \sigma >)$ **at** t

  Assume that any correct processor can send a message to all its neighbors within $T_s \in CVAL$ time units and any correct processor can convey all the updates initiated at the same clock time to client processes within $T_c \in CVAL$ time units. Let $T_r \in CVAL$, $T_r \geq T_s$, be the minimum time to ensure that all correct processors have received a message containing an update after it is initiated. These parameters will be used to determine the values of D_1 and D_2 occurring in the top-level specification.

  We formalize the first property for p by $Start(p)$ as follows:

  $Start(p) \equiv initiate(p, \sigma)$ $\mathbf{at_p}$ $T \rightarrow$

$$\forall l \in Link(p) : send(p, < T, p, \sigma >, l) \text{ in}_\text{p} [T, T + T_s] \wedge$$
$$convey(p, < T, p, \sigma >) \text{ in}_\text{p} [T + T_r, T + T_r + T_c]$$

- Relay requirement.

  When p receives a message $< T, s, \sigma >$, it will relay this message to all its neighbors except the one which just sent this message to itself. But it will do so only if it receives the message at some local time in the interval $[T, T + T_r)$, since T is the initiation time of σ and T_r is the maximum time needed for every correct processor to receive this message. Later, similarly as in the initiator's case, when its clock reaches $T + T_r$, p will convey $< T, s, \sigma >$ to client processes. This property is formalized by the following formula $Relay(p)$:

  $$Relay(p) \equiv receive(p, < T, s, \sigma >, l) \textbf{ at}_\textbf{p} U \wedge U \in [T, T + T_r) \rightarrow$$
  $$\forall l_1 \in Link(p) \setminus \{l\} : send(p, < T, s, \sigma >, l_1) \text{ in}_\text{p} [U, U + T_s] \wedge$$
  $$convey(p, < T, s, \sigma >) \text{ in}_\text{p} [T + T_r, T + T_r + T_c]$$

- Convey requirement.

  If processor p conveys a message $< T, s, \sigma >$ at time U on its clock, then there can be only two possibilities: either p initiated σ itself at local clock value T with $U \in [T + T_r, T + T_r + T_c]$, or p received the message $< T, s, \sigma >$ at some clock value in the interval $[T, T + T_r)$ and $p \not\equiv s \wedge U \in [T + T_r, T + T_r + T_c]$ holds.

  When p initiates σ at local time T or it receives $< T, s, \sigma >$ at some local time in the interval $[T, T + T_r)$, we say that p *learns of* message $< T, s, \sigma >$ and define an abbreviation for it as follows:

  $$Learn(p, < T, s, \sigma >) \equiv (initiate(p, \sigma) \textbf{ at}_\textbf{p} T \wedge p \equiv s) \vee$$
  $$(\exists l : receive(p, < T, s, \sigma >, l) \text{ in}_\text{p} [T, T + T_r) \wedge p \not\equiv s)$$

  Then the requirement is formalized by the following formula $Origin(p)$:

  $$Origin(p) \equiv convey(p, < T, s, \sigma >) \textbf{ at}_\textbf{p} U \rightarrow$$
  $$Learn(p, < T, s, \sigma >) \wedge U \in [T + T_r, T + T_r + T_c]$$

- Ordering requirement.

  If two messages are conveyed by processor p, then they will be conveyed in the order of initiation times of updates contained in these two messages. If initiation times are the same, then they will be conveyed according to the priority of initiators. Therefore it is assumed that there is a total order $\prec$ on the set of processor names P. This total order specifies a priority ordering among processors.

  We define a lexicographical ordering $\sqsubset$ on pairs $< T, s >$.

Definition 4.4.1 For any two pairs (T_1, s_1) and (T_2, s_2),
$(T_1, s_1) \sqsubset (T_2, s_2)$ iff $(T_1 < T_2) \vee (T_1 = T_2 \wedge s_1 \prec s_2)$.

Then the fourth requirement is formalized by the following formula $Sequen(p)$:

$$Sequen(p) \equiv convey(p, < T_1, s_1, \sigma_1 >) \ \mathbf{at_p} \ V_1 \wedge convey(p, < T_2, s_2, \sigma_2 >) \ \mathbf{at_p} \ V_2 \\ \rightarrow (V_1 < V_2 \leftrightarrow (T_1, s_1) \sqsubset (T_2, s_2))$$

The requirements mentioned above are only for correct processors, i.e., they define the standard behaviour of correct processors. Since we assume that processors can only suffer omission failures, we still need to define what is the acceptable behaviour for faulty processors. Thus we have the following requirement for any arbitrary processor p.

- Failure requirement.

  When p sends a message $< T, s, \sigma >$ to one neighbor at local time U, there can be only two possibilities: either p initiated σ itself at local time T and $U \in [T, T + T_s]$ holds, or p received $< T, s, \sigma >$ at some local time V and $correct(p)$ $\mathbf{at_p}$ $V \wedge U \in [V, V + T_s] \wedge V \in [T, T + T_r)$ holds. This requirement is expressed by the following formula $Source(p)$:

  $$Source(p) \equiv send(p, < T, s, \sigma >, l) \ \mathbf{at_p} \ U \rightarrow \\ (initiate(p, \sigma) \ \mathbf{at_p} \ T \wedge U \in [T, T + T_s] \wedge p \equiv s) \vee \\ \exists l_1, V : (receive(p, < T, s, \sigma >, l_1) \ \mathbf{at_p} \ V \wedge correct(p) \ \mathbf{at_p} \ V \wedge \\ p \not\equiv s \wedge U \in [V, V + T_s] \wedge V \in [T, T + T_r))$$

  When $send(p, < T, s, \sigma >, l)$ $\mathbf{at_p}$ U holds, by the fail silence axiom 4.3.7, it implies that $correct(p)$ $\mathbf{at_p}$ U holds. But $correct(p)$ $\mathbf{at_p}$ U does not imply $correct(p)$. It is quite possible that p is faulty at some other time. That is why this requirement should be for any processor p and not only for correct one.

Recall that $Spec(p)$ denotes the specification for server process $S(p)$. Thus,

$$Spec(p) \equiv [correct(p) \rightarrow Start(p) \wedge Relay(p) \wedge Origin(p) \wedge Sequen(p)] \wedge Source(p)$$

We assume that server process $S(p)$ satisfies specification $Spec(p)$.

Axiom 4.4.1 (Server Process Specification) $S(p)$ **sat** $Spec(p)$

Thus the behavior of any processor p is specified by this axiom and the fail silence axiom 4.3.7.

4.5 Verification of Termination

In this section, we prove that the termination property of the atomic broadcast protocol follows from the axioms and lemmas given in the previous sections. To make the proof easier, we first give some additional lemmas.

The first lemma expresses that if a correct processor p receives a message $< T, s, \sigma >$ at local time U in the interval $[T, T + T_r)$, then its correct neighbor q which is not s will receive $< T, s, \sigma >$ at local time V in the interval $[T, U + T_s + \delta + \epsilon)$, provided $\gamma \geq \epsilon$.

Lemma 4.5.1 (Propagation) If $\gamma \geq \epsilon$, then
$correct(p) \wedge correct(q) \wedge link(l_2, p, q) \wedge correct(l_2) \wedge receive(p, < T, s, \sigma >, l_1)$ $\mathbf{at_p}$ $U \wedge$
$U \in [T, T + T_r) \wedge q \not\equiv s \rightarrow \exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{in_q}$ $[T, U + T_s + \delta + \epsilon)$.

Proof: Assume that the premise of the lemma holds. Since $receive(p, < T, s, \sigma >, l_1)$ $\mathbf{at_p}$ U holds, there are two possibilities.

- If $l_1 \not\equiv l_2$, then q is not the processor which just sent the message $< T, s, \sigma >$ to p. By $Relay(p)$, p will send the message $< T, s, \sigma >$ to all its neighbors except the one that just sent this message to itself within T_s time units. Hence p will send $< T, s, \sigma >$ to q along link l_2 and thus we have
  $send(p, < T, s, \sigma >, l_2)$ $\mathbf{in_p}$ $[U, U + T_s]$.
  By definition, there exists an U_1 such that
  $send(p, < T, s, \sigma >, l_2)$ $\mathbf{at_p}$ $U_1 \wedge U_1 \in [U, U + T_s]$.
  By the bounded communication lemma 4.3.5, we obtain
  $receive(q, < T, p, \sigma >, l_2)$ $\mathbf{in_q}$ $(U_1 + \gamma - \epsilon, U_1 + \delta + \epsilon)$.
  Since $U_1 \geq U$ and $U \geq T$, we have $U_1 \geq T$. It is assumed that $\gamma \geq \epsilon$. Thus we obtain $U_1 + \gamma - \epsilon \geq T$. Together with $U_1 \leq U + T_s$, we obtain
  $\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{in_q}$ $[T, U + T_s + \delta + \epsilon)$.

- If $l_1 \equiv l_2$, then p receives $< T, p, \sigma >$ from link l_2 and thus we have
  $receive(p, < T, s, \sigma >, l_2)$ $\mathbf{at_p}$ U.
  Since $correct(p)$ holds, by the only omission failure lemma 4.3.6, there exists a p_1 such that
  $p_1 \not\equiv p \wedge send(p_1, < T, s, \sigma >, l_2)$ $\mathbf{in_{p_1}}$ $(U - \delta - \epsilon, U - \gamma + \epsilon)$
  holds. By the neighbor axiom 4.3.4, we have $l_2 \in Link(p) \wedge l_2 \in Link(p_1)$. Since $p \not\equiv p_1$, by the link axiom 4.3.1, we obtain $link(l_2, p, p_1)$. But it is assumed that $link(l_2, p, q)$. Thus by the point-to-point axiom 4.3.2, we obtain $p_1 \equiv q$. Thus there exists a U_2 such that
  $send(q, < T, s, \sigma >, l_2)$ $\mathbf{at_q}$ $U_2 \wedge U_2 \in (U - \delta - \epsilon, U - \gamma + \epsilon)$
  holds. Since $q \not\equiv s$, by $Source(q)$, we obtain
  $\exists l, V : (receive(q, < T, s, \sigma >, l)$ $\mathbf{at_q}$ $V \wedge correct(q)$ $\mathbf{at_q}$ $V \wedge$
  $q \not\equiv s \wedge U_2 \in [V, V + T_s] \wedge V \in [T, T + T_r))$.
  From $V \leq U_2$ and $U_2 < U - \gamma + \epsilon$, we obtain $V < U - \gamma + \epsilon$ and thus $V < U + T_s + \delta + \epsilon$. Together with $V \geq T$, we have
  $\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{in_q}$ $[T, U + T_s + \delta + \epsilon)$.

Hence this lemma holds. □

The intuition behind this lemma is as follows. When a correct processor p receives a message $< T, s, \sigma >$ at clock time U and it does not receive $< T, s, \sigma >$ from its correct neighbor q, p will relay $< T, s, \sigma >$ to q within T_s time units. That is, the latest clock time at which p starts to send $< T, s, \sigma >$ to q is $U + T_s$. Since p and q are correct processors, the latest corresponding clock time to $U + T_s$ on q is $U + T_s + \epsilon$. Sending $< T, s, \sigma >$ from p to q takes at most δ time units. Thus, the latest clock time at which q receives $< T, s, \sigma >$ is $U + T_s + \delta + \epsilon$. Figure 4.2 shows the timing relation between the local clocks of processors.

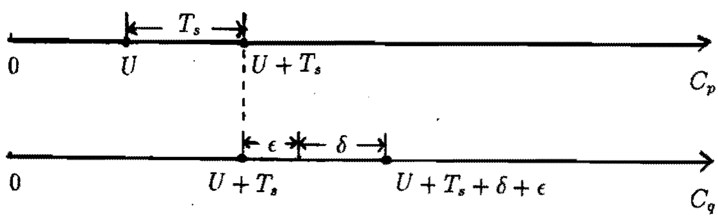

Fig. 4.2. Timing Relation Picture for Lemma 4.5.1

Recall that d is the diameter of the graph consisting of all correct processors and links. The following lemma shows that if $T_r \geq (d-1)(T_s + \delta + \epsilon)$ and $\gamma \geq \epsilon$ and correct processor s initiates an update σ at local time T, then any other correct processor q will receive $< T, s, \sigma >$ in the interval $[T, T + d(s,q)(T_s + \delta + \epsilon))$.

Lemma 4.5.2 (Bounded Receiving) If $T_r \geq (d-1)(T_s + \delta + \epsilon)$ and $\gamma \geq \epsilon$, then

$$correct(s) \wedge correct(q) \wedge initiate(s,\sigma) \ \mathbf{at_s} \ T \wedge q \not\equiv s \rightarrow$$
$$\exists l : receive(q, < T, s, \sigma >, l) \ \mathbf{in_q} \ [T, T + d(s,q)(T_s + \delta + \epsilon)).$$

Proof: Assume that the premise of the lemma holds. We prove this lemma by induction on the distance between s and q. Since $s \not\equiv q$, we start with $d(s,q) = 1$.

- $d(s,q) = 1$. Since both s and q are correct processors, by the definition of $d(s,q)$, they are connected by some correct link. Let l be that link. Then we obtain $link(l,s,q) \wedge correct(l)$. By the server process specification axiom 4.4.1 and $correct(s)$, we have $Start(s)$. From $Start(s)$ and $initiate(s,\sigma)$ $\mathbf{at_s}$ T, s will send the message $< T, s, \sigma >$ to all its neighbors within T_s time units. Thus it will also send $< T, s, \sigma >$ to processor q along link l. Thus we have $send(s, < T, s, \sigma >, l)$ $\mathbf{in_s}$ $[T, T + T_s]$.
  By definition, there exists a U such that

$send(s, < T, s, \sigma >, l)$ $\mathbf{at_s}$ $U \wedge U \in [T, T + T_s]$.
By the bounded communication lemma 4.3.5, we obtain
$receive(q, < T, s, \sigma >, l)$ $\mathbf{in_q}$ $(U + \gamma - \epsilon, U + \delta + \epsilon)$.
Since it is assumed that $\gamma \geq \epsilon$, together with $U \geq T$, we obtain $U + \gamma - \epsilon \geq T$.
By $U < T + T_s$, we obtain
$receive(q, < T, s, \sigma >, l)$ $\mathbf{in_q}$ $[T, T + T_s + \delta + \epsilon)$, i.e.,
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{in_q}$ $[T, T + d(s, q)(T_s + \delta + \epsilon))$.

- $d(s, q) = k + 1$ with $k \geq 1$. By definition, there must exist a link l_2 and a processor q_1 such that $link(l_2, q_1, q) \wedge correct(l_2) \wedge correct(q_1) \wedge d(s, q_1) = k \wedge d(q_1, q) = 1$ holds. By the induction hypothesis, we have
$\exists l_1 : receive(q_1, < T, s, \sigma >, l_1)$ $\mathbf{in_{q_1}}$ $[T, T + k(T_s + \delta + \epsilon))$.
By definition, there exists a V_1 such that
$\exists l_1 : (receive(q_1, < T, s, \sigma >, l_1)$ $\mathbf{at_{q_1}}$ $V_1 \wedge V_1 \in [T, T + k(T_s + \delta + \epsilon))$).
Since $T_r \geq (d - 1)(T_s + \delta + \epsilon)$ and $d \geq k + 1$, where d is the diameter of $G \setminus F$, we obtain
$k(T_s + \delta + \epsilon) \leq T_r$ and thus we have
$\exists l_1 : (receive(q_1, < T, s, \sigma >, l_1)$ $\mathbf{at_{q_1}}$ $V_1 \wedge V_1 \in [T, T + T_r)$).
Since $\gamma \geq \epsilon$, by the propagation lemma 4.5.1, we have
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{in_q}$ $[T, V_1 + T_s + \delta + \epsilon)$, i.e.,
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{in_q}$ $[T, T + (k + 1)(T_s + \delta + \epsilon))$.
Hence we have proved
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{in_q}$ $[T, T + d(s, q)(T_s + \delta + \epsilon))$.

Hence this lemma holds. □

This lemma can be informally explained as follows. When a correct processor s initiates an update σ at clock time T, it will send message $< T, s, \sigma >$ to all its neighbors within T_s time units, i.e., the latest clock time at which s starts to send $< T, s, \sigma >$ to all its neighbors is $T + T_s$. Suppose q_1 is a correct neighbor of s. Then the latest corresponding clock time to $T + T_s$ on q_1 is $T + T_s + \epsilon$. Sending $< T, s, \sigma >$ from s to q_1 takes at most δ time units. Thus the latest clock time at which q_1 receives $< T, s, \sigma >$ is $T + T_s + \delta + \epsilon$. Then q_1 will relay $< T, s, \sigma >$ to all its neighbors except s within T_s time units, i.e., the latest clock time at which q_1 starts to send $< T, s, \sigma >$ to its neighbors is $T + 2T_s + \delta + \epsilon$. Suppose q_2 is a correct neighbor of q_1 but $q_2 \not\equiv s$. Then the latest corresponding clock time to $T + 2T_s + \delta + \epsilon$ on q_2 is $T + 2T_s + \delta + 2\epsilon$. Similarly, sending $< T, s, \sigma >$ from q_1 to q_2 takes at most δ time units. Thus the latest clock time at which q_2 receives $< T, s, \sigma >$ is $T + 2T_s + 2\delta + 2\epsilon$. This procedure can go on until every correct processor has received

$< T, s, \sigma >$. Figure 4.3 shows the timing relation between the local clocks of processors.

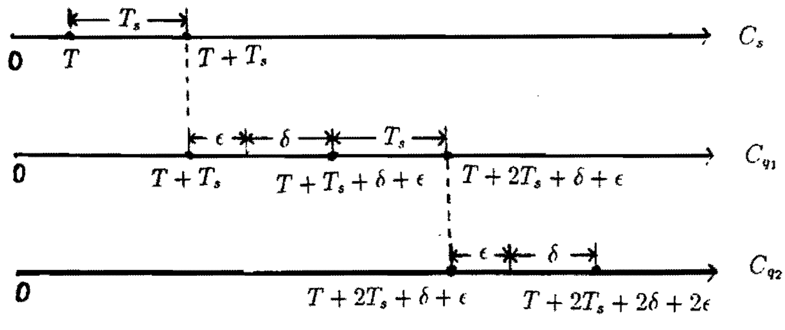

Fig. 4.3. Timing Relation Picture for Lemma 4.5.2

The next lemma shows that if a correct processor s initiates σ at local clock time T, then every correct processor q will convey $< T, s, \sigma >$ in the interval $[T+T_r, T+T_r+T_c]$ according to their own clocks, provided $T_r \geq d(T_s + \delta + \epsilon)$ and $\gamma \geq \epsilon$.

Lemma 4.5.3 (Convey) If $T_r \geq d(T_s + \delta + \epsilon)$ and $\gamma \geq \epsilon$, then $correct(s) \wedge correct(q) \wedge initiate(s, \sigma)$ $\mathbf{at_s}$ $T \rightarrow convey(q, < T, s, \sigma >)$ $\mathbf{in_q}$ $[T + T_r, T + T_r + T_c]$.

Proof: Assume that the premise of the lemma holds. We prove this lemma in two cases.

- $d(s, q) = 0$. By definition, we have $s \equiv q$. By the server process specification axiom 4.4.1 and $correct(q)$, we have $Start(q)$. From $Start(q)$ and $initiate(s, \sigma)$ $\mathbf{at_s}$ $T \wedge s \equiv q$, we obtain
  $convey(q, < T, s, \sigma >)$ $\mathbf{in_q}$ $[T + T_r, T + T_r + T_c]$.
- $d(s, q) > 0$. By definition, we have $s \not\equiv q$. Since $T_r \geq d(T_s + \delta + \epsilon)$ and $\gamma \geq \epsilon$, by the bounded receiving lemma 4.5.2, we obtain
  $\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{in_q}$ $[T, T + d(s, q)(T_s + \delta + \epsilon))$, i.e.,
  $\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{in_q}$ $[T, T + T_r)$.
  By $Relay(q)$, we obtain $convey(q, < T, s, \sigma >)$ $\mathbf{in_q}$ $[T + T_r, T + T_r + T_c]$.

Hence this lemma holds. □

Next we prove that the termination property follows from the axioms and lemmas given before.

Theorem 4.5.1 (Termination) If $T_r \geq d(T_s + \delta + \epsilon)$, $\gamma \geq \epsilon$, and $D_1 \geq T_r + T_c$, then

$$correct(s) \wedge correct(q) \wedge initiate(s, \sigma) \ \mathbf{at_s} \ T \rightarrow deliver(q, \sigma) \ \mathbf{by_q} \ T + D_1,$$

i.e., the termination property TERM holds.

Proof: Assume that the premise of this theorem holds. Since $T_r \geq d(T_s+\delta+\epsilon)$ and $\gamma \geq \epsilon$, by the convey lemma 4.5.3, we obtain $convey(q, < T, s, \sigma >)$ $\mathbf{in_q}$ $[T+T_r, T+T_r+T_c]$. By definition, we obtain $deliver(q, \sigma)$ $\mathbf{in_q}$ $[T+T_r, T+T_r+T_c]$.
Since $D_1 \geq T + r + T_c$, we have $deliver(q, \sigma)$ $\mathbf{by_q}$ $T + D_1$.
Hence this theorem holds. □

4.6 Verification of Atomicity

In this section, we prove the atomicity property of the atomic broadcast protocol. We first show some lemmas which will help prove the atomicity property.

The next lemma states that if correct processor p receives message $< T, s, \sigma >$ at some local time in the interval $[T, T+T_r)$, then that update σ was initiated by processor s at local time T, provided $\gamma > 2\epsilon$.

Lemma 4.6.1 (Initiation) If $\gamma > 2\epsilon$, then

$$correct(p) \wedge receive(p, < T, s, \sigma >, l)\ \mathbf{in_p}\ [T, T+T_r) \rightarrow initiate(s, \sigma)\ \mathbf{at_s}\ T.$$

Proof: Assume that the premise of the lemma holds. By definition, there exists a V such that

$$correct(p) \wedge receive(p, < T, s, \sigma >, l)\ \mathbf{at_p}\ V \wedge V \in [T, T+T_r) \tag{1}$$

holds. By the only omission failure lemma 4.3.6, there exists a s_1 and a U_1 such that

$$s_1 \not\equiv p \wedge send(s_1, < T, s, \sigma >, l)\ \mathbf{at_{s_1}}\ U_1 \wedge U_1 \in (V - \delta - 2\epsilon, V - \gamma + 2\epsilon). \tag{2}$$

By $Source(s_1)$, there exist l_1 and V_1 such that

$$(initiate(s_1, \sigma)\ \mathbf{at_{s_1}}\ T \wedge s_1 \equiv s) \vee \tag{3}$$

$$receive(s_1, < T, s, \sigma >, l_1)\ \mathbf{at_{s_1}}\ V_1 \wedge correct(s_1)\ \mathbf{at_{s_1}}\ V_1 \wedge$$
$$s_1 \not\equiv s \wedge U_1 \in [V_1, V_1 + T_s] \wedge V_1 \in [T, T+T_r) \tag{4}$$

holds.

If (3) holds, we have proved $initiate(s, \sigma)$ $\mathbf{at_s}$ T.

If (3) does not hold, then s_1 is not the initiator of σ and (4) holds.
By (1) and (4), we obtain $V \in [T, T+T_r)$ and $V_1 \in [T, T+T_r)$.
From (2), we have $U_1 < V - \gamma + 2\epsilon$, i.e., $V > U_1 + \gamma - 2\epsilon$. From (4), we have $U_1 \geq V_1$. Thus we obtain $V > V_1 + \gamma - 2\epsilon$, i.e., $V - V_1 > \gamma - 2\epsilon$.

From $receive(s_1, < T, s, \sigma >, l_1)$ $\mathbf{at_{s_1}}$ V_1 and $correct(s_1)$ $\mathbf{at_{s_1}}$ V_1 in (4), we obtain by the only omission failure lemma 4.3.6 another processor $s_2 \not\equiv s_1$. If s_2 is not the initiator of σ, we follow the above steps and then obtain another processor $s_3 \not\equiv s_2$. This procedure can continue until we obtain a processor s_{k-1} such that $s_1, \ldots, s_{k-1}$ are not the initiator of σ, where $k \in I\!N^+ \wedge k \geq 2$. Since k is arbitray and $\gamma > 2\epsilon$, let $k \geq (V-T)/(\gamma - 2\epsilon)$. Then, for any $i = 2, 3, \ldots, k-1$, there exist l_i and V_i such that

$$s_i \not\equiv s_{i-1} \ \wedge\ receive(s_i, < T, s, \sigma >, l_i)\ \mathbf{at}_{s_i}\ V_i\ \wedge\ correct(s_i)\ \mathbf{at}_{s_i}\ V_i\ \wedge \\ s_i \not\equiv s\ \wedge\ V_i \in [T, T+T_r)\ \wedge\ V_{i-1} - V_i > \gamma - 2\epsilon)$$

holds. From $V_{i-1} - V_i > \gamma - 2\epsilon$ and $V - V_1 > \gamma - 2\epsilon$, we obtain $V - V_i > i(\gamma - 2\epsilon)$, for any $i = 1, 2, \ldots, k-1$. From $receive(s_{k-1}, < T, s, \sigma >, l_{k-1})\ \mathbf{at}_{s_{k-1}}\ V_{k-1}$, by the only omission failure lemma 4.3.6, there exists a processor $s_k \not\equiv s_{k-1}$ such that $send(s_k, < T, s, \sigma >, l_{k-1})\ \mathbf{in}_{s_k}\ (V_{k-1} - \delta - 2\epsilon, V_{k-1} - \gamma + 2\epsilon)$ holds.
By $Source(s_k)$, there exist l_k and V_k such that

$$(initiate(s_k, \sigma)\ \mathbf{at}_{s_k}\ T \wedge s_k \equiv s) \vee \tag{5}$$

$$receive(s_k, < T, s, \sigma >, l_k)\ \mathbf{at}_{s_k}\ V_k \wedge s_k \not\equiv s \wedge V_k \in [T, T+T_r) \tag{6}$$

holds.
If (6) holds, similar as before, we can derive $V_{k-1} - V_k > \gamma - 2\epsilon$. From $V - V_i > i(\gamma - 2\epsilon)$, we obtain $V - V_k > k(\gamma - 2\epsilon)$. Since $\gamma > 2\epsilon$ and $k \geq (V - T)/(\gamma - 2\epsilon)$, we have $V_k < T$ and thus (6) does not hold. Therefore (5) must hold, i.e., s_k is the initiator of σ. Hence this lemma holds. □

We define an abbreviation $Firstrec(p, < T, s, \sigma >, l)\ \mathbf{at}_p\ U$,

is one of the first correct processors which have received $< T, s, \sigma >$ according to their own clocks, as follows:

$$Firstrec(p, < T, s, \sigma >, l)\ \mathbf{at}_p\ U \equiv correct(p) \wedge receive(p, < T, s, \sigma >, l)\ \mathbf{at}_p\ U\ \wedge \\ \forall p', l', U' : (correct(p') \wedge p' \not\equiv p \wedge receive(p', < T, s, \sigma >, l')\ \mathbf{at}_{p'}\ U' \rightarrow U' \geq U)$$

The next lemma shows that if p receives $< T, s, \sigma >$ at local time U, p is one of the first correct processors which have received $< T, s, \sigma >$, and s is faulty, then any processor q which is not p and has sent $< T, s, \sigma >$ to p earlier than U is a faulty processor.

Lemma 4.6.2 (Faulty Sender)

$$Firstrec(p, < T, s, \sigma >, l_1)\ \mathbf{at}_p\ U \wedge \neg correct(s) \wedge send(q, < T, s, \sigma >, l_2)\ \mathbf{at}_q\ V\ \wedge \\ U > V \wedge q \not\equiv p \rightarrow \neg correct(q)$$

Proof: Assume that the premise of the lemma holds. From $send(q, < T, s, \sigma >, l_2)\ \mathbf{at}_q\ V$, by $Source(q)$, we obtain

$$(initiate(q, \sigma)\ \mathbf{at}_q\ T \wedge q \equiv s) \vee \tag{1}$$

$$\exists l', U' : (receive(q, < T, s, \sigma >, l')\ \mathbf{at}_q\ U' \wedge correct(q)\ \mathbf{at}_q\ U' \wedge V \in [U', U' + T_s]\ ). \tag{2}$$

Then there exist two possibilities:

- if (1) holds, then $q \equiv s$ and thus, by assumption, $\neg correct(q)$ holds;
- if (2) holds, we have $V \geq U'$. Since $U > V$, we obtain $U > U'$.
  If $correct(q)$ holds, by $Firstrec(p, < T, s, \sigma >, l)\ \mathbf{at}_p\ U$, we should have $U' \geq U$ and thus it leads to a contradiction. Thus $\neg correct(q)$ holds.

For both cases, we obtain $\neg correct(q)$. Hence this lemma holds. □

The following lemma shows that if p receives $< T, s, \sigma >$ at local time V, p is one of the first correct processors which have received $< T, s, \sigma >$, and s is faulty, then $V < T + m(T_s + \delta + 2\epsilon)$, where m is the maximum number of faulty processors in the network, provided $\gamma \geq 2\epsilon$.

Lemma 4.6.3 (First Correct Receiving) If $\gamma \geq 2\epsilon$, then

$Firstrec(p, < T, s, \sigma >, l)$ $\mathbf{at_p}$ $V \wedge \neg correct(s) \rightarrow V < T + m(T_s + \delta + 2\epsilon)$.

Proof: Assume that the premise of the lemma holds. From $receive(p, < T, s, \sigma >, l)$ $\mathbf{at_p}$ V and $correct(p)$, by the only omission failure lemma 4.3.6, there exists a s_1 and a U_1 such that

$s_1 \not\equiv p \wedge send(s_1, < T, s, \sigma >, l)$ $\mathbf{at_{s_1}}$ $U_1 \wedge U_1 \in (V - \delta - 2\epsilon, V - \gamma + 2\epsilon)$

holds. Thus we have

$$V < U_1 + \delta + 2\epsilon \text{ and } U_1 < V - \gamma + 2\epsilon. \quad (1)$$

Since $Firstrec(p, < T, s, \sigma >, l)$ $\mathbf{at_p}$ V holds, by the faulty sender lemma 4.6.2, s_1 is a faulty processor, i.e., $\neg correct(s_1)$ holds. By $Source(s_1)$, there exist l_1 and V_1 such that

$$(initiate(s_1, \sigma) \ \mathbf{at_{s_1}} \ T \wedge s_1 \equiv s \wedge U_1 \in [T, T + T_s)\ ) \vee \quad (2)$$

$$(receive(s_1, < T, s, \sigma >, l_1) \ \mathbf{at_{s_1}} \ V_1 \wedge correct(s_1) \ \mathbf{at_{s_1}} \ V_1 \wedge$$
$$s_1 \not\equiv s \wedge U_1 \in [V_1, V_1 + T_s] \wedge V_1 \in [T, T + T_r)\ ). \quad (3)$$

holds. Then there are two possibilities.

- If (2) holds, then s_1 is the initiator of σ and we have $U_1 \leq T + T_s$.
  From (1), we obtain $V < T + T_s + \delta + 2\epsilon$.
  Since $\neg correct(s)$ holds, there is at least one faulty processor, i.e., the maximum number of faulty processors $m \geq 1$.
  Thus we obtain $V < T + m(T_s + \delta + 2\epsilon)$.

- If (3) holds, then together with (1), we obtain
  $$V < V_1 + T_s + \delta + 2\epsilon. \quad (4)$$
  From $receive(s_1, < T, s, \sigma >, l_1)$ $\mathbf{at_{s_1}}$ V_1 and $correct(s_1)$ $\mathbf{at_{s_1}} V_1$, by the only omission failure lemma 4.3.6, there exist s_2 and U_2 such that s_2 has sent $< T, s, \sigma >$ to s_1 along link l_1 at clock time U_2.
  Similar as before, we have $U_2 \in (V_1 - \delta - 2\epsilon, V_1 - \gamma + 2\epsilon)$, i.e., $U_2 < V_1 - \gamma + 2\epsilon$.
  Since it is assumed that $\gamma \geq 2\epsilon$, we obtain $U_2 < V_1$.
  From (1), we have $U_1 < V - \gamma + 2\epsilon$. By $\gamma \geq 2\epsilon$, we have $U_1 < V$.
  From (3), we have $V_1 \leq U_1$ and thus $V_1 < V$. Therefore we obtain $U_2 < V$.
  Then by the faulty sender lemma 4.6.2, $\neg correct(s_2)$ holds.
  By $Source(s_2)$, we obtain a formula similar as (2) and (3).

If s_2 is not the initiator of σ, we follow the above steps and then obtain another s_3 which is also a faulty processor, by the same reason as for s_2. Since there are at most m faulty processors, we cannot continue this procedure infinitely. We must obtain a s_k with $k \leq m$ and it is the initiator of σ.

Thus we have faulty processors $s_1, \ldots, s_{k-1}$ which are not the initiator of σ. For any $i = 2, 3, \ldots, k-1$, by the only omission failure lemma 4.3.6 and $Source(s_i)$, there exist l_i and V_i such that

$$s_i \not\equiv s_{i-1} \wedge receive(s_i, < T, s, \sigma >, l_i) \ \mathbf{at}_{\mathbf{s_i}} \ V_i \wedge correct(s_i) \ \mathbf{at}_{\mathbf{s_i}} \ V_i \wedge s_i \not\equiv s \ \wedge$$
$$V_{i-1} < V_i + T_s + \delta + 2\epsilon$$

holds. Then we obtain

$$V_1 < V_{k-1} + (k-2)(T_s + \delta + 2\epsilon). \quad (5)$$

From $receive(s_{k-1}, < T, s, \sigma >, l_{k-1}) \ \mathbf{at}_{\mathbf{s_{k-1}}} \ V_{k-1}$ and $correct(s_{k-1}) \ \mathbf{at}_{\mathbf{s_{k-1}}} \ V_{k-1}$, by the only omission failure lemma 4.3.6, there exists a U_k such that

$s_k \not\equiv s_{k-1} \wedge send(s_k, < T, s, \sigma >, l_{k-1}) \ \mathbf{at}_{\mathbf{s_k}} \ U_k \wedge U_k \in (V_{k-1} - \delta - 2\epsilon, V_{k-1} - \gamma + 2\epsilon)$ holds. Then we obtain $V_{k-1} < U_k + \delta + 2\epsilon$.

Together with (5), we obtain

$$V_1 < U_k + (k-2)T_s + (k-1)(\delta + 2\epsilon). \quad (6)$$

Since s_k is the initiator of σ, by $Source(s_k)$, we have

$initiate(s_k, \sigma) \ \mathbf{at}_{\mathbf{s_k}} \ T \wedge s_k \equiv s \wedge U_k \in [T, T + T_s]$.

Together with (6), we obtain

$$V_1 < T + (k-1)(T_s + \delta + 2\epsilon). \quad (7)$$

Combining (4) and (7), it results in $V < T + k(T_s + \delta + 2\epsilon)$.

Since $k \leq m$, we finally obtain $V < T + m(T_s + \delta + 2\epsilon)$.

Hence this lemma holds. □

Here we give an intuitive explanation of the lemma 4.6.3 for the case $m = 2$. Assume that s_1 and s_2 are faulty processors and connected by a link l. Suppose that s_2 initiated an update σ at local time T. As we have seen from the proof of the lemma, s_2 behaved in the same way as a correct initiator. Namely, s_2 will send the message $< T, s_2, \sigma >$ to all its neighbors within T_s time units according to its own clock. When s_1 receives $< T, s_2, \sigma >$ from s_2 at some local time V, it is derived (by $Source(s_1)$) that $correct(s_1) \ \mathbf{at}_{\mathbf{s_1}} \ V$ holds. By the only omission failure lemma 4.3.6, sending $< T, s_2, \sigma >$ from s_2 to s_1 takes at most $\delta + 2\epsilon$ time units as measured on the clock of s_1. Thus the latest clock time at which s_1 receives $< T, s_2, \sigma >$ is $T + T_s + \delta + 2\epsilon$. Then s_1 will relay $< T, s_2, \sigma >$ to all its neighbors except s_2 within T_s time units according to its own clock, as a correct processor will do. Suppose p is a correct neighbor of s_1. Since s_1 is faulty and p is correct, by the only omission failure lemma 4.3.6 again, sending $< T, s_2, \sigma >$ from s_1 to

p takes at most $\delta + 2\epsilon$ time units as measured on the clock of p. Thus the latest clock time at which p receives $< T, s_2, \sigma >$ is $T + 2T_s + 2\delta + 4\epsilon$. Then we have the following figure 4.4, which is similar to figure 4.3, but the upper bound is slightly different.

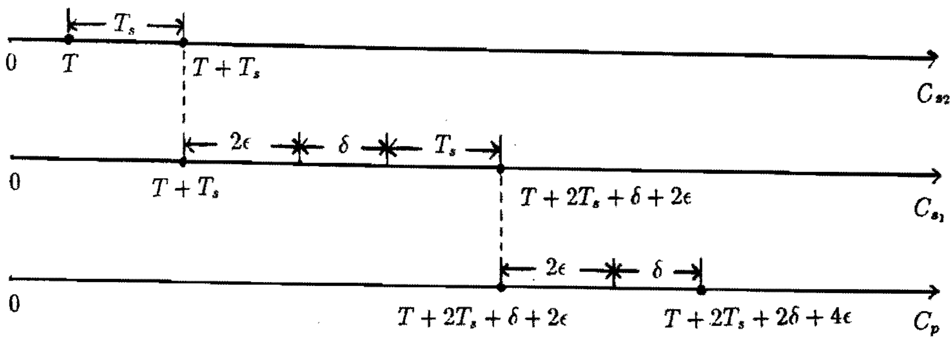

Fig. 4.4. Timing Relation Picture for Lemma 4.6.3

The following lemma shows that if p receives $< T, s, \sigma >$ at local time U in the interval $[T, T + T_r)$, p is one of the first correct processors which have received $< T, s, \sigma >$, and s is faulty, then any other correct processor q will receive $< T, s, \sigma >$ at some local time in the interval $[T, U + d(p,q)(T_s + \delta + \epsilon))$, provided $T_r \geq (d+m-1)(T_s+\delta) + (d+2m-1)\epsilon$ and $\gamma \geq 2\epsilon$.

Lemma 4.6.4 (Correct Receiving) If $T_r \geq (d+m-1)(T_s+\delta) + (d+2m-1)\epsilon$ and $\gamma \geq 2\epsilon$, then

$$Firstrec(p, < T, s, \sigma >, l') \ \mathbf{at_p} \ U \wedge U \in [T, T+T_r) \wedge \neg correct(s) \wedge correct(q) \wedge p \not\equiv q \rightarrow \exists l : receive(q, < T, s, \sigma >, l) \ \mathbf{in_q} \ [T, U + d(p,q)(T_s + \delta + \epsilon)).$$

Proof: Assume that the premise of the lemma holds. We prove this lemma by induction on the distance between p and q. Since $p \not\equiv q$, we start with $d(p,q) = 1$.

- $d(p,q) = 1$. By definition, p and q are connected by some correct link. Let that link be l. Then we have $link(l,p,q) \wedge correct(l)$.
  From $Firstrec(p, < T, s, \sigma >, l')$ $\mathbf{at_p}$ U, by the only omission failure lemma 4.3.6, there exist a p_1 and a U_1 such that
  $p_1 \not\equiv p \wedge send(p_1, < T, s, \sigma >, l')$ $\mathbf{at_{p_1}}$ $U_1 \wedge U_1 \in (U - \delta - \epsilon, U - \gamma + \epsilon)$
  holds. Since $\gamma \geq 2\epsilon$, we have $\gamma > \epsilon$. Thus we obtain $U > U - \gamma + \epsilon$ and then $U > U_1$. By the faulty sender lemma 4.6.2, we have $\neg correct(p_1)$. Thus correct processor q is not that sender p_1.
  By $Relay(p)$, p will send $< T, s, \sigma >$ to q along link l within T_s time units. Thus we have $send(p, < T, s, \sigma >, l)$ $\mathbf{in_p}$ $[U, U + T_s]$.

By definition, there exists an X such that
$send(p, < T, s, \sigma >, l)$ $\mathbf{at_p}$ $X \wedge X \in [U, U + T_s)$
holds. By the bounded communication lemma 4.3.5, we obtain
$receive(q, < T, s, \sigma >, l)$ $\mathbf{in_q}$ $(X + \gamma - \epsilon, X + \delta + \epsilon)$.
Since $X \geq U$ and $U \geq T$, we have $X \geq T$. By $\gamma \geq 2\epsilon$, we obtain $X + \gamma - \epsilon \geq T$.
Together with $X < U + T_s$, we have proved
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{in_q}$ $[T, U + T_s + \delta + \epsilon)$, i.e.,
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{in_q}$ $[T, U + d(p, q)(T_s + \delta + \epsilon))$.

- $d(p, q) = k + 1$ with $k \geq 1$. By definition, there must exist a processor q_1 and a link l_2 such that $correct(q_1) \wedge correct(l_2) \wedge link(l_2, q_1, q) \wedge d(p, q_1) = k \wedge d(q_1, q) = 1$ holds. By the induction hypothesis, we have
$\exists l_1 : receive(q_1, < T, s, \sigma >, l_1)$ $\mathbf{in_{q_1}}$ $[T, U + k(T_s + \delta + \epsilon))$.
By definition, there exists a V_1 such that
$\exists l_1 : (receive(q_1, < T, s, \sigma >, l_1)$ $\mathbf{at_{q_1}}$ $V_1 \wedge V_1 \in [T, U + k(T_s + \delta + \epsilon))$).
Since $Firstrec(p, < T, s, \sigma >, l')$ $\mathbf{at_p}$ U and $\gamma \geq 2\epsilon$ holds, by the first correct receiving lemma 4.6.3, we have $U < T + m(T_s + \delta + 2\epsilon)$. Thus we obtain
$\exists l_1 : (receive(q_1, < T, s, \sigma >, l_1)$ $\mathbf{at_{q_1}}$ $V_1 \wedge V_1 \in [T, T + (k+m)(T_s + \delta) + (k+2m)\epsilon)$).
Since $T_r \geq (d + m - 1)(T_s + \delta) + (d + 2m - 1)\epsilon$ and $k \leq d - 1$ hold, we have
$\exists l_1 : (receive(q_1, < T, s, \sigma >, l_1)$ $\mathbf{at_{q_1}}$ $V_1 \wedge V_1 \in [T, T + T_r)$).
Since $correct(q)$ and $\neg correct(s)$ hold, we obtain $q \not\equiv s$.
By assumption, $\gamma \geq 2\epsilon$. Then by the propagation lemma 4.5.1, we have
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{in_q}$ $[T, V_1 + T_s + \delta + \epsilon)$, i.e.,
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{in_q}$ $[T, U + (k + 1)(T_s + \delta + \epsilon))$.
Therefore we have proved
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{in_q}$ $[T, U + d(p, q)(T_s + \delta + \epsilon))$.

Hence this lemma holds. □

Next lemma shows that if correct processor p learns of $< T, s, \sigma >$, then any correct processor q also learns of $< T, s, \sigma >$, provided $T_r \geq (d + m)(T_s + \delta) + (d + 2m)\epsilon$ and $\gamma > 2\epsilon$.

Lemma 4.6.5 (All Learn) If $T_r \geq (d + m)(T_s + \delta) + (d + 2m)\epsilon$ and $\gamma > 2\epsilon$, then
$correct(p) \wedge correct(q) \wedge Learn(p, < T, s, \sigma >) \rightarrow Learn(q, < T, s, \sigma >)$.

Proof: Assume that the premise of the lemma holds. By $Learn(p, < T, s, \sigma >)$, we have
$(initiate(p, \sigma)$ $\mathbf{at_p}$ $T \wedge p \equiv s) \vee$ (1)
$\exists l_1 : (receive(p, < T, s, \sigma >, l_1)$ $\mathbf{in_p}$ $[T, T + T_r) \wedge p \not\equiv s)$ (2)
From (2), since $\gamma > 2\epsilon$, by the initiation lemma 4.6.1, we obtain $initiate(s, \sigma)$ $\mathbf{at_s}$ T.

Since either (1) or (2) hold, we obtain $initiate(s, \sigma)$ $\mathbf{at_s}$ T from the premise.
We have to prove $Learn(q, < T, s, \sigma >)$, i.e., the following formula holds:

$$(initiate(q, \sigma) \ \mathbf{at_q}\ T \wedge q \equiv s) \vee \tag{3}$$

$$(\exists l_2 : receive(q, < T, s, \sigma >, l_2) \ \mathbf{in_q}\ [T, T + T_r) \wedge q \not\equiv s). \tag{4}$$

There are two possibilities:

- if $s \equiv q$, then we have $initiate(q, \sigma)$ $\mathbf{at_q}$ $T \wedge q \equiv s$ holds, i.e., (3) holds;
- if $s \not\equiv q$, we prove that (4) holds by the following two cases.

1. If $correct(s)$ holds, since $T_r \geq (d + m)(T_s + \delta) + (d + 2m)\epsilon$ and $\gamma > 2\epsilon$, by the bounded receiving lemma 4.5.2, we obtain
$\exists l_2 : receive(q, < T, s, \sigma >, l_2)$ $\mathbf{in_q}$ $[T, T + d(s, q)(T_s + \delta + \epsilon))$, i.e.,
$\exists l_2 : receive(q, < T, s, \sigma >, l_2)$ $\mathbf{in_q}$ $[T, T + T_r) \wedge q \not\equiv s$,
i.e., (4) holds.

2. If $\neg correct(s)$ holds, then by $receive(p, < T, s, \sigma >, l_1)$ $\mathbf{in_p}$ $[T, T + T_r)$, there exists a processor p_1 which is one of the first correct processors that have received $< T, s, \sigma >$ in the interval $[T, T + T_r)$ according to their own clocks. Thus, there exist l_3 and U such that
$Firstrec(p_1, < T, s, \sigma >, l_3)$ $\mathbf{at_{p_1}}$ $U \wedge U \in [T, T + T_r)$ holds.
Since $\gamma > 2\epsilon$, by the first correct receiving lemma 4.6.3, we obtain that p_1 receives $< T, s, \sigma >$ at local time U with $U < T + m(T_s + \delta + 2\epsilon)$.
Then we have also two cases:

   - if $q \equiv p_1$, then by $Firstrec(p_1, < T, s, \sigma >, l_3)$ $\mathbf{at_p}$ U, we have
   $receive(q, < T, s, \sigma >, l_3)$ $\mathbf{in_q}$ $[T, T + m(T_s + \delta + 2\epsilon))$, i.e.,
   $\exists l_2 : receive(q, < T, s, \sigma >, l_2)$ $\mathbf{in_q}$ $[T, T + m(T_s + \delta + 2\epsilon))$;
   - if $q \not\equiv p_1$, since $\gamma > 2\epsilon$, by the correct receiving lemma 4.6.4, we have
   $\exists l_2 : receive(q, < T, s, \sigma >, l_2)$ $\mathbf{in_q}$ $[T, U + d(p, q)(T_s + \delta + \epsilon))$, i.e.,
   $\exists l_2 : receive(q, < T, s, \sigma >, l_2)$ $\mathbf{in_q}$ $[T, T + m(T_s + \delta + 2\epsilon) + d(p, q)(T_s + \delta + \epsilon))$.

   Combining both cases, since $d(p, q) \leq d$, we obtain
   $\exists l_2 : receive(q, < T, s, \sigma >, l_2)$ $\mathbf{in_q}$ $[T, T + (d + m)(T_s + \delta) + (d + 2m)\epsilon)$.
   Since $T_r \geq (d + m)(T_s + \delta) + (d + 2m)\epsilon$, together with $s \not\equiv q$, we have
   $(\exists l_2 : receive(q, < T, s, \sigma >, l_2)$ $\mathbf{in_q}$ $[T, T + T_r) \wedge q \not\equiv s)$.

Thus for both cases, (4) holds.

Hence this lemma holds. □

Next lemma expresses that if correct processor p conveys $< T, s, \sigma >$ at some local time U, then any correct processor q conveys $< T, s, \sigma >$ in the interval $[T + T_r, T + T_r + T_c]$, provided $T_r \geq (d+m)(T_s + \delta) + (d + 2m)\epsilon$ and $\gamma > 2\epsilon$.

Lemma 4.6.6 (All Convey) If $T_r \geq (d+m)(T_s + \delta) + (d + 2m)\epsilon$ and $\gamma > 2\epsilon$, then

$$correct(p) \wedge correct(q) \wedge convey(p, < T, s, \sigma >) \ \mathbf{at_p} \ U \rightarrow$$
$$convey(q, < T, s, \sigma >) \ \mathbf{in_q} \ [T + T_r, T + T_r + T_c].$$

Proof: Assume that the premise of this lemma holds. By the server process specification axiom 4.4.1 and $correct(p)$, we have $Origin(p)$. From $Origin(p)$ and $convey(p, < T, s, \sigma >) \ \mathbf{at_p} \ U$, we obtain $Learn(p, < T, s, \sigma >)$. Since $T_r \geq (d+m)(T_s + \delta) + (d + 2m)\epsilon$ and $\gamma > 2\epsilon$, by the all learn lemma 4.6.5, we have $Learn(q, < T, s, \sigma >)$, i.e.,

$$(initiate(q, \sigma) \ \mathbf{at_q} \ T \wedge q \equiv s) \ \vee \quad (1)$$
$$(\exists l : receive(q, < T, s, \sigma >, l) \ \mathbf{in_q} \ [T, T + T_r) \wedge q \not\equiv s). \quad (2)$$

If (1) holds, by $Start(q)$, we have $convey(q, < T, s, \sigma >) \ \mathbf{in_q} \ [T + T_r, T + T_r + T_c]$.
If (2) holds, by $Relay(q)$, we have $convey(q, < T, s, \sigma >) \ \mathbf{in_q} \ [T + T_r, T + T_r + T_c]$.
Thus for both cases, we obtain $convey(q, < T, s, \sigma >) \ \mathbf{in_q} \ [T + T_r, T + T_r + T_c]$.
Hence this lemma holds. □

Next we prove a theorem which shows that the atomicity property follows from the axioms and lemmas given before.

Theorem 4.6.1 (Atomicity) If $T_r \geq (d+m)(T_s + \delta) + (d + 2m)\epsilon$, $\gamma > 2\epsilon$, and $D_2 \geq T_c$, then

$$correct(p) \wedge correct(q) \wedge deliver(p, \sigma) \ \mathbf{at_p} \ U \rightarrow$$
$$\exists s, T : initiate(s, \sigma) \ \mathbf{at_s} \ T \wedge deliver(q, \sigma) \ \mathbf{in_q} \ [U - D_2, U + D_2],$$

i.e., the atomicity property $ATOM$ holds.

Proof: Assume that the premise of the theorem holds. From $deliver(p, \sigma) \ \mathbf{at_p} \ U$, by definition, there exist s and T such that $convey(p, < T, s, \sigma >) \ \mathbf{at_p} \ U$ holds. By the server process specification axiom 4.4.1 and $correct(p)$, we have $Origin(p)$. By $Origin(p)$, we obtain

$Learn(p, < T, s, \sigma >) \wedge U \in [T + T_r, T + T_r + T_c]$, i.e.,

$$((initiate(p, \sigma) \ \mathbf{at_p} \ T \wedge p \equiv s) \ \vee \quad (1)$$
$$(\exists l : receive(p, < T, s, \sigma >, l) \ \mathbf{in_p} \ [T, T + T_r) \wedge p \not\equiv s)) \ \wedge \quad (2)$$
$$U \in [T + T_r, T + T_r + T_c]. \quad (3)$$

From (1), we have $initiate(s, \sigma) \ \mathbf{at_s} \ T$.
From (2), since $\gamma > 2\epsilon$, by the initiation lemma 4.6.1, we obtain $initiate(s, \sigma) \ \mathbf{at_s} \ T$.
Thus for both cases, we have

$$\exists s, T : initiate(s, \sigma) \textbf{ at}_{\mathbf{s}} \; T. \tag{4}$$

From $convey(p, < T, s, \sigma >) \textbf{ at}_{\mathbf{p}} \; U$, since $T_r \geq (d+m)(T_s + \delta) + (d + 2m)\epsilon$ and $\gamma > 2\epsilon$, by the all convey lemma 4.6.6, we have
$convey(q, < T, s, \sigma >) \textbf{ in}_{\mathbf{q}} \; [T + T_r, T + T_r + T_c]$.
From (3), we have $T \in [U - T_r - T_c, U - T_r]$.
Hence we obtain $convey(q, < T, s, \sigma >) \textbf{ in}_{\mathbf{q}} \; [U - T_c, U + T_c]$.
By definition, we obtain $deliver(q, \sigma) \textbf{ in}_{\mathbf{q}} \; [U - T_c, U + T_c]$.
Since $D_2 \geq T_c$, we have

$$deliver(q, \sigma) \textbf{ in}_{\mathbf{q}} \; [U - D_2, U + D_2]. \tag{5}$$

Combining (4) and (5), this theorem holds. □

4.7 Verification of Order

The order property of the atomic broadcast protocol will be proved in this section. We first give two lemmas which will be used to prove the order property.

The following lemma shows that, for any correct processors p and q, if p conveys $< T, s, \sigma >$ at local time U, q conveys $< T, s, \sigma >$ at local time V, and no update is delivered by p in the interval $[0, U)$, then there is also no update delivered by q in the interval $[0, V)$, provided $T_r \geq (d+m)(T_s + \delta) + (d + 2m)\epsilon$ and $\gamma > 2\epsilon$.

Lemma 4.7.1 (First Delivery) If $T_r \geq (d+m)(T_s + \delta) + (d + 2m)\epsilon$ and $\gamma > 2\epsilon$, then

$$\begin{aligned} & correct(p) \wedge convey(p, < T, s, \sigma >) \textbf{ at}_{\mathbf{p}} \; U \; \wedge \\ & correct(q) \wedge convey(q, < T, s, \sigma >) \textbf{ at}_{\mathbf{q}} \; V \; \wedge \\ & \neg deliver(p) \textbf{ in}_{\mathbf{p}} \; [0, U) \rightarrow \neg deliver(q) \textbf{ in}_{\mathbf{q}} \; [0, V). \end{aligned}$$

Proof: Assume that the premise of this lemma holds. Suppose $deliver(q) \textbf{ in}_{\mathbf{q}} \; [0, V)$ holds. By definition, there exist s_0, T_0, and V_0 such that
$convey(q, < T_0, s_0, \sigma_0 >) \textbf{ at}_{\mathbf{q}} \; V_0 \wedge V_0 \in [0, V)$ holds.
By assumption, we have $convey(q, < T, s, \sigma >) \textbf{ at}_{\mathbf{q}} \; V$.
From $V_0 < V$, by $Sequen(q)$, we obtain $(T_0, s_0) \sqsubset (T, s)$.
Since $T_r \geq (d+m)(T_s + \delta) + (d + 2m)\epsilon$ and $\gamma > 2\epsilon$, by the all convey lemma 4.6.6, we have $convey(p, < T_0, s_0, \sigma_0 >) \textbf{ in}_{\mathbf{p}} \; [T + T_r, T + T_r + T_c]$, i.e., there exists a $U_0 \in CVAL$ such that $convey(p, < T_0, s_0, \sigma_0 >) \textbf{ at}_{\mathbf{p}} \; U_0$ holds.
By assumption, we have $convey(p, < T, s, \sigma >) \textbf{ at}_{\mathbf{p}} \; U$.
Since $(T_0, s_0) \sqsubset (T, s)$, by $Sequen(p)$, we obtain $U_0 < U$.
From $U_0 \in CVAL$, we have $U_0 \geq 0$ and thus $U_0 \in [0, U)$.
Therefore we obtain $convey(p, < T_0, s_0, \sigma_0 >) \textbf{ at}_{\mathbf{p}} \; U_0 \wedge U_0 \in [0, U)$, i.e.,
$deliver(p, \sigma_0) \textbf{ in}_{\mathbf{p}} \; [0, U)$.

But by assumption, we have $\neg deliver(p)$ $\mathbf{in_p}$ $[0, U)$.
Thus it leads to contradiction and then $deliver(q)$ $\mathbf{in_q}$ $[0, V)$ does not hold, i.e.,
$\neg deliver(q)$ $\mathbf{in_q}$ $[0, V)$ holds.
Hence this lemma holds. □

Next lemma shows that, for any correct processors p and q, if p conveys $< T_1, s_1, \sigma_1 >$ at clock time U_1 and $< T_2, s_2, \sigma_2 >$ at clock time U_2, q conveys $< T_1, s_1, \sigma_1 >$ at clock time V_1 and $< T_2, s_2, \sigma_2 >$ at clock time V_2, and there is no update delivered by p in the interval (U_1, U_2), then there is also no update delivered by q in the interval (V_1, V_2), provided $T_r \geq (d+m)(T_s+\delta)+(d+2m)\epsilon$ and $\gamma > 2\epsilon$.

Lemma 4.7.2 (No Delivery) If $T_r \geq (d+m)(T_s+\delta)+(d+2m)\epsilon$ and $\gamma > 2\epsilon$, then

$$correct(p) \wedge convey(p, < T_1, s_1, \sigma_1 >)\ \mathbf{at_p}\ U_1 \wedge convey(p, < T_2, s_2, \sigma_2 >)\ \mathbf{at_p}\ U_2 \wedge$$
$$correct(q) \wedge convey(q, < T_1, s_1, \sigma_1 >)\ \mathbf{at_p}\ V_1 \wedge convey(q, < T_2, s_2, \sigma_2 >)\ \mathbf{at_p}\ V_2 \wedge$$
$$\neg deliver(p)\ \mathbf{in_p}\ (U_1, U_2) \rightarrow \neg deliver(q)\ \mathbf{in_q}\ (V_1, V_2).$$

Proof: Assume that the premise of this lemma holds. Suppose $deliver(q)$ $\mathbf{in_q}$ (V_1, V_2) holds. By definition, there exist s and T such that $convey(q, < T, s, \sigma >)$ $\mathbf{in_q}$ (V_1, V_2) holds. Then there exists a V such that $convey(q, < T, s, \sigma >)$ $\mathbf{at_q}$ $V \wedge V \in (V_1, V_2)$ holds.
By assumption, we have $convey(q, < T_1, s_1, \sigma_1 >)$ $\mathbf{at_p}$ V_1.
Since $V_1 < V$, by $Sequen(q)$, we obtain $(T_1, s_1) \sqsubset (T, s)$.
Similarly, from assumption, we have $convey(q, < T_2, s_2, \sigma_2 >)$ $\mathbf{at_p}$ V_2.
Since $V < V_2$, by $Sequen(q)$ again, we obtain $(T, s) \sqsubset (T_2, s_2)$.
From $convey(q, < T, s, \sigma >)$ $\mathbf{at_q}$ V, since $T_r \geq (d+m)(T_s+\delta)+(d+2m)\epsilon$ and $\gamma > 2\epsilon$, by the all convey lemma 4.6.6, we have $convey(p, < T, s, \sigma >)$ $\mathbf{in_p}$ $[T+T_r, T+T_r+T_c]$, i.e., there exists a U such that $convey(p, < T, s, \sigma >)$ $\mathbf{at_p}$ U holds.
By assumption, we have $convey(p, < T_1, s_1, \sigma_1 >)$ $\mathbf{at_p}$ U_1.
Since $(T_1, s_1) \sqsubset (T, s)$, by $Sequen(p)$, we obtain $U_1 < U$.
Similarly, from assumption, we have $convey(p, < T_2, s_2, \sigma_2 >)$ $\mathbf{at_p}$ U_2.
Since $(T, s) \sqsubset (T_2, s_2)$, by $Sequen(p)$, we obtain $U < U_2$.
Thus we obtain $convey(p, < T, s, \sigma >)$ $\mathbf{at_p}$ $U \wedge U \in (U_1, U_2)$.
By definition, we have $deliver(p, \sigma)$ $\mathbf{in_p}$ (U_1, U_2).
But from assumption, we have $\neg deliver(p)$ $\mathbf{in_p}$ (U_1, U_2).
Thus it leads to contradiction and then $deliver(q, \sigma)$ $\mathbf{in_q}$ (V_1, V_2) does not holds,
i.e., $\neg deliver(q)$ $\mathbf{in_q}$ (V_1, V_2) holds.
Hence this lemma holds. □

Next we prove, by the following theorem, that the order property holds.

Theorem 4.7.1 (Order) If $T_r \geq (d+m)(T_s+\delta)+(d+2m)\epsilon$ and $\gamma > 2\epsilon$, then
$$correct(p) \wedge correct(q) \rightarrow \forall U \exists V : List(p,U) \subseteq List(q,V),$$
i.e., the order property holds.

Proof: For any clock value $U \in CVAL$, assume $\langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle \in List(p,U)$. By definition, there exist $k \in I\!N^+$, $U_1, U_2, \ldots, U_k$ such that $U_1 \leq U_2 \leq \ldots \leq U_k < U$, $deliver(p,\sigma_i)$ $\mathbf{at_p}$ U_i, for $i = 1,2,\ldots,k$, $\neg deliver(p)$ $\mathbf{in_p}$ (U_j, U_{j+1}), for $j = 1,2,\ldots,k-1$, and $\neg deliver(p)$ $\mathbf{in_p}$ $[0, U_1)$. From $deliver(p,\sigma_i)$ $\mathbf{at_p}$ U_i, there exist s_i and T_i such that $convey(p, < T_i, s_i, \sigma_i >)$ $\mathbf{at_p}$ U_i holds. Let $V = U + T_c$. We prove that, by induction on k, there exist $V_1, V_2, \ldots, V_k$ such that $V_1 \leq V_2 \leq \ldots \leq V_k < V$, $convey(q, < T_i, s_i, \sigma_i >)$ $\mathbf{at_q}$ V_i, for $i = 1,2,\ldots,k$, $\neg deliver(q)$ $\mathbf{in_q}$ (V_j, V_{j+1}), for $j = 1,2,\ldots,k-1$, and $\neg deliver(q)$ $\mathbf{in_q}$ $[0, V_1)$ hold.

- $k = 1$. By assumption, we have $convey(p, < T_1, s_1, \sigma_1 >)$ $\mathbf{at_p}$ U_1 and $\neg deliver(p)$ $\mathbf{in_p}$ $[0, U_1)$.
  Since $T_r \geq (d+m)(T_s+\delta)+(d+2m)\epsilon$ and $\gamma > 2\epsilon$, by the all convey lemma 4.6.6, we obtain $convey(p, < T_1, s_1, \sigma_1 >)$ $\mathbf{in_p}$ $[T_1+T_r, T_1+T_r+T_c]$ and $convey(q, < T_1, s_1, \sigma_1 >)$ $\mathbf{in_q}$ $[T_1+T_r, T_1+T_r+T_c]$.
  Thus we have $U_1 \in [T_1+T_r, T_1+T_r+T_c]$. Since $U_1 < U$, we obtain $T_1 + T_r < U$.
  There exists a $V_1 \in CVAL$ such that
  $convey(q, < T_1, s_1, \sigma_1 >)$ $\mathbf{at_q}$ $V_1 \wedge V_1 \in [T_1+T_r, T_1+T_r+T_c]$ holds.
  Then we have $V_1 \leq T_1 + T_r + T_c$ and thus $V_1 < U + T_c$, i.e., $V_1 < V$.
  By the first deliver lemma 4.7.1, we also obtain $\neg deliver(q)$ $\mathbf{in_q}$ $[0, V_1)$.

- $k > 1$. By the induction hypothesis, there exist $V_1, V_2, \ldots, V_{k-1}$ such that $V_1 \leq V_2 \leq \ldots \leq V_{k-1}$, $convey(q, < T_i, s_i, \sigma_i >)$ $\mathbf{at_q}$ V_i, for $i = 1,2,\ldots,k-1$, $\neg deliver(q)$ $\mathbf{in_q}$ (V_j, V_{j+1}), for $j = 1,2,\ldots,k-2$, and $\neg deliver(q)$ $\mathbf{in_q}$ $[0, V_1)$ hold.
  By assumption, we have $convey(p, < T_k, s_k, \sigma_k >)$ $\mathbf{at_p}$ U_k.
  By the all convey lemma 4.6.6, we obtain that there exists a V_k such that $convey(q, < T_k, s_k, \sigma_k >)$ $\mathbf{at_q}$ $V_k \wedge V_k \in [T_k+T_r, T_k+T_r+T_c]$ holds.
  Since $U_{k-1} \leq U_k$, we prove $V_{k-1} \leq V_k$ by the following two cases.

  1. Assume $U_{k-1} < U_k$. By assumption, we have
     $convey(p, < T_{k-1}, s_{k-1}, \sigma_{k-1} >)$ $\mathbf{at_p}$ U_{k-1} and $convey(p, < T_k, s_k, \sigma_k >)$ $\mathbf{at_p}$ U_k.
     Since $U_{k-1} < U_k$, by $Sequen(p)$, we obtain $(T_{k-1}, s_{k-1}) \sqsubset (T_k, s_k)$.
     From the induction hypothesis and above, we have
     $convey(q, < T_{k-1}, s_{k-1}, \sigma_{k-1} >)$ $\mathbf{at_q}$ V_{k-1} and $convey(q, < T_k, s_k, \sigma_k >)$ $\mathbf{at_q}$ V_k.
     Since $(T_{k-1}, s_{k-1}) \sqsubset (T_k, s_k)$, by $Sequen(q)$, we obtain $V_{k-1} < V_k$.
  2. Assume $U_{k-1} = U_k$.
     Suppose $V_{k-1} < V_k$. Similar as above, we obtain $U_{k-1} < U_k$ which does not

hold.

Suppose $V_{k-1} > V_k$. Similarly, we obtain $U_{k-1} > U_k$ which also does not hold. Therefore only $V_{k-1} = V_k$ holds.

Combining these two cases, we obtain $V_{k-1} \leq V_k$.
Similar as the case for $k = 1$, we have $U_k \in [T_k + T_r, T_k + T_r + T_c]$ and $U_k < U$. Thus we obtain $T_k + T_r < U$. Since $V_k \leq T_k + T_r + T_c$, we have $V_k < U + T_c$, i.e., $V_k < V$.
By assumption, we have $\neg deliver(p)$ $\mathbf{in_p}$ (U_{k-1}, U_k).
Then by the no delivery lemma 4.7.2, we obtain $\neg deliver(q)$ $\mathbf{in_q}$ (V_{k-1}, V_k).

Hence we have proved that there exist $V_1, V_2, \ldots, V_k$ such that $V_1 \leq V_2 \leq \ldots \leq V_k < V$, $convey(q, < T_i, s_i, \sigma_i >)$ $\mathbf{at_q}$ V_i, for $i = 1, 2, \ldots, k$, $\neg deliver(q)$ $\mathbf{in_q}$ (V_j, V_{j+1}), for $j = 1, 2, \ldots, k-1$, and $\neg deliver(q)$ $\mathbf{in_q}$ $[0, V_1)$ hold.
Since $convey(q, < T_i, s_i, \sigma_i >)$ $\mathbf{at_q}$ V_i implies $deliver(q, \sigma_i)$ $\mathbf{at_q}$ V_i, we obtain $deliver(q, \sigma_i)$ $\mathbf{at_q}$ V_i, for $i = 1, 2, \ldots, k$.
Therefore we have $\langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle \in List(q, V)$.
Hence for any U there exists a V, i.e., $V = U + T_c$, such that $List(p, U) \subseteq List(q, V)$.
Thus this theorem holds. □

We have proved that, if $T_r \geq (d + m)(T_s + \delta) + (d + 2m)\epsilon$, $\gamma > 2\epsilon$, $D_1 \geq T_r + T_c$, and $D_2 \geq T_c$, then the termination, atomicity, and order properties hold. Since T_r is the minimum time to ensure that all correct processors have received a message containing an updates after it is initiated, we take $T_r = (d + m)(T_s + \delta) + (d + 2m)\epsilon$. Since D_1 is the broadcast termination time, it should be as small as possible and thus we take $D_1 = T_r + T_c$. Similarly, since D_2 indicates the difference of delivery times of an update by two correct processors, it should be also as small as possible and therefore we take $D_2 = T_c$.

Recall that AX is the conjunction of all axioms for the system, $Spec(p_i)$ is the specification for the server process running on processor p_i, and ABS is the top-level specification of the protocol, i.e., $ABS \equiv TERM \wedge ATOM \wedge ORDER$. Hence we have proved $\bigwedge_{i=1}^{n} Spec(p_i) \wedge AX \rightarrow ABS$, provided $T_r = (d + m)(T_s + \delta) + (d + 2m)\epsilon$, $\gamma > 2\epsilon$, $D_1 = T_r + T_c$, and $D_2 = T_c$.

4.8 Comparison

Comparing our paper with [CASD89], the basic ideas of proving properties of the protocol are similar. The assumptions and proofs presented in [CASD89] are simplified and informal. For instance, it is assumed there that when a correct processor p initiates an update, it takes zero time units for p to send a message to all its neighbors. In our framework, it takes at most T_s time units. Similarly, when p receives a message, [CASD89] assumes zero time units for p to relay the message to its neighbors, but we assume at most T_s time units. We also assume that p will take at most T_c time units to convey updates initiated at the same clock time to client processes.

Recall that d is the diameter of the graph consisting of all correct processors and links, m is the maximum number of faulty processors in the network, δ is the upper bound of message transmission delay between two correct processors as measured on any correct processor, and ϵ is the maximum deviation of local clocks of correct processors.

The minimum time to ensure that all correct processors have received a message containing an update after it is initiated is T_r in our paper with $T_r = (d+m)(T_s + \delta) + (d+2m)\epsilon$, which is more detailed than that in [CASD89], where it is Δ with $\Delta = (d+m)\delta + \epsilon$. If we assume $T_s = 0$, then we have $T_r = (d+m)\delta + (d+2m)\epsilon$ and thus T_r is similar as Δ except the part concerning ϵ. Consequently, the broadcast termination time in our framework, which is D_1 with $D_1 = T_r + T_c$, is not exactly the same as that in [CASD89], which is Δ. If we also assume $T_c = 0$, then we have $D_1 = T_r$ and thus D_1 is similar as Δ.

In this paper we express the termination property by using $deliver(q,\sigma)$ $\mathbf{by_q}$ $T + D_1$ instead of $deliver(q,\sigma)$ $\mathbf{at_q}$ $T + D_1$. In the termination theorem 4.5.1, we have proved that if $initiate(s,\sigma)$ $\mathbf{at_s}$ T, then $deliver(q,\sigma)$ $\mathbf{in_q}$ $[T + T_r, T + T_r + T_c]$. If we assume $T_c = 0$, since $D_1 = T_r + T_c$, we obtain $deliver(q,\sigma)$ $\mathbf{at_q}$ $T + D_1$. Therefore the termination property described here can be reduced to that in [CASD89] if $T_c = 0$.

Similarly, if $T_c = 0$, then the atomicity property expressed in this paper can also be reduced to that in [CASD89]. In the atomicity theorem 4.6.1, we have proved that if $deliver(p,\sigma)$ $\mathbf{at_p}$ U, then $deliver(q,\sigma)$ $\mathbf{in_q}$ $[U - T_c, U + T_c]$. If $T_c = 0$, then we obtain $deliver(q,\sigma)$ $\mathbf{at_q}$ U.

To prove the atomicity property, we need to show that if a correct processor p delivers σ at some time U, then σ was initiated by some processor s at some clock time T. This is not proved in [CASD89]. We have proved it in lemma 4.6.1 by using available timing information. There we need a lower bound for message transmission delay between two correct processors. Thus we add a lower bound γ in the bounded communication axiom 4.3.5. This lower bound is also used in other lemmas, e.g. the propagation lemma 4.5.1 and the first correct receiving lemma 4.6.3.

The behavior of any processor p is specified by the fail silence axiom 4.3.7 and the server process specification axiom 4.4.1. Notice that axiom 4.3.7 and formula $Source(p)$ hold for any arbitrary processor p, i.e., even if p is faulty. To prove the atomicity property, we have to show that if a correct processor p delivers an update σ at local time U, then σ was initiated by some processor and σ will be delivered by each correct processor in the interval $[U-D_2, U+D_2]$ according to their own clocks. By the initiation lemma 4.6.1 and $Origin(p)$, we can prove that there exists a processor s which initiates σ at some local time T. If s is correct, by the server process specification axiom 4.4.1, we have $Start(s)$, $Relay(s)$, and $Origin(s)$. Then we can derive that each correct processor will deliver σ in the interval $[U - D_2, U + D_2]$. But if s is not correct, all we have is $Source(s)$ and axiom 4.3.7. Then we can only use them and other axioms to reason backwards to prove the atomicity property. This idea is represented in the first correct receiving lemma 4.6.3.

In [CASD89], it is required that a processor will relay a message to its neighbors only if it receives the message for the first time. We do not require this in our paper. When a processor receives a message it will always relay the message to its neighbors. The requirement in [CASD89] is to make the server process more efficient and avoid memory overflow. Since we focus ourselves on the correctness of the protocol, this is not considered here.

An assumption mentioned in [CASD89], but not in this paper, is that the resolution of processor clocks is fine enough so that separate clock readings yield different values. This is an assumption for the implementation of the protocol. In this paper, we only express those assumptions needed for our verification and nothing more. Therefore another assumption of [CASD89], namely that there is a finite bound on the number of messages any processor can send per time unit, is also not included.

Just before the deadline of this thesis, we received the comments on this chapter from the first author of [CASD89]. According to [Cri93], the clock synchronization assumption can be made to all local clocks of processors, not only to local clocks of correct processors, since we only allow omission failures in the protocol. If a local clock could suffer from omission failures, the processor having that clock could exhibit Byzantine behavior (e.g. timestamp different updates with the same timestamp). Thus the clock synchronization axiom 4.3.6 can be strengthened as

$$|C_p(t) - C_q(t)| < \epsilon$$

Lemma 4.3.1 then can be removed.

Having done this, some axioms and lemmas can be simplified and their proofs will be easier. For instance, the only omission failure axiom 4.3.8 will look like

$$correct(q) \ \mathbf{at_r} \ V \wedge receive(q,m,l) \ \mathbf{at_r} \ V \rightarrow \exists p \not\equiv q : send(p,m,l) \ \mathbf{in_r} \ [V - \delta, V - \gamma]$$

And the only omission failure lemma 4.3.6 will become

$$correct(q) \ \mathbf{at_q} \ V \wedge receive(q,m,l) \ \mathbf{at_q} \ V \rightarrow \exists p \not\equiv q : send(p,m,l) \ \mathrm{in_p} \ [V-\delta-\epsilon, V-\gamma+\epsilon].$$

Chapter 5

Conclusions

5.1 Summary

In chapters 2 and 3 of this thesis, we developed two versions of a formalism to specify and verify real-time systems, one of which was for synchronously communicating real-time systems and the other was for asynchronously communicating real-time systems. We started with two versions of an Occam-like programming language. One version contained synchronous communication primitives and the other included asynchronous communication primitives. We gave a compositional semantics for this programming language. The specification language (also with two versions according to the communication mechanism) for systems written in this programming language was based on Explicit Clock Temporal Logic (ECTL). A compositional proof system was formulated for each version of the programming and specification languages. These two proof systems were shown to be sound with respect to the semantics and relatively complete with respect to a proof system for ECTL. We also demonstrated the use of the formalism for synchronous communication by specifying and verifying a small part of an avionics system.

In chapter 4, we specified and verified an atomic broadcast protocol tolerating omission failures. As we saw in this thesis, using ECTL-based formalism to reason about properties was not easy. We would like to describe the protocol in an intuitive and informal way. Therefore the specification language for the protocol was not based on ECTL but on first-order logic. We described the top-level requirements of the atomic broadcast protocol and the server process in the specification language. We also axiomatized the lower level communication mechanism, clock synchronization assumptions, and failure assumptions. Thereafter we proved, by using an assertional, compositional approach, that parallel execution of the server processes on a network of distributed processors satisfied the top-level specification of the protocol. Hence we formally verified the protocol which was only informally proved in [CASD89]. This increased our confidence that the properties of the protocol were indeed guaranteed by the parallel execution of the server processes.

Notice that, in the top-level specification of the protocol, in the axioms about the service system, and in the server process specification, we used local clock values instead of global clock values. An essential idea of the atomic broadcast protocol was that the messages used to broadcast among processors contained time stamps which recorded the initiation time of updates. These time stamps were in terms of local clocks and were used to achieve the so-called order property of the protocol. Following [CASD89], other properties of the system, for instance the bounded communication axiom and the only omission failure axiom, were also expressed using local clocks. This suggested that reasoning about the protocol in terms of local clocks would be easy and natural. After verifying the protocol, this turned out to be true. The clock synchronization assumption for correct processors made the specification and verification of the protocol in terms of local clocks values meaningful. This is new in real-time specification and verification, since many formal methods only use global clock values, see e.g. [BHRR91].

Also observe that the formal method we used is compositional. This enables us to use only the specification of the server process to verify the protocol, without knowing any implementation details of the server process. Thus we can separate the concern of implementing the server from the concern of formal verification of the protocol.

As we have seen from this thesis, specifying and verifying real-time fault-tolerant systems are not easy. Applications of the ECTL-based proof systems show that proving a simple process correct needs a lot of effort. Moreover, the specification language contains the chop operator $\mathcal{C}$ and the iterated chop operator $\mathcal{C}^*$ which make the reasoning even more difficult. However, in [RP86] there are some nice axioms and rules for the chop operator, for example: $(\varphi_1 \mathcal{C} \varphi_2) \mathcal{C} \varphi_3 = \varphi_1 \mathcal{C} (\varphi_2 \mathcal{C} \varphi_3)$, $(\varphi_1 \vee \varphi_2) \mathcal{C} \varphi_3 = \varphi_1 \mathcal{C} \varphi_3 \ \vee \ \varphi_2 \mathcal{C} \varphi_3$, $\varphi_1 \mathcal{C} (\varphi_2 \vee \varphi_3) = \varphi_1 \mathcal{C} \varphi_2 \ \vee \ \varphi_1 \mathcal{C} \varphi_3$, etc., where φ_i, for $i = 1, 2, 3$, are formulae interpreted over sequences of states. Furthermore, one of our aims in this thesis is to formulate a compositional proof system which can provide elegant rules for compound statements including sequential composition and iteration. As shown in the thesis, it is reasonably easy to derive properties from formulae containing chop operators in an intuitive way or by reasoning at the semantic level.

5.2 Related Work

We mention some research results which are related to our work. In [Lam83a], interesting examples, e.g., the alternating bit protocol, are specified using generalized temporal logic (i.e., with predicates), but time is not considered. Compositional proof systems based on temporal logic can be found in [BKP84,BKP85,NDGO86], where time is also not concerned. Untimed modular verification of communication protocols (including the alternating bit protocol) using temporal logic and history variables is shown in [HO83]. How to compose untimed specifications are extensively discussed in [AL90], where the precise distinction between a system and its specification is examined. In [AL92], problems arised in real-time systems are addressed and a formal framework provided by TLA (the Temporal Logic of Actions) is used to study these problems. A state-based, compositional semantics for real-time programs is proposed in [GJ88], where it models termination, failure, divergence, deadlock, and startvation. A distributed real-time arbitration protocol is verified compositionally in [Hoo93], which follows the same principle presented in this thesis. Real-time extensions of CCS [Mil89] are proposed in [MT90, Yi91]. A hierarchy of untimed and timed models for CSP [Hoa85] is presented in [Ree89], which enables one to reason about concurrent processes in a uniform fashion with the minimum of complexity. A complete set of inference rules for reasoning about timed CSP processes is given in [DS89]. Untimed process algebra for synchronous communication in [BK84] is extended with real-time in [BB91]. Another algebra for timed processes is suggested in [NRSV90]. A calculus of durations to reason about design and requirements for real-time systems, which is an extension of Interval Temporal Logic, can be found in [CHR91]. This calculus is used in [CHRR92] to express specifications for shared processors. Process algebras dealing with asynchronous communicatiom mechanism appear in [Mil83,BKT85,JJH90,BB92]. A trace-based model and proof system for asynchronous network is presented in [Jon85]. A compositional semantics for an asynchronous version of CSP can be found in [BH92].

There is also some progress on the specification and verification of (real-time and) fault-tolerant systems. A rigorous programming approach for fault-tolerant systems is presented in [Cri85], where only sequential programs are considered. A compositional proof system for fault-tolerant programs written in a CSP-like language are shown in [JMS87]. Mechanical verification of a Byzantine fault-tolerant algorithm for clock synchronization is described in [RH91,Sha92]. A reliable broadcast protocol proposed in [CM84] is formally verified in [Yod92], where the so called "modal primitive recursive" functions are used. In [Pel91] CSP is used to design and verify fault-tolerant systems. Deontic logic is applied in [Coe92] to specify layered fault-tolerant systems in a natural way. A compositional semantics for fault-tolerant real-time systems appears in [CH92], where the occurrence of failures are allowed and the effect of these failures is described in the real-time behavior of programs. Fault-tolerant real-time systems are specified using "Minimal Three-Sorted Modal Logic" in [CW92]. A trace-based compositional network proof theory for fault-tolerant systems is shown in [SH93], where the fault hypothesis which specifies the class of faults that must be tolerated is an important feature. This is also a key point in a traced-based compositional framework for refinement of fault-tolerant system proposed in [SC93]. Exception handling in process algebra can be found in [BCG92], where ACP [BK84] is extended with an exception handling construct and the theory is applied to an fault-tolerant system presented in [Pel91].

Appendix A

Proofs of Lemmas in Chapter 2

Proof of Lemma 2.6.1

Consider any expression e from the programming language, any model σ, and any $\tau \geq begin(\sigma)$. We prove $\mathcal{E}(e)(\sigma(\tau).s) = \mathcal{V}(e)(\sigma, \tau)$ by induction on the structure of e.

- $e \equiv \vartheta$. $\mathcal{E}(\vartheta)(\sigma(\tau).s) = \vartheta = \mathcal{V}(\vartheta)(\sigma, \tau)$.

- $e \equiv x$. $\mathcal{E}(x)(\sigma(\tau).s) = \sigma(\tau).s(x) = \mathcal{V}(x)(\sigma, \tau)$.

- $e \equiv e_1 \odot e_2$, where $\odot \in \{+, -, \times\}$. By the induction hypothesis, we have, for $i = 1, 2$, $\mathcal{E}(e_i)(\sigma(\tau).s) = \mathcal{V}(e_i)(\sigma, \tau)$. Then $\mathcal{E}(e_1 \odot e_2)(\sigma(\tau).s) =$
  $\mathcal{E}(e_1)(\sigma(\tau).s) \odot \mathcal{E}(e_2)(\sigma(\tau).s) = \mathcal{V}(e_1)(\sigma, \tau) \odot \mathcal{V}(e_2)(\sigma, \tau) = \mathcal{V}(e_1 \odot e_2)(\sigma, \tau)$.

Proof of Lemma 2.6.2

Consider any boolean guard g from the programming language, any model σ, and any $\tau \geq begin(\sigma)$. We prove $\mathcal{G}(g)(\sigma(\tau).s)$ iff $\langle \sigma, \tau \rangle \models g$ by induction on the structure of g.

- $g \equiv e_1 = e_2$. $\mathcal{G}(e_1 = e_2)(\sigma(\tau).s)$ iff $\mathcal{E}(e_1)(\sigma(\tau).s) = \mathcal{E}(e_2)(\sigma(\tau).s)$ iff, by lemma 2.6.1, $\mathcal{V}(e_1)(\sigma, \tau) = \mathcal{V}(e_2)(\sigma, \tau)$ iff $\langle \sigma, \tau \rangle \models e_1 = e_2$.

- $g \equiv e_1 < e_2$. Similar to the proof for $g \equiv e_1 = e_2$.

- $g \equiv \neg g_1$. $\mathcal{G}(\neg g_1)(\sigma(\tau).s)$ iff *not* $\mathcal{G}(g_1)(\sigma(\tau).s)$ iff, by the induction hypothesis, *not* $\langle \sigma, \tau \rangle \models g_1$ iff $\langle \sigma, \tau \rangle \models \neg g_1$.

- $g \equiv g_1 \vee g_2$. $\mathcal{G}(g_1 \vee g_2)(\sigma(\tau).s)$ iff $\mathcal{G}(g_1)(\sigma(\tau).s)$ or $\mathcal{G}(g_2)(\sigma(\tau).s)$ iff, by the induction hypothesis, $\langle \sigma, \tau \rangle \models g_1$ or $\langle \sigma, \tau \rangle \models g_2$ iff $\langle \sigma, \tau \rangle \models g_1 \vee g_2$.

Proof of Lemma 2.6.3

Consider any expression *vexp* of type *VAL*, any model σ, any *cset* $\subseteq$ *DCHAN*, and any $\tau \geq begin(\sigma)$. We prove $\mathcal{V}(vexp)(\sigma,\tau) = \mathcal{V}(vexp)([\sigma]_{cset},\tau)$ by induction on the structure of *vexp*.

- $vexp \equiv \vartheta$. $\mathcal{V}(\vartheta)(\sigma,\tau) = \vartheta = \mathcal{V}(\vartheta)([\sigma]_{cset},\tau)$.

- $vexp \equiv x$. By definition, if $\tau \leq end(\sigma)$, then $\sigma(\tau).s(x) = [\sigma]_{cset}(\tau).s(x)$, i.e., if $\tau \leq end([\sigma]_{cset})$, then $\mathcal{V}(x)(\sigma,\tau) = \mathcal{V}(x)([\sigma]_{cset},\tau)$.
  If $\tau > end(\sigma)$, then $\mathcal{V}(x)(\sigma,\tau) = \sigma^e.s(x) = [\sigma]^e_{cset}.s(x)$, i.e.,
  if $\tau > end([\sigma]_{cset})$, then $\mathcal{V}(x)(\sigma,\tau) = \mathcal{V}(x)([\sigma]_{cset},\tau)$.
  Hence $\mathcal{V}(x)(\sigma,\tau) = \mathcal{V}(x)([\sigma]_{cset},\tau)$.

- $vexp \equiv first(x)$. $\mathcal{V}(first(x))(\sigma,\tau) = \sigma^b.s(x) = [\sigma]^b_{cset}.s(x) = \mathcal{V}(first(x))([\sigma]_{cset},\tau)$.

- $vexp \equiv last(x)$. If $end(\sigma) < \infty$, then $\mathcal{V}(last(x))(\sigma,\tau) = \sigma^e.s(x) = [\sigma]^e_{cset}.s(x) = \mathcal{V}(last(x))([\sigma]_{cset},\tau)$. If $end(\sigma) = \infty$, then $\mathcal{V}(last(x))(\sigma,\tau) = \sigma^b.s(x) = [\sigma]^b_{cset}.s(x) = \mathcal{V}(last(x))([\sigma]_{cset},\tau)$.

- $vexp \equiv max(vexp_1, vexp_2)$. By the induction hypothesis, we have, for $i = 1,2$, $\mathcal{V}(vexp_i)(\sigma,\tau) = \mathcal{V}(vexp_i)([\sigma]_{cset},\tau)$. Then
  $\mathcal{V}(max(vexp_1,vexp_2))(\sigma,\tau) = max(\mathcal{V}(vexp_1)(\sigma,\tau), \mathcal{V}(vexp_2)(\sigma,\tau))$
  $= max(\mathcal{V}(vexp_1)([\sigma]_{cset},\tau), \mathcal{V}(vexp_2)([\sigma]_{cset},\tau)) = \mathcal{V}(max(vexp_1,vexp_2))([\sigma]_{cset},\tau)$.

- $vexp \equiv vexp_1 \odot vexp_2$, where $\odot \in \{+,-,\times\}$. By the induction hypothesis, we have, for $i = 1,2$, $\mathcal{V}(vexp_i)(\sigma,\tau) = \mathcal{V}(vexp_i)([\sigma]_{cset},\tau)$. Thus
  $\mathcal{V}(vexp_1 \odot vexp_2)(\sigma,\tau) = \mathcal{V}(vexp_1)(\sigma,\tau) \odot \mathcal{V}(vexp_2)(\sigma,\tau)$
  $= \mathcal{V}(vexp_1)([\sigma]_{cset},\tau) \odot \mathcal{V}(vexp_2)([\sigma]_{cset},\tau) = \mathcal{V}(vexp_1 \odot vexp_2)([\sigma]_{cset},\tau)$.

Proof of Lemma 2.6.4

Consider any expression *vexp* of type *VAL*, any model σ, any *vset* $\subseteq$ *VAR*, and any $\tau \geq begin(\sigma)$. We prove, by induction on *vexp*, that if $var(vexp) \subseteq vset$, then $\mathcal{V}(vexp)(\sigma,\tau) = \mathcal{V}(vexp)(\sigma \downarrow vset,\tau)$.

- $vexp \equiv \vartheta$. $\mathcal{V}(\vartheta)(\sigma,\tau) = \vartheta = \mathcal{V}(\vartheta)(\sigma \downarrow vset,\tau)$.

- $vexp \equiv x$. $var(vexp) = \{x\}$ and thus $x \in vset$. By definition, if $\tau \leq end(\sigma)$, then $\sigma(\tau).s(x) = (\sigma \downarrow vset)(\tau).s(x)$, i.e., if $\tau \leq end(\sigma \downarrow vset)$, then $\mathcal{V}(x)(\sigma,\tau) = \mathcal{V}(x)(\sigma \downarrow vset,\tau)$. If $\tau > end(\sigma)$, then $\mathcal{V}(x)(\sigma,\tau) = \sigma^e.s(x) = (\sigma \downarrow vset)^e.s(x)$, i.e., if $\tau > end(\sigma \downarrow vset)$, then $\mathcal{V}(x)(\sigma,\tau) = \mathcal{V}(x)(\sigma \downarrow vset,\tau)$.
  Hence $\mathcal{V}(x)(\sigma,\tau) = \mathcal{V}(x)(\sigma \downarrow vset,\tau)$.

- $vexp \equiv first(x)$. $var(vexp) = \{x\}$ and then $x \in vset$. Thus $\mathcal{V}(first(x))(\sigma, \tau) = \sigma^b.s(x) = (\sigma \downarrow vset)^b.s(x) = \mathcal{V}(first(x))(\sigma \downarrow vset, \tau)$.

- $vexp \equiv last(x)$. $var(vexp) = \{x\}$ and then $x \in vset$. If $end(\sigma) < \infty$, then $\mathcal{V}(last(x))(\sigma, \tau) = \sigma^e.s(x) = (\sigma \downarrow vset)^e.s(x) = \mathcal{V}(last(x))(\sigma \downarrow vset, \tau)$.
  If $end(\sigma) = \infty$, then $\mathcal{V}(last(x))(\sigma, \tau) = \sigma^b.s(x) = (\sigma \downarrow vset)^b.s(x) = \mathcal{V}(last(x))(\sigma \downarrow vset, \tau)$.

- $vexp \equiv max(vexp_1, vexp_2)$. For $i = 1, 2$, $var(vexp_i) \subseteq var(vexp) \subseteq vset$. Then by the induction hypothesis, $\mathcal{V}(vexp_i)(\sigma, \tau) = \mathcal{V}(vexp_i)(\sigma \downarrow vset, \tau)$. Then
  $\mathcal{V}(max(vexp_1, vexp_2))(\sigma, \tau) = max(\mathcal{V}(vexp_1)(\sigma, \tau), \mathcal{V}(vexp_2)(\sigma, \tau)) =$
  $max(\mathcal{V}(vexp_1)(\sigma \downarrow vset, \tau), \mathcal{V}(vexp_2)(\sigma \downarrow vset, \tau)) =$
  $\mathcal{V}(max(vexp_1, vexp_2))(\sigma \downarrow vset, \tau)$.

- $vexp \equiv vexp_1 \odot vexp_2$, where $\odot \in \{+, -, \times\}$. For $i = 1, 2$, $var(vexp_i) \subseteq var(vexp) \subseteq vset$. Then by the induction hypothesis,
  $\mathcal{V}(vexp_i)(\sigma, \tau) = \mathcal{V}(vexp_i)(\sigma \downarrow vset, \tau)$. Thus
  $\mathcal{V}(vexp_1 \odot vexp_2)(\sigma, \tau) = \mathcal{V}(vexp_1)(\sigma, \tau) \odot \mathcal{V}(vexp_2)(\sigma, \tau)$
  $= \mathcal{V}(vexp_1)(\sigma \downarrow vset, \tau) \odot \mathcal{V}(vexp_2)(\sigma \downarrow vset, \tau) = \mathcal{V}(vexp_1 \odot vexp_2)(\sigma \downarrow vset, \tau)$.

Proof of Lemma 2.6.5

Consider any expression $texp$ of type *TIME*, any model σ, any $cset \subseteq DCHAN$, and any $\tau \geq begin(\sigma)$. We prove $\mathcal{T}(texp)(\sigma, \tau) = \mathcal{T}(texp)([\sigma]_{cset}, \tau)$ by induction on the structure of $texp$.

- $texp \equiv \hat{\tau}$. $\mathcal{T}(\hat{\tau})(\sigma, \tau) = \hat{\tau} = \mathcal{T}(\hat{\tau})([\sigma]_{cset}, \tau)$.

- $texp \equiv T$. $\mathcal{T}(T)(\sigma, \tau) = \tau = \mathcal{T}(T)([\sigma]_{cset}, \tau)$.

- $texp \equiv start$. $\mathcal{T}(start)(\sigma, \tau) = begin(\sigma) = begin([\sigma]_{cset}) = \mathcal{T}(start)([\sigma]_{cset}, \tau)$.

- $texp \equiv term$. $\mathcal{T}(term)(\sigma, \tau) = end(\sigma) = end([\sigma]_{cset}) = \mathcal{T}(term)([\sigma]_{cset}, \tau)$.

- $texp \equiv vexp$. By lemma 2.6.3, we have $\mathcal{V}(vexp)(\sigma, \tau) = \mathcal{V}(vexp)([\sigma]_{cset}, \tau)$.
  Then $\mathcal{T}(vexp)(\sigma, \tau) = \mathcal{V}(vexp)(\sigma, \tau) = \mathcal{V}(vexp)([\sigma]_{cset}, \tau) = \mathcal{T}(vexp)([\sigma]_{cset}, \tau)$.

- $texp \equiv texp_1 \odot texp_2$, where $\odot \in \{+, -, \times\}$. By the induction hypothesis, we have, for $i = 1, 2$, $\mathcal{T}(texp_i)(\sigma, \tau) = \mathcal{T}(texp_i)([\sigma]_{cset}, \tau)$. Then, by definition, $\mathcal{T}(texp_1 \odot texp_2)(\sigma, \tau) = \mathcal{T}(texp_1 \odot texp_2)([\sigma]_{cset}, \tau)$.

Proof of Lemma 2.6.6

Consider any expression *texp* of type *TIME*, any model σ, any $vset \subseteq VAR$, and any $\tau \geq begin(\sigma)$. We prove, by induction on *texp*, that if $var(texp) \subseteq vset$, then $\mathcal{T}(texp)(\sigma, \tau) = \mathcal{T}(texp)(\sigma \downarrow vset, \tau)$.

- $texp \equiv \hat{\tau}$. $\mathcal{T}(\hat{\tau})(\sigma, \tau) = \hat{\tau} = \mathcal{T}(\hat{\tau})(\sigma \downarrow vset, \tau)$.
- $texp \equiv T$. $\mathcal{T}(T)(\sigma, \tau) = \tau = \mathcal{T}(T)(\sigma \downarrow vset, \tau)$.
- $texp \equiv start$. $\mathcal{T}(start)(\sigma, \tau) = begin(\sigma) = begin(\sigma \downarrow vset) = \mathcal{T}(start)(\sigma \downarrow vset, \tau)$.
- $texp \equiv term$. $\mathcal{T}(term)(\sigma, \tau) = end(\sigma) = end(\sigma \downarrow vset) = \mathcal{T}(term)(\sigma \downarrow vset, \tau)$.
- $texp \equiv vexp$. $var(texp) = var(vexp)$ and thus $var(vexp) \subseteq vset$. By lemma 2.6.4, $\mathcal{V}(vexp)(\sigma, \tau) = \mathcal{V}(vexp)(\sigma \downarrow vset, \tau)$. Then $\mathcal{T}(vexp)(\sigma, \tau) = \mathcal{V}(vexp)(\sigma, \tau) = \mathcal{V}(vexp)(\sigma \downarrow vset, \tau) = \mathcal{T}(vexp)(\sigma \downarrow vset, \tau)$.
- $texp \equiv texp_1 \odot texp_2$, where $\odot \in \{+, -, \times\}$. For $i = 1, 2$, $var(texp_i) \subseteq var(texp) \subseteq vset$. By the induction hypothesis, $\mathcal{T}(texp_i)(\sigma, \tau) = \mathcal{T}(texp_i)(\sigma \downarrow vset, \tau)$. Then, by definition, $\mathcal{T}(texp_1 \odot texp_2)(\sigma, \tau) = \mathcal{T}(texp_1 \odot texp_2)(\sigma \downarrow vset, \tau)$.

Proof of Lemma 2.6.7

Consider any $cset \subseteq DCHAN$ and any specification φ. We prove that if $dch(\varphi) \subseteq cset$ then, for any model σ and any $\tau \geq begin(\sigma)$, $\langle \sigma, \tau \rangle \models \varphi$ iff $\langle [\sigma]_{cset}, \tau \rangle \models \varphi$, by induction on the structure of φ.

- $\varphi \equiv texp_1 = texp_2$. $\langle \sigma, \tau \rangle \models texp_1 = texp_2$ iff $\mathcal{T}(texp_1)(\sigma, \tau) = \mathcal{T}(texp_2)(\sigma, \tau)$ iff, by lemma 2.6.5, $\mathcal{T}(texp_1)([\sigma]_{cset}, \tau) = \mathcal{T}(texp_2)([\sigma]_{cset}, \tau)$ iff $\langle [\sigma]_{cset}, \tau \rangle \models texp_1 = texp_2$.
- $\varphi \equiv texp_1 < texp_2$. Similar to the proof for $\varphi \equiv texp_1 = texp_2$.
- $\varphi \equiv comm(c, vexp)$. $dch(\varphi) = \{c\}$ and thus $c \in cset$. Hence $\langle \sigma, \tau \rangle \models comm(c, vexp)$ iff $\tau < end(\sigma)$ and $(c, \mathcal{V}(vexp)(\sigma, \tau)) \in \sigma(\tau).c$ iff, by definition and lemma 2.6.3, $\tau < end([\sigma]_{cset})$ and $(c, \mathcal{V}(vexp)([\sigma]_{cset}, \tau)) \in [\sigma]_{cset}(\tau).c$ iff $\langle [\sigma]_{cset}, \tau \rangle \models comm(c, vexp)$.
- $\varphi \equiv comm(c)$. $dch(\varphi) = \{c\}$ and thus $c \in cset$. Hence $\langle \sigma, \tau \rangle \models comm(c)$ iff $\tau < end(\sigma)$ and there exists a value ϑ such that $(c, \vartheta) \in \sigma(\tau).c$ iff $\tau < end([\sigma]_{cset})$ and there exists a value ϑ such that $(c, \vartheta) \in [\sigma]_{cset}(\tau).c$ iff $\langle [\sigma]_{cset}, \tau \rangle \models comm(c)$.

- $\varphi \equiv wait(c!)$. $dch(\varphi) = \{c!\}$ and then $c! \in cset$. Hence $\langle\sigma,\tau\rangle \models wait(c!)$ iff $\tau < end(\sigma)$ and $c! \in \sigma(\tau).c$ iff $\tau < end([\sigma]_{cset})$ and $c! \in [\sigma]_{cset}(\tau).c$ iff $\langle[\sigma]_{cset},\tau\rangle \models wait(c!)$.

- $\varphi \equiv wait(c?)$. $dch(\varphi) = \{c?\}$ and then $c? \in cset$. Hence $\langle\sigma,\tau\rangle \models wait(c?)$ iff $\tau < end(\sigma)$ and $c? \in \sigma(\tau).c$ iff $\tau < end([\sigma]_{cset})$ and $c? \in [\sigma]_{cset}(\tau).c$ iff $\langle[\sigma]_{cset},\tau\rangle \models wait(c?)$.

- $\varphi \equiv \varphi_1 \vee \varphi_2$. For $i = 1,2$, we have $dch(\varphi_i) \subseteq (dch(\varphi_1) \cup dch(\varphi_2)) = dch(\varphi) \subseteq cset$. Hence $\langle\sigma,\tau\rangle \models \varphi_1 \vee \varphi_2$ iff $\langle\sigma,\tau\rangle \models \varphi_1$ or $\langle\sigma,\tau\rangle \models \varphi_2$ iff, by the induction hypothesis, $\langle[\sigma]_{cset},\tau\rangle \models \varphi_1$ or $\langle[\sigma]_{cset},\tau\rangle \models \varphi_2$ iff $\langle[\sigma]_{cset},\tau\rangle \models \varphi_1 \vee \varphi_2$.

- $\varphi \equiv \neg\varphi_1$ and $\varphi \equiv \varphi_1\ \mathcal{U}\ \varphi_2$. Similar to the proof for $\varphi \equiv \varphi_1 \vee \varphi_2$.

- $\varphi \equiv \varphi_1\ \mathcal{C}\ \varphi_2$. For $i = 1,2$, we have $dch(\varphi_i) \subseteq dch(\varphi) \subseteq cset$.
  Hence $\langle\sigma,\tau\rangle \models \varphi_1\ \mathcal{C}\ \varphi_2$ iff
  - either $\langle\sigma,\tau\rangle \models \varphi_1$ and $end(\sigma) = \infty$ iff, by the induction hypothesis, $\langle[\sigma]_{cset},\tau\rangle \models \varphi_1$ and $end([\sigma]_{cset}) = \infty$ iff $\langle[\sigma]_{cset},\tau\rangle \models \varphi_1\ \mathcal{C}\ \varphi_2$;
  - or there exist models σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $\tau \le end(\sigma_1) < \infty$, $\langle\sigma_1,\tau\rangle \models \varphi_1$, and $\langle\sigma_2, begin(\sigma_2)\rangle \models \varphi_2$ iff, by the induction hypothesis, there exist models σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $\langle[\sigma_1]_{cset},\tau\rangle \models \varphi_1$, and $\langle[\sigma_2]_{cset}, begin(\sigma_2)\rangle \models \varphi_2$ iff, there exist models $[\sigma_1]_{cset}$ and $[\sigma_2]_{cset}$ such that $[\sigma]_{cset} = [\sigma_1]_{cset}[\sigma_2]_{cset}$, $\langle[\sigma_1]_{cset},\tau\rangle \models \varphi_1$, and $\langle[\sigma_2]_{cset}, begin([\sigma_2]_{cset})\rangle \models \varphi_2$ iff $\langle[\sigma]_{cset},\tau\rangle \models \varphi_1\ \mathcal{C}\ \varphi_2$.

- $\varphi \equiv \varphi_1\ \mathcal{C}^*\ \varphi_2$. Similar to the proof for $\varphi \equiv \varphi_1\ \mathcal{C}\ \varphi_2$.

Proof of Lemma 2.6.8

Consider any $vset \subseteq VAR$ and any specification φ. We prove, by induction on φ, that if $var(\varphi) \subseteq vset$ then, for any model σ and any $\tau \ge begin(\sigma)$, $\langle\sigma,\tau\rangle \models \varphi$ iff $\langle\sigma \downarrow vset,\tau\rangle \models \varphi$.

- $\varphi \equiv texp_1 = texp_2$. For $i = 1,2$, $var(texp_i) \subseteq var(\varphi) \subseteq vset$. Hence $\langle\sigma,\tau\rangle \models texp_1 = texp_2$ iff $\mathcal{T}(texp_1)(\sigma,\tau) = \mathcal{T}(texp_2)(\sigma,\tau)$ iff, by lemma 2.6.6, $\mathcal{T}(texp_1)(\sigma \downarrow vset,\tau) = \mathcal{T}(texp_2)(\sigma \downarrow vset,\tau)$ iff $\langle\sigma \downarrow vset,\tau\rangle \models texp_1 = texp_2$.

- $\varphi \equiv texp_1 < texp_2$. Similar to the proof for $\varphi \equiv texp_1 = texp_2$.

- $\varphi \equiv comm(c, vexp)$. $var(vexp) = var(\varphi)$ and thus $var(vexp) \subseteq vset$. Hence $\langle\sigma,\tau\rangle \models comm(c, vexp)$ iff $\tau < end(\sigma)$ and $(c, \mathcal{V}(vexp)(\sigma,\tau)) \in \sigma(\tau).c$ iff, by definition and lemma 2.6.4, $\tau < end(\sigma \downarrow vset)$ and $(c, \mathcal{V}(vexp)(\sigma \downarrow vset, \tau)) \in (\sigma \downarrow vset)(\tau).c$ iff $\langle \sigma \downarrow vset, \tau\rangle \models comm(c, vexp)$.

- $\varphi \equiv comm(c)$. $\langle \sigma, \tau\rangle \models comm(c)$ iff $\tau < end(\sigma)$ and there exists a value ϑ such that $(c, \vartheta) \in \sigma(\tau).c$ iff $\tau < end(\sigma \downarrow vset)$ and there exists a value ϑ such that $(c, \vartheta) \in (\sigma \downarrow vset)(\tau).c$ iff $\langle \sigma \downarrow vset, \tau\rangle \models comm(c)$.

- $\varphi \equiv wait(c!)$. $\langle \sigma, \tau\rangle \models wait(c!)$ iff $\tau < end(\sigma)$ and $c! \in \sigma(\tau).c$ iff $\tau < end(\sigma \downarrow vset)$ and $c! \in (\sigma \downarrow vset)(\tau).c$ iff $\langle \sigma \downarrow vset, \tau\rangle \models wait(c!)$.

- $\varphi \equiv wait(c?)$. $\langle \sigma, \tau\rangle \models wait(c?)$ iff $\tau < end(\sigma)$ and $c? \in \sigma(\tau).c$ iff $\tau < end(\sigma \downarrow vset)$ and $c? \in (\sigma \downarrow vset)(\tau).c$ iff $\langle \sigma \downarrow vset, \tau\rangle \models wait(c?)$.

- $\varphi \equiv \varphi_1 \vee \varphi_2$. For $i = 1, 2$, $var(\varphi_i) \subseteq var(\varphi) \subseteq vset$. Hence $\langle \sigma, \tau\rangle \models \varphi_1 \vee \varphi_2$ iff $\langle \sigma, \tau\rangle \models \varphi_1$ or $\langle \sigma, \tau\rangle \models \varphi_2$ iff, by the induction hypothesis, $\langle \sigma \downarrow vset, \tau\rangle \models \varphi_1$ or $\langle \sigma \downarrow vset, \tau\rangle \models \varphi_2$ iff $\langle \sigma \downarrow vset, \tau\rangle \models \varphi_1 \vee \varphi_2$.

- $\varphi \equiv \neg\varphi_1$ and $\varphi \equiv \varphi_1 \,\mathcal{U}\, \varphi_2$. Similar to the proof for $\varphi \equiv \varphi_1 \vee \varphi_2$.

- $\varphi \equiv \varphi_1 \;\mathcal{C}\; \varphi_2$. For $i = 1, 2$, $var(\varphi_i) \subseteq var(\varphi) \subseteq vset$. Hence $\langle \sigma, \tau\rangle \models \varphi_1 \;\mathcal{C}\; \varphi_2$ iff
  - either $\langle \sigma, \tau\rangle \models \varphi_1$ and $end(\sigma) = \infty$ iff, by the induction hypothesis, $\langle \sigma \downarrow vset, \tau\rangle \models \varphi_1$ and $end(\sigma \downarrow vset) = \infty$ iff $\langle \sigma \downarrow vset, \tau\rangle \models \varphi_1 \;\mathcal{C}\; \varphi_2$;
  - or there exist models σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $\tau \le end(\sigma) < \infty$, $\langle \sigma_1, \tau\rangle \models \varphi_1$, and $\langle \sigma_2, begin(\sigma_2)\rangle \models \varphi_2$ iff, by the induction hypothesis, there exist models σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $\langle \sigma_1 \downarrow vset, \tau\rangle \models \varphi_1$, and $\langle \sigma_2 \downarrow vset, begin(\sigma_2)\rangle \models \varphi_2$ iff, there exist models $\sigma_1 \downarrow vset$ and $\sigma_2 \downarrow vset$ such that $\sigma \downarrow vset = (\sigma_1 \downarrow vset)(\sigma_2 \downarrow vset)$, $\langle \sigma_1 \downarrow vset, \tau\rangle \models \varphi_1$, and $\langle \sigma_2 \downarrow vset, begin(\sigma_2 \downarrow vset)\rangle \models \varphi_2$ iff $\langle \sigma \downarrow vset, \tau\rangle \models \varphi_1 \;\mathcal{C}\; \varphi_2$.

- $\varphi \equiv \varphi_1 \;\mathcal{C}^*\; \varphi_2$. Similar to the proof for $\varphi \equiv \varphi_1 \,\mathcal{C}\, \varphi_2$.

Proof of Lemma 2.6.9

Consider any model σ and $cset \subseteq DCHAN$. We prove that $dch(\sigma) \subseteq cset$ iff $\sigma = [\sigma]_{cset}$.

By the definition of projection onto variables, $begin(\sigma) = begin([\sigma]_{cset})$, $end(\sigma) = end([\sigma]_{cset})$, and for any τ_1, $begin(\sigma) \le \tau_1 \le end(\sigma)$, $\sigma(\tau_1).s = [\sigma]_{cset}(\tau_1).s$. Then we only have to prove that, for any τ, $begin(\sigma) \le \tau < end(\sigma)$, $dch(\sigma) \subseteq cset$ iff $\sigma(\tau).c = [\sigma]_{cset}(\tau).c$.

Let $c \in CHAN$ and $\vartheta \in VAL$. By definition, for any τ, $begin(\sigma) \leq \tau < end(\sigma)$,

$$\begin{array}{rcl}[\sigma]_{cset}(\tau).c & = & \{c! \mid c! \in \sigma(\tau).c \wedge c! \in cset\} \cup \{c? \mid c? \in \sigma(\tau).c \wedge c? \in cset\} \cup \\ & & \{(c,\vartheta) \mid (c,\vartheta) \in \sigma(\tau).c \wedge c \in cset\}\end{array}$$

and

$$\begin{array}{rl}dch(\sigma) = \bigcup_{begin(\sigma)\leq\tau<end(\sigma)} & \{c! \mid c! \in \sigma(\tau).c\} \cup \{c? \mid c? \in \sigma(\tau).c\} \cup \\ & \{c \mid \text{there exists a } \vartheta \text{ such that } (c,\vartheta) \in \sigma(\tau).c\}\end{array}$$

Assume $dch(\sigma) \subseteq cset$. We show $\sigma(\tau).c = [\sigma]_{cset}(\tau).c$, for any τ, $begin(\sigma) \leq \tau < end(\sigma)$. If $c! \in \sigma(\tau).c$, then $c! \in dch(\sigma)$. By the assumption, $c! \in cset$ and thus $c! \in [\sigma]_{cset}(\tau).c$. Similarly, if $c? \in \sigma(\tau).c$ then $c? \in [\sigma]_{cset}(\tau).c$, and if $(c,\vartheta) \in \sigma(\tau).c$, then $(c,\vartheta) \in [\sigma]_{cset}(\tau).c$. Thus $\sigma(\tau).c \subseteq [\sigma]_{cset}(\tau).c$. On the other hand, if $c! \in [\sigma]_{cset}(\tau).c$, then $c! \in \sigma(\tau).c$. If $c? \in [\sigma]_{cset}(\tau).c$, then $c? \in \sigma(\tau).c$. If $(c,\vartheta) \in [\sigma]_{cset}(\tau).c$, then $(c,\vartheta) \in \sigma(\tau).c$. Therefore $[\sigma]_{cset}(\tau).c \subseteq \sigma(\tau).c$. Hence $\sigma(\tau).c = [\sigma]_{cset}(\tau).c$.
Now assume $\sigma(\tau).c = [\sigma]_{cset}(\tau).c$, for any τ, $begin(\sigma) \leq \tau < end(\sigma)$. We prove $dch(\sigma) \subseteq cset$. Consider any $c! \in dch(\sigma)$. By definition, there exists a τ, $begin(\sigma) \leq \tau < end(\sigma)$, such that $c! \in \sigma(\tau).c$. By the assumption, $c! \in [\sigma]_{cset}(\tau).c$ and then $c! \in cset$. Similarly, if $c? \in dch(\sigma)$, then $c? \in cset$, and if $c \in dch(\sigma)$, then $c \in cset$. Hence $dch(\sigma) \subseteq cset$.
Hence the lemma holds.

Proof of Lemma 2.6.10

Consider a model σ and two sets $cset_1, cset_2 \subseteq DCHAN$. We prove that
if $\langle\sigma, begin(\sigma)\rangle \models \Box\, empty(cset_2 \setminus cset_1)$, then $[\sigma]_{cset_1 \cup cset_2} = [\sigma]_{cset_1}$.

By the definition of projection onto channels, $begin([\sigma]_{cset_1 \cup cset_2}) = begin([\sigma]_{cset_1})$, $end([\sigma]_{cset_1 \cup cset_2}) = end([\sigma]_{cset_1})$, and for any τ, $begin(\sigma) \leq \tau \leq end(\sigma)$, $[\sigma]_{cset_1 \cup cset_2}(\tau).s = \sigma(\tau).s = [\sigma]_{cset_1}(\tau).s$. Then we only have to prove, for any τ, $begin(\sigma) \leq \tau < end(\sigma)$, $[\sigma]_{cset_1 \cup cset_2}(\tau).c = [\sigma]_{cset_1}(\tau).c$.
Since $cset_1 \cup cset_2 = cset_1 \cup (cset_2 \setminus cset_1)$, we obtain $[\sigma]_{cset_1 \cup cset_2} = [\sigma]_{cset_1 \cup (cset_2 \setminus cset_1)}$ and then $[\sigma]_{cset_1 \cup cset_2}(\tau).c = [\sigma]_{cset_1 \cup (cset_2 \setminus cset_1)}(\tau).c = [\sigma]_{cset_1}(\tau).c\ \cup [\sigma]_{(cset_2 \setminus cset_1)}(\tau).c$.
We show $[\sigma]_{(cset_2 \setminus cset_1)}(\tau).c = \emptyset$.
Assume $\langle\sigma, begin(\sigma)\rangle \models \Box\, empty(cset_2 \setminus cset_1)$. For any $c \in cset_2 \setminus cset_1$, by definition, we have $\langle\sigma, begin(\sigma)\rangle \models \Box \neg comm(c)$. Thus, for any τ, $begin(\sigma) \leq \tau < end(\sigma)$, and for any value $\vartheta \in VAL$, $(c,\vartheta) \notin \sigma(\tau).c$. Thus $(c,\vartheta) \notin [\sigma]_{(cset_2 \setminus cset_1)}(\tau).c$. Similarly, for any $c! \in cset_2 \setminus cset_1$, we obtain $c! \notin [\sigma]_{(cset_2 \setminus cset_1)}(\tau).c$, and for any $c? \in cset_2 \setminus cset_1$, $c? \notin [\sigma]_{(cset_2 \setminus cset_1)}(\tau).c$. Hence $[\sigma]_{(cset_2 \setminus cset_1)}(\tau).c = \emptyset$ and then $[\sigma]_{cset_1 \cup cset_2}(\tau).c = [\sigma]_{cset_1}(\tau).c$.
Thus the lemma holds.

Proof of Lemma 2.6.11

Consider a model σ and two sets $vset_1, vset_2 \subseteq VAR$. We prove that if $\langle \sigma, begin(\sigma)\rangle \models \Box\, inv(vset_2 \setminus vset_1)$, then $\sigma \downarrow (vset_1 \cup vset_2) = \sigma \downarrow vset_1$.

By the definition of projection onto variables,
$begin(\sigma \downarrow (vset_1 \cup vset_2)) = begin(\sigma \downarrow vset_1)$,
$end(\sigma \downarrow (vset_1 \cup vset_2)) = end(\sigma \downarrow vset_1)$, and for any τ, $begin(\sigma) \leq \tau \leq end(\sigma)$,
$(\sigma \downarrow (vset_1 \cup vset_2))(\tau).c = (\sigma \downarrow vset_1)(\tau).c$.
We only have to prove $(\sigma \downarrow (vset_1 \cup vset_2))(\tau).s = (\sigma \downarrow vset_1)(\tau).s$.

By definition, we have $(\sigma \downarrow (vset_1 \cup vset_2))(\tau).s(x) = \begin{cases} \sigma(\tau).s(x) & \text{if } x \in vset_1 \cup vset_2 \\ \sigma^b.s(x) & \text{otherwise} \end{cases}$

If $x \in vset_1 \cup vset_2$, since $vset_1 \cup vset_2 = vset_1 \cup (vset_2 \setminus vset_1)$, we have $x \in vset_1$ or $x \in vset_2 \setminus vset_1$. Assume $\langle \sigma, begin(\sigma)\rangle \models \Box\, inv(vset_2 \setminus vset_1)$. Then for any $x \in vset_2 \setminus vset_1$, any τ, $begin(\sigma) \leq \tau \leq end(\sigma)$, we obtain $\sigma(\tau).s(x) = \sigma^b.s(x)$.

Thus, $(\sigma \downarrow (vset_1 \cup vset_2))(\tau).s(x) = \begin{cases} \sigma(\tau).s(x) & \text{if } x \in vset_1 \\ \sigma^b.s(x) & \text{otherwise} \end{cases}$

Hence $(\sigma \downarrow (vset_1 \cup vset_2))(\tau).s = (\sigma \downarrow vset_1)(\tau).s$ and thus this lemma holds.

Proof of Lemma 2.6.12

Consider a model σ. We prove that if $dch(\sigma) \subseteq cset$ and $\langle \sigma, begin(\sigma)\rangle \models WF_{cset}$, then σ is well-formed.

Assume $\langle \sigma, begin(\sigma)\rangle \models WF_{cset}$. Then
$\langle \sigma, begin(\sigma)\rangle \models \Box(MinWait_{cset} \wedge Exclusion_{cset} \wedge Unique_{cset})$. Hence, for any $\tau \geq begin(\sigma)$,

1. $\langle \sigma, \tau\rangle \models \neg(wait(c!) \wedge wait(c?))$, for any $\{c!, c?\} \subseteq cset$;

2. $\langle \sigma, \tau\rangle \models \neg(comm(c) \wedge wait(c!))$, for any $\{c, c!\} \subseteq cset$, and
   $\langle \sigma, \tau\rangle \models \neg(comm(c) \wedge wait(c?))$, for any $\{c, c?\} \subseteq cset$;

3. $\langle \sigma, \tau\rangle \models comm(c, vexp_1) \wedge comm(c, vexp_2) \rightarrow vexp_1 = vexp_2$, for any $c \in cset$.

Given the interpretation of specifications (section 2.3), this implies, for any $\tau \geq begin(\sigma)$,

1. $\neg(c! \in \sigma(\tau).c \wedge c? \in \sigma(\tau).c)$, for any $\{c!, c?\} \subseteq cset$;

2. There does not exist a value $\vartheta \in VAL$ such that
   $(c, \vartheta) \in \sigma(\tau).c \wedge c! \in \sigma(\tau).c$ or $(c, \vartheta) \in \sigma(\tau).c \wedge c? \in \sigma(\tau).c$.
   Thus, for any value $\vartheta \in VAL$,

$\neg((c,\vartheta) \in \sigma(\tau).c \wedge c! \in \sigma(\tau).c)$, for any $\{c, c!\} \subseteq cset$, and
$\neg((c,\vartheta) \in \sigma(\tau).c \wedge c? \in \sigma(\tau).c)$, for any $\{c, c?\} \subseteq cset$;

3. $(c, \mathcal{V}(vexp_1)(\sigma,\tau)) \in \sigma(\tau).c \;\wedge\; (c, \mathcal{V}(vexp_2)(\sigma,\tau)) \in \sigma(\tau).c \rightarrow \mathcal{V}(vexp_1)(\sigma,\tau) = \mathcal{V}(vexp_2)(\sigma,\tau)$, for any $c \in cset$.
Since $vexp_1$ and $vexp_2$ are arbitrary expressions of type VAL, let ϑ_1, $\vartheta_2 \in VAL$ be such that $\vartheta_1 \equiv vexp_1$ and $\vartheta_2 \equiv vexp_2$. Hence $\vartheta_1 = \mathcal{V}(vexp_1)(\sigma,\tau)$ and $\vartheta_2 = \mathcal{V}(vexp_2)(\sigma,\tau)$. Thus, for any $\tau \geq begin(\sigma)$,
$(c,\vartheta_1) \in \sigma(\tau).c \;\wedge\; (c,\vartheta_2) \in \sigma(\tau).c \rightarrow \vartheta_1 = \vartheta_2$, for any $c \in cset$.

Notice that if $c! \notin cset$ then, by $dch(\sigma) \subseteq cset$, we have $c! \notin dch(\sigma)$ and thus $c! \notin \sigma(\tau).c$, for any τ, $begin(\sigma) \leq \tau < end(\sigma)$. Similarly, if $c? \notin cset$ then $c? \notin \sigma(\tau).c$ and if $c \notin cset$ then, for any value $\vartheta \in VAL$, $(c,\vartheta) \notin \sigma(\tau).c$. Thus, for any $c \in CHAN$, for any values $\vartheta, \vartheta_1, \vartheta_2 \in VAL$, and for any τ, $begin(\sigma) \leq \tau < end(\sigma)$, we have:

1. $\neg(c! \in \sigma(\tau).c \wedge c? \in \sigma(\tau).c)$;

2. $\neg((c,\vartheta) \in \sigma(\tau).c \wedge c! \in \sigma(\tau).c)$ and $\neg((c,\vartheta) \in \sigma(\tau).c \wedge c? \in \sigma(\tau).c)$;

3. $(c,\vartheta_1) \in \sigma(\tau).c \wedge (c,\vartheta_2) \in \sigma(\tau).c \rightarrow \vartheta_1 = \vartheta_2$.

Hence σ is well-formed.

Appendix B

Soundness of the Proof System in Chapter 2

To prove the soundness of a proof system, we must show that every axiom in the proof system is indeed valid and every inference rule preserves validity, i.e., if the hypotheses of an inference rule are valid, so is the conclusion.

Well-Formedness

Consider any procee S and any finite set $cset \subseteq DCHAN$. We prove that the well-formedness axiom 2.4.1 is valid.

For any $\sigma \in \mathcal{M}(S)$, by theorem 2.2.1, σ is well-formed, that is, for any τ, $begin(\sigma) \leq \tau < end(\sigma)$, any $c \in CHAN$, and any $\vartheta_1, \vartheta_2, \vartheta \in VAL$, we have:

1. $\neg(c! \in \sigma(\tau).c \wedge c? \in \sigma(\tau).c)$,

2. $\neg((c,\vartheta) \in \sigma(\tau).c \wedge c! \in \sigma(\tau).c) \ \wedge \ \neg((c,\vartheta) \in \sigma(\tau).c \wedge c? \in \sigma(\tau).c)$, and

3. $(c,\vartheta_1) \in \sigma(\tau).c \wedge (c,\vartheta_2) \in \sigma(\tau).c \rightarrow \vartheta_1 = \vartheta_2$.

For any expressions $vexp_1$ and $vexp_2$ of type VAL and any τ, $begin(\sigma) \leq \tau < end(\sigma)$, we have $\mathcal{V}(vexp_1)(\sigma,\tau) \in VAL$ and $\mathcal{V}(vexp_2)(\sigma,\tau) \in VAL$. Since ϑ_1 and ϑ_2 are arbitrary values in VAL, we can replace ϑ_1 and ϑ_2 by $\mathcal{V}(vexp_1)(\sigma,\tau)$ and $\mathcal{V}(vexp_2)(\sigma,\tau)$, respectively. Thus, for any τ, $begin(\sigma) \leq \tau < end(\sigma)$, any $\vartheta \in VAL$, and any expressions $vexp_1$, $vexp_2$, we have:

1. $\neg(c! \in \sigma(\tau).c \wedge c? \in \sigma(\tau).c)$, for any c with $\{c!, c?\} \subseteq cset$,

2. $\neg((c,\vartheta) \in \sigma(\tau).c \wedge c! \in \sigma(\tau).c)$, for any c with $\{c, c!\} \subseteq cset$,
   $\neg((c,\vartheta) \in \sigma(\tau).c \wedge c? \in \sigma(\tau).c)$, for any c with $\{c, c?\} \subseteq cset$, and

3. $(c, \mathcal{V}(vexp_1)(\sigma,\tau)) \in \sigma(\tau).c \ \wedge \ (c, \mathcal{V}(vexp_2)(\sigma,\tau)) \in \sigma(\tau).c \rightarrow$
   $\mathcal{V}(vexp_1)(\sigma,\tau) = \mathcal{V}(vexp_2)(\sigma,\tau)$, for any $c \in cset$.

By the interpretation of specifications, we obtain that, for any τ, $begin(\sigma) \leq \tau < end(\sigma)$, any $\vartheta \in VAL$, and any $vexp_1$ and $vexp_2$:

1. $\langle\sigma,\tau\rangle \models \bigwedge_{\{c!,c?\}\subseteq cset} \neg(wait(c!) \wedge wait(c?))$;

2. $\langle\sigma,\tau\rangle \models \bigwedge_{\{c,c!\}\subseteq cset} \neg(comm(c) \wedge wait(c!)) \wedge \bigwedge_{\{c,c?\}\subseteq cset} \neg(comm(c) \wedge wait(c?))$;

3. $\langle\sigma,\tau\rangle \models \bigwedge_{c\in cset} comm(c, vexp_1) \wedge comm(c, vexp_2) \rightarrow vexp_1 = vexp_2$.

Furthermore, for any $\tau' \geq end(\sigma)$, any $c \in cset$, and any $vexp$, we have
$\langle\sigma,\tau'\rangle \models \neg wait(c!) \wedge \neg wait(c?) \wedge \neg comm(c) \wedge \neg comm(c, vexp)$.
Thus, for any $\tau \geq begin(\sigma)$, and any $vexp_1$ and $vexp_2$, we obtain:

1. $\langle\sigma,\tau\rangle \models \bigwedge_{\{c!,c?\}\subseteq cset} \neg(wait(c!) \wedge wait(c?))$;

2. $\langle\sigma,\tau\rangle \models \bigwedge_{\{c,c!\}\subseteq cset} \neg(comm(c) \wedge wait(c!)) \wedge \bigwedge_{\{c,c?\}\subseteq cset} \neg(comm(c) \wedge wait(c?))$;

3. $\langle\sigma,\tau\rangle \models \bigwedge_{c\in cset} comm(c, vexp_1) \wedge comm(c, vexp_2) \rightarrow vexp_1 = vexp_2$.

Thus, by definition, $\langle\sigma, begin(\sigma)\rangle \models \Box(MinWait_{cset} \wedge Exclusion_{cset} \wedge Unique_{cset})$ and then $\langle\sigma, begin(\sigma)\rangle \models WF_{cset}$. Hence, axiom 2.4.1 is indeed valid.

Communication Invariance

Consider any process S and any set $cset \subseteq DCHAN$ such that $cset \cap dch(S) = \emptyset$. We prove that the communication invariance axiom 2.4.2 is valid.

For any $\sigma \in \mathcal{M}(S)$, by theorem 2.2.1, we obtain $dch(\sigma) \subseteq dch(S)$ and then $cset \cap dch(\sigma) = \emptyset$. Thus, by definition of $dch(\sigma)$, for any τ, $begin(\sigma) \leq \tau < end(\sigma)$, we have:

1. If $c \in cset$, then there does not exist any value ϑ such that $(c, \vartheta) \in \sigma(\tau).c$;

2. If $c! \in cset$, then $c! \notin \sigma(\tau).c$;

3. If $c? \in cset$, then $c? \notin \sigma(\tau).c$.

Thus, for any τ, $begin(\sigma) \leq \tau < end(\sigma)$, we obtain:

1. $\langle\sigma,\tau\rangle \models \neg comm(c)$, for any $c \in cset$;

2. $\langle\sigma,\tau\rangle \models \neg wait(c!)$, for any $c! \in cset$;

3. $\langle\sigma,\tau\rangle \models \neg wait(c?)$, for any $c? \in cset$.

Furthermore, for any $c \in CHAN$ and any $\tau' \geq end(\sigma)$, we have
$\langle\sigma,\tau'\rangle \models \neg comm(c) \wedge \neg wait(c!) \wedge \neg wait(c?)$.
Thus, for any $\tau \geq begin(\sigma)$, we have $\langle\sigma,\tau\rangle \models empty(cset)$ and then
$\langle\sigma, begin(\sigma)\rangle \models \Box\, empty(cset)$.
Hence axiom 2.4.2 is valid.

Variable Invariance

Consider any process S and any $vset \subseteq VAR$ with $vset \cap wvar(S) = \emptyset$. We prove that the variable invariance axiom 2.4.3 is valid.

For any $\sigma \in \mathcal{M}(S)$, any $x \in vset$, and any τ, $begin(\sigma) \leq \tau \leq end(\sigma)$, by theorem 2.2.1, we obtain $\sigma(\tau).s(x) = \sigma^b.s(x)$. Then, by definition, we obtain $\mathcal{V}(x)(\sigma,\tau) = \mathcal{V}(first(x))(\sigma,\tau)$ and thus $\langle\sigma,\tau\rangle \models x = first(x)$. For any $\tau' > end(\sigma)$, by definition, we have $\mathcal{V}(x)(\sigma,\tau') = \sigma^e.s(x) = \sigma^b.s(x) = \mathcal{V}(first(x))(\sigma,\tau')$. Then we obtain $\langle\sigma,\tau'\rangle \models x = first(x)$. Hence, for any $\tau \geq begin(\sigma)$, we have $\langle\sigma,\tau\rangle \models x = first(x)$, i.e., $\langle\sigma, begin(\sigma)\rangle \models \Box\,(x = first(x))$. Since $x \in vset$, we have $\langle\sigma, begin(\sigma)\rangle \models \bigwedge_{x\in vset} \Box\,(x = first(x))$, i.e., $\langle\sigma, begin(\sigma)\rangle \models \Box \bigwedge_{x\in vset}(x = first(x))$. Hence we obtain $\langle\sigma, begin(\sigma)\rangle \models \Box\, inv(vset)$ and thus axiom 2.4.3 is valid.

Conjunction

We prove that the conjunction rule 2.4.1 preserves validity.

Assume that S **sat** φ_1 and S **sat** φ_2 are valid. For any $\sigma \in \mathcal{M}(S)$, we obtain
$\langle\sigma, begin(\sigma)\rangle \models \varphi_1$. Similarly, we have $\langle\sigma, begin(\sigma)\rangle \models \varphi_2$. Hence we obtain
$\langle\sigma, begin(\sigma)\rangle \models \varphi_1 \wedge \varphi_2$, i.e., rule 2.4.1 preserves validity.

Consequence

We prove that the consequence rule 2.4.2 preserves validity.

Assume that S **sat** φ_1 and $\varphi_1 \rightarrow \varphi_2$ are valid. For any $\sigma \in \mathcal{M}(S)$, we obtain $\langle\sigma, begin(\sigma)\rangle \models \varphi_1$. By the implication, we have $\langle\sigma, begin(\sigma)\rangle \models \varphi_2$. Thus rule 2.4.2 preserves validity.

Skip

We prove that the skip axiom 2.4.4 is valid.

Consider any $\sigma \in \mathcal{M}(\textbf{skip})$. We have $begin(\sigma) = end(\sigma)$ and then $\langle \sigma, begin(\sigma) \rangle \models term = start$. Hence axiom 2.4.4 is valid.

Assignment

We prove that the assignment axiom 2.4.5 is valid.

For any $\sigma \in \mathcal{M}(x := e)$, for any τ, $begin(\sigma) \leq \tau < end(\sigma)$, we obtain $\sigma(\tau).s(x) = \sigma^b.s(x)$. By definition, we have $\langle \sigma, \tau \rangle \models x = first(x)$. From the semantics, we have $\sigma^e.s(x) = \mathcal{E}(e)(\sigma^b.s)$. By lemma 2.6.1, we obtain $\mathcal{V}(x)(\sigma, end(\sigma)) = \mathcal{E}(e)(\sigma^b.s) = \mathcal{V}(e)(\sigma, begin(\sigma))$. By definition, we have $\mathcal{V}(e)(\sigma, begin(\sigma)) = \mathcal{V}(e[first(x)/x])(\sigma, begin(\sigma))$ $= \mathcal{V}(e[first(x)/x])(\sigma, end(\sigma))$. Hence $\mathcal{V}(x)(\sigma, end(\sigma)) = \mathcal{V}(e[first(x)/x])(\sigma, end(\sigma))$ and then $\langle \sigma, end(\sigma) \rangle \models x = e[first(x)/x]$. Since $end(\sigma) = begin(\sigma) + K_a$, we obtain $\langle \sigma, end(\sigma) \rangle \models term = start + K_a$ and $\langle \sigma, end(\sigma) \rangle \models T = term$. Thus, we obtain $\langle \sigma, begin(\sigma) \rangle \models (x = first(x)) \; \mathcal{U} \; (T = term = start + K_a \wedge x = e[first(x)/x])$, i.e., axiom 2.4.5 is valid.

Delay

We prove that the delay axiom 2.4.6 is valid.

Consider any $\sigma \in \mathcal{M}(\textbf{delay } e)$. By lemma 2.6.1, $\mathcal{E}(e)(\sigma^b.s) = \mathcal{V}(e)(\sigma, begin(\sigma))$. Since $\sigma \in \mathcal{M}(\textbf{delay } e)$, we have $end(\sigma) = begin(\sigma) + max(0, \mathcal{E}(e)(\sigma^b.s))$. Hence we obtain $end(\sigma) = begin(\sigma) + max(0, \mathcal{V}(e)(\sigma, begin(\sigma)))$ and then $\langle \sigma, begin(\sigma) \rangle \models term = start + max(0, e)$, i.e., axiom 2.4.6 is valid.

Output

We prove that the output axiom 2.4.7 is valid.

Consider any $\sigma \in \mathcal{M}(c!e)$. Then there are two possibilities:

1. either $end(\sigma) = \infty$ and $\sigma \in Wait(c!)$, i.e., for any $\tau \geq begin(\sigma)$, $\sigma(\tau).comm = \{c!\}$;

2. or there exist models σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $\sigma_1 \in Wait(c!)$, $\sigma_2 \in Send(c, e)$, and $end(\sigma_1) < \infty$. That is, there exists a $\tau \in TIME$ such that, $end(\sigma_1) = \tau$, for

any τ_1, $begin(\sigma_1) \le \tau_1 < end(\sigma_1)$, $\sigma_1(\tau_1).s = \sigma_1^b.s$, $\sigma_1(\tau_1).c = \{c!\}$, $\sigma_1^e.s = \sigma_1^b.s$, $end(\sigma_2) = begin(\sigma_2) + K_c$, for any τ_2, $begin(\sigma_2) \le \tau_2 < end(\sigma_2)$, $\sigma_2(\tau_2).c = \{(c, \mathcal{E}(e)(\sigma_2^b.s))\}$, $\sigma_2(\tau_2).s = \sigma_2^b.s$, and $\sigma_2^e.s = \sigma_2^b.s$.

That is,

1. either $end(\sigma) = \infty$ and, for any $\tau \ge begin(\sigma)$, $\langle \sigma, \tau \rangle \models wait(c!)$, i.e., $\langle \sigma, begin(\sigma) \rangle \models \Box\, wait(c!)$;

2. or, from $\sigma = \sigma_1\sigma_2$, we can derive that there exists a $\tau \in TIME$ such that, for any τ_1, $begin(\sigma) \le \tau_1 < \tau$, $\langle \sigma, \tau_1 \rangle \models wait(c!)$. Since $end(\sigma_1) < \infty$, we obtain $begin(\sigma_2) = end(\sigma_1) = \tau$. By lemma 2.6.1, for any τ_2, $\tau \le \tau_2 < end(\sigma)$, $\mathcal{E}(e)(\sigma_2^b.s) = \mathcal{V}(e)(\sigma_2, begin(\sigma_2)) = \mathcal{V}(e)(\sigma_2, \tau_2)$. Thus we have $\langle \sigma, \tau_2 \rangle \models comm(c, e)$. Since $end(\sigma_2) = begin(\sigma_2) + K_c$, we obtain $end(\sigma) = \tau + K_c$ and then $\langle \sigma, \tau \rangle \models T = term - K_c$ as well as $\langle \sigma, end(\sigma) \rangle \models T = term$. Therefore we have $\langle \sigma, begin(\sigma) \rangle \models wait(c!)\ \mathcal{U}\ (T = term - K_c \wedge (comm(c, e)\ \mathcal{U}\ T = term))$.

Hence we obtain $\langle \sigma, begin(\sigma) \rangle \models wait(c!)\ \mathbf{U}\ (T = term - K_c \wedge (comm(c, e)\ \mathcal{U}\ T = term))$, i.e., axiom 2.4.7 is valid.

Input

We prove that the input axiom 2.4.8 is valid.

Consider any $\sigma \in \mathcal{M}(c?x)$. There are two possibilities:

1. either $end(\sigma) = \infty$ and $\sigma \in Wait(c?)$, i.e., for any $\tau \ge begin(\sigma)$, $\sigma(\tau).c = \{c?\}$, and $\sigma(\tau).s = \sigma^b.s$;

2. or there exist models σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $\sigma_1 \in Wait(c?)$, $\sigma_2 \in Receive(c, x)$, and $end(\sigma_1) < \infty$. That is, there exists a $\tau \in TIME$ such that, $end(\sigma_1) = \tau$, for any τ_1, $begin(\sigma_1) \le \tau_1 < end(\sigma_1)$, $\sigma_1(\tau_1).s = \sigma_1^b.s$, $\sigma_1(\tau_1).c = \{c?\}$, $\sigma_1^e.s = \sigma_2^b.s$, $end(\sigma_2) = begin(\sigma_2) + K_c$, there exists a value $\vartheta \in VAL$ such that, for any τ_2, $begin(\sigma_2) \le \tau_2 < end(\sigma_2)$, $\sigma_2(\tau_2).c = \{(c, \vartheta)\}$, $\sigma_2(\tau_2).s = \sigma_2^b.s$, and $\sigma_2^e.s = (\sigma_2^b.s : x \mapsto \vartheta)$.

That is,

1. either $end(\sigma) = \infty$, for any $\tau \ge begin(\sigma)$, $\langle \sigma, \tau \rangle \models wait(c?)$ and $\langle \sigma, \tau \rangle \models x = first(x)$, i.e., $\langle \sigma, begin(\sigma) \rangle \models \Box\,(x = first(x) \wedge wait(c?))$;

2. or, from $\sigma = \sigma_1\sigma_2$, we obtain $begin(\sigma_2) = end(\sigma_1) = \tau$. Thus for any τ_1, $begin(\sigma) \le \tau_1 < \tau$, $\langle \sigma, \tau_1 \rangle \models x = first(x) \wedge wait(c?)$, for any τ_2, $\tau \le \tau_2 <$

$end(\sigma)$, $\langle\sigma,\tau_2\rangle \models x = first(x) \wedge comm(c,\vartheta)$. Since $end(\sigma_2) = begin(\sigma_2) + K_c$, we obtain $end(\sigma) = \tau + K_c$ and then $\langle\sigma,\tau\rangle \models T = term - K_c$ as well as $\langle\sigma, end(\sigma)\rangle \models T = term$. Hence we have $\langle\sigma,\tau\rangle \models T = term - K_c \wedge ((x = first(x) \wedge comm(c,\vartheta))\ \mathcal{U}\ T = term)$. From $\sigma^e.s(x) = \vartheta$, by definition, we obtain that, for any $\tau_2, \tau \leq \tau_2 < end(\sigma)$, $\mathcal{V}(last(x))(\sigma,\tau_2) = \vartheta$. Thus we have $\langle\sigma,\tau\rangle \models (x = first(x) \wedge comm(c, last(x)))\ \mathcal{U}\ T = term$. Therefore we obtain $\langle\sigma, begin(\sigma)\rangle \models (x = first(x) \wedge wait(c?))\ \mathcal{U}\ (T = term - K_c \wedge ((x = first(x) \wedge comm(c, last(x)))\ \mathcal{U}\ T = term))$.

Hence we have $\langle\sigma, begin(\sigma)\rangle \models (x = first(x) \wedge wait(c?))\ \mathbf{U}\ (T = term - K_c \wedge ((x = first(x) \wedge comm(c, last(x)))\ \mathcal{U}\ T = term))$, i.e., axiom 2.4.8 is valid.

Sequential Composition

We prove that the sequential composition rule 2.4.3 preserves validity.

Assume that S_1 **sat** φ_1 and S_2 **sat** φ_2 are valid. We show that $S_1; S_2$ **sat** $\varphi_1\ \mathcal{C}\ \varphi_2$ is also valid. Consider any $\sigma \in \mathcal{M}(S_1; S_2)$. Then there exist $\sigma_1 \in \mathcal{M}(S_1)$ and $\sigma_2 \in \mathcal{M}(S_2)$ such that $\sigma = \sigma_1\sigma_2$. By definition, $end(\sigma_1) \geq begin(\sigma_1)$. From S_1 **sat** φ_1 and S_2 **sat** φ_2, we obtain $\langle\sigma_1, begin(\sigma_1)\rangle \models \varphi_1$ and $\langle\sigma_2, begin(\sigma_2)\rangle \models \varphi_2$. By the definition of the $\mathcal{C}$ operator, we have $\langle\sigma, begin(\sigma_1)\rangle \models \varphi_1\ \mathcal{C}\ \varphi_2$, i.e., $\langle\sigma, begin(\sigma)\rangle \models \varphi_1\ \mathcal{C}\ \varphi_2$. Hence, rule 2.4.3 preserves validity.

Guarded Command with Purely Boolean Guards

Consider $G \equiv [[]_{i=1}^n g_i \rightarrow S_i]$. We prove that the guarded command evaluation axiom 2.4.9 is valid for G.

For any $\sigma \in \mathcal{M}(G)$, there are two possibilities:

1. either $\mathcal{G}(\neg\bar{g})(\sigma^b.s)$ and $\sigma \in \mathcal{M}(\textbf{delay } K_g)$;

2. or there exists a k, $1 \leq k \leq n$, such that $\mathcal{G}(g_k)(\sigma^b.s)$ and $\sigma \in \mathcal{M}(\textbf{delay } K_g; S_k)$.

That is,

1. either, from $\mathcal{G}(\neg\bar{g})(\sigma^b.s)$, by lemma 2.6.2, we obtain $\langle\sigma, begin(\sigma)\rangle \models \neg\bar{g}$. Since $\sigma \in \mathcal{M}(\textbf{delay } K_g)$, we have $end(\sigma) = begin(\sigma) + K_g$ and then $\langle\sigma, begin(\sigma)\rangle \models term = start + K_g$. Recall $Eval \equiv term = start + K_g$. Hence we have $\langle\sigma, begin(\sigma)\rangle \models \neg\bar{g} \rightarrow Eval$.
   From the semantics, for any τ_1, $begin(\sigma) \leq \tau_1 \leq end(\sigma)$, we have $\sigma(\tau_1).s = \sigma^b.s$ and then $\langle\sigma,\tau_1\rangle \models \bigwedge_{x \in wvar(G)} x = first(x)$, i.e., $\langle\sigma,\tau_1\rangle \models inv(wvar(G))$. Also, for

any τ_2, $begin(\sigma) \leq \tau_2 < end(\sigma)$, we have $\sigma(\tau_2).c = \emptyset$, i.e.,
$\langle \sigma, \tau_2 \rangle \models \bigwedge_{c! \in dch(G)} \neg wait(c!) \ \wedge \ \bigwedge_{c? \in dch(G)} \neg wait(c?) \ \wedge \ \bigwedge_{c \in dch(G)} \neg comm(c)$.
Thus we obtain $\langle \sigma, \tau_2 \rangle \models empty(dch(G))$. We also have $\langle \sigma, end(\sigma) \rangle \models T = start + K_g$. Then we have
$\langle \sigma, begin(\sigma) \rangle \models (inv(wvar(G)) \wedge empty(dch(G)))\ \mathcal{U}\ (T = start + K_g \wedge inv(wvar(G)))$.
Therefore we have
$\langle \sigma, begin(\sigma) \rangle \models [(inv(wvar(G)) \wedge empty(dch(G)))\ \mathcal{U}\ (T = start + K_g \wedge inv(wvar(G)))]$
$\wedge (\neg \bar{g} \rightarrow Eval)$;

2. Or, by $\mathcal{G}(g_k)(\sigma^b.s)$, we obtain $\mathcal{G}(\bar{g})(\sigma^b.s)$ and then $\langle \sigma, begin(\sigma) \rangle \models \bar{g}$. Then we have $\langle \sigma, begin(\sigma) \rangle \models \neg \bar{g} \rightarrow Eval$. Since $\sigma \in \mathcal{M}(\mathbf{delay}\ K_g; S_k)$, there exist models $\sigma_1 \in \mathcal{M}(\mathbf{delay}\ K_g)$ and $\sigma_2 \in \mathcal{M}(S_k)$ such that $\sigma = \sigma_1 \sigma_2$. From $\sigma_1 \in \mathcal{M}(\mathbf{delay}\ K_g)$, we obtain the same result as previous case, i.e.,
$\langle \sigma_1, begin(\sigma_1) \rangle \models (inv(wvar(G)) \wedge empty(dch(G)))\ \mathcal{U}\ (T = start + K_g \wedge inv(wvar(G)))$.
Thus we obtain
$\langle \sigma, begin(\sigma) \rangle \models [(inv(wvar(G)) \wedge empty(dch(G)))\ \mathcal{U}\ (T = start + K_g \wedge inv(wvar(G)))]$
$\wedge (\neg \bar{g} \rightarrow Eval)$.

Hence we conclude that axiom 2.4.9 is indeed valid for $G \equiv [[]_{i=1}^{n} g_i \rightarrow S_i]$.

Next we prove that the guarded command with purely boolean guards rule 2.4.4 preserves validity.

Assume S_i **sat** φ_i are valid, $i = 1, 2, \ldots, n$. Consider any $\sigma \in \mathcal{M}(G)$.

1. If $\mathcal{G}(\neg \bar{g})(\sigma^b.s)$ holds, then we have $\langle \sigma, begin(\sigma) \rangle \models \neg \bar{g}$ and then
$\langle \sigma, begin(\sigma) \rangle \models \bar{g} \rightarrow (Eval\ \ \mathcal{C}\ \ \bigvee_{i=1}^{n} g_i \wedge \varphi_i)$.

2. If $\mathcal{G}(g_k)(\sigma^b.s)$ holds, then we obtain $\mathcal{G}(\bar{g})(\sigma^b.s)$ and then $\langle \sigma, begin(\sigma) \rangle \models \bar{g}$.
Since $\sigma \in \mathcal{M}(\mathbf{delay}\ K_g; S_k)$, there exist models $\sigma_1 \in \mathcal{M}(\mathbf{delay}\ K_g)$ and $\sigma_2 \in \mathcal{M}(S_k)$ such that $\sigma = \sigma_1 \sigma_2$. Thus we have $end(\sigma_1) = begin(\sigma_1) + K_g$ and then $\langle \sigma_1, begin(\sigma_1) \rangle \models Eval$. From the assumption, S_i **sat** φ_i are valid, $i = 1, 2, \ldots, n$. Since $\sigma_2 \in \mathcal{M}(S_k)$, we have $\langle \sigma_2, begin(\sigma_2) \rangle \models \varphi_k$. From $\mathcal{G}(g_k)(\sigma^b.s)$, we obtain $\mathcal{G}(g_k)(\sigma_2^b.s)$ and then $\langle \sigma_2, begin(\sigma_2) \rangle \models g_k$. Thus we have $\langle \sigma_2, begin(\sigma_2) \rangle \models g_k \wedge \varphi_k$ and then $\langle \sigma_2, begin(\sigma_2) \rangle \models \bigvee_{i=1}^{n} g_i \wedge \varphi_i$. Since $begin(\sigma_1) \leq end(\sigma_1) < \infty$, by the definition of the $\mathcal{C}$ operator, we obtain $\langle \sigma, begin(\sigma_1) \rangle \models Eval\ \ \mathcal{C}\ \ \bigvee_{i=1}^{n} g_i \wedge \varphi_i$, i.e., $\langle \sigma, begin(\sigma) \rangle \models Eval\ \ \mathcal{C}\ \ \bigvee_{i=1}^{n} g_i \wedge \varphi_i$.
Thus we have $\langle \sigma, begin(\sigma) \rangle \models \bar{g} \rightarrow (Eval\ \ \mathcal{C}\ \ \bigvee_{i=1}^{n} b_i \wedge \varphi_i)$.

Hence rule 2.4.4 preserves validity.

Guarded Command with IO-Guards

Consider $G \equiv [[]_{i=1}^{n} g_i; c_i?x_i \rightarrow S_i \; [] \; g_0; \textbf{delay } e \rightarrow S_0]$. We first prove that the guarded command evaluation axiom 2.4.9 is also valid for G.

Let $\sigma \in \mathcal{M}(G)$. There are four possibilities:

1. $\mathcal{G}(\neg\bar{g})(\sigma^b.s)$ and $\sigma \in \mathcal{M}(\textbf{delay } K_g)$;
2. or $\sigma \in SEQ(\mathcal{M}(\textbf{delay } K_g), FinWait(G), Comm(G))$;
3. or $\sigma \in SEQ(\mathcal{M}(\textbf{delay } K_g), TimeOut(G), \mathcal{M}(S_0))$;
4. or $\sigma \in SEQ(\mathcal{M}(\textbf{delay } K_g), AnyWait(G), Comm(G))$.

Following the proof of axiom 2.4.9 for the case $G \equiv [[]_{i=1}^{n} g_i \rightarrow S_i]$, we conclude that axiom 2.4.9 is also valid for $G \equiv [[]_{i=1}^{n} g_i; c_i?x_i \rightarrow S_i \; [] \; g_0; \textbf{delay } e \rightarrow S_0]$.

Next we prove that the guarded command with io-guards rule 2.4.5 preserves validity.

Assume $c_i?x_i; S_i$ **sat** φ_i, $i = 1, 2, \ldots, n$ and S_0 **sat** φ_0 are valid.

1. If $\mathcal{G}(\neg\bar{g})(\sigma^b.s)$, then we have $\langle\sigma, begin(\sigma)\rangle \models \neg\bar{g}$. Thus we obtain $\langle\sigma, begin(\sigma)\rangle \models \bar{g} \rightarrow (Eval \;\, \mathcal{C} \;\, (Comm \vee TimeOut))$.

2. If $\sigma \in SEQ(\mathcal{M}(\textbf{delay } K_g), FinWait(G), Comm(G))$, then there exist models $\sigma_1 \in \mathcal{M}(\textbf{delay } K_g)$, $\sigma_2 \in FinWait(G)$, and $\sigma_3 \in Comm(G)$ such that $\sigma = \sigma_1\sigma_2\sigma_3$. From $\sigma_1 \in \mathcal{M}(\textbf{delay } K_g)$, we obtain $end(\sigma_1) = begin(\sigma_1) + K_g$ and then $\langle\sigma_1, begin(\sigma_1)\rangle \models term = start + K_g$, i.e., $\langle\sigma_1, begin(\sigma_1)\rangle \models Eval$.
   From $\sigma_2 \in FinWait(G)$, we obtain $end(\sigma_2) < begin(\sigma_2) + max(0, \mathcal{E}(e)(\sigma_2^b.s))$, $\mathcal{G}(g_0)(\sigma_2^b.s)$, for any τ_2, $begin(\sigma_2) \leq \tau_2 < end(\sigma_2)$, $\sigma_2(\tau_2).s = \sigma_2^b.s$, $\sigma_2(\tau_2).c = \{c_i? \mid \mathcal{G}(g_i)(\sigma_2^b.s), 1 \leq i \leq n\}$, and $\sigma_2^e.s = \sigma_2^b.s$. Then for any τ_2', $begin(\sigma_2) \leq \tau_2' \leq end(\sigma_2)$, we have $\langle\sigma_2, \tau_2'\rangle \models inv(wvar(G))$. For any τ_2, $begin(\sigma_2) \leq \tau_2 < end(\sigma_2)$, we obtain $\langle\sigma_2, \tau_2\rangle \models empty(dch(G) \setminus \{c_1?, \ldots, c_n?\})$. By assumption, we have $c_i? \in \sigma_2(\tau_2).c$ iff $\mathcal{G}(g_i)(\sigma_2^b.s)$, for any i, $1 \leq i \leq n$. Then we have $\langle\sigma_2, \tau_2\rangle \models wait(c_i?)$ iff $\langle\sigma_2, begin(\sigma_2)\rangle \models g_i$ iff $\langle\sigma_2, \tau_2\rangle \models g_i$. Thus we obtain $\langle\sigma_2, \tau_2\rangle \models \bigwedge_{i=1}^{n} g_i \leftrightarrow wait(c_i?)$. From $end(\sigma_2) < begin(\sigma_2) + max(0, \mathcal{E}(e)(\sigma_2^b.s))$, we have, for any τ_2', $begin(\sigma_2) \leq \tau_2' \leq end(\sigma_2)$, $\langle\sigma_2, \tau_2'\rangle \models T < start + max(0, e)$. From $\mathcal{G}(g_0)(\sigma_2^b.s)$, we have $\langle\sigma_2, \tau_2'\rangle \models g_0$ and then $\langle\sigma, begin(\sigma)\rangle \models \bar{g}$. Thus $\langle\sigma_2, \tau_2'\rangle \models g_0 \rightarrow T < start + max(0, e)$. It is obvious that $\langle\sigma_2, end(\sigma_2)\rangle \models T = term$ holds. Hence we obtain
   $\langle\sigma_2, begin(\sigma_2)\rangle \models [(inv(wvar(G)) \wedge empty(dch(G) \setminus \{c_1?, \ldots, c_n?\}) \wedge (g_0 \rightarrow T < start + max(0, e)) \wedge \bigwedge_{i=1}^{n}(g_i \leftrightarrow wait(c_i?))] \; \mathcal{U} \; (inv(wvar(G)) \wedge T = term \wedge (g_0 \rightarrow T < start + max(0, e)))$, i.e., $\langle\sigma_2, begin(\sigma_2)\rangle \models Wait \; \mathcal{U} \; InTime$.

From $\sigma_3 \in Comm(G)$, there exists a k, $1 \leq k \leq n$, such that $\mathcal{G}(g_k)(\sigma_3^b.s)$ and $\sigma_3 \in SEQ(Receive(c_k, x_k), \mathcal{M}(S_k))$. Then $\langle \sigma_3, begin(\sigma_3)\rangle \models g_k$, $\sigma_3 \in \mathcal{M}(c_k?x_k; S_k)$, and $\langle \sigma_3, begin(\sigma_3)\rangle \models comm(c_k)$. By assumption, $c_k?x_k; S_k$ **sat** φ_k is valid. Thus we have $\langle \sigma_3, begin(\sigma_3)\rangle \models \varphi_k$ and then $\langle \sigma_3, begin(\sigma_3)\rangle \models g_k \wedge \varphi_k \wedge comm(c_k)$. Hence we obtain $\langle \sigma_3, begin(\sigma_3)\rangle \models \bigvee_{i=1}^n g_i \wedge \varphi_i \wedge comm(c_i)$.
Then we have $\langle \sigma_2\sigma_3, begin(\sigma_2)\rangle \models (Wait\ \mathcal{U}\ InTime)\ \mathcal{C}\ \ \bigvee_{i=1}^n g_i \wedge \varphi_i \wedge comm(c_i)$, i.e., $\langle \sigma_2\sigma_3, begin(\sigma_2)\rangle \models Comm$.
By $\sigma = \sigma_1\sigma_2\sigma_3$, we obtain $\langle \sigma, begin(\sigma)\rangle \models Eval\ \mathcal{C}\ \ Comm$.
Hence we have $\langle \sigma, begin(\sigma)\rangle \models \bar{g} \rightarrow (Eval\ \mathcal{C}\ \ Comm)$;

3. If $\sigma \in SEQ(\mathcal{M}(\textbf{delay}\ K_g), TimeOut(G), \mathcal{M}(S_0))$, there exist models $\sigma_1 \in \mathcal{M}(\textbf{delay}\ K_g)$, $\sigma_2 \in TimeOut(G)$, and $\sigma_3 \in \mathcal{M}(S_0)$ such that $\sigma = \sigma_1\sigma_2\sigma_3$.
$\sigma_1 \in \mathcal{M}(\textbf{delay}\ K_g)$ implies $\langle \sigma_1, begin(\sigma_1)\rangle \models Eval$.
$\sigma_2 \in TimeOut(G)$ implies $\mathcal{G}(g_0)(\sigma_2^b.s)$ and $end(\sigma_2) = begin(\sigma_2) + max(0, \mathcal{E}(e)(\sigma_2^b.s))$. Thus we have $\langle \sigma_2, begin(\sigma_2)\rangle \models g_0$ and then $\langle \sigma, begin(\sigma)\rangle \models \bar{g}$. By lemma 2.6.1, we have $end(\sigma_2) = begin(\sigma_2) + max(0, \mathcal{E}(e)(\sigma_2^b.s)) = begin(\sigma_2) + max(0, \mathcal{V}(e)(\sigma_2, end(\sigma_2)))$ and then $\langle \sigma_2, end(\sigma_2)\rangle \models T = term = start + max(0, e)$. Similar to previous case, we can also derive that, for any τ_2, $begin(\sigma_2) \leq \tau_2 < end(\sigma_2)$, $\langle \sigma_2, \tau_2\rangle \models empty(dch(G) \setminus \{c_1?, \ldots, c_n?\}) \wedge (g_0 \rightarrow T < start + max(0, e)) \wedge \bigwedge_{i=1}^n (g_i \leftrightarrow wait(c_i?))$, and for any τ_2', $begin(\sigma_2) \leq \tau_2' \leq end(\sigma_2)$, $\langle \sigma_2, \tau_2'\rangle \models inv(wvar(G)) \wedge g_0$. Hence, we obtain $\langle \sigma_2, begin(\sigma_2)\rangle \models Wait\ \mathcal{U}\ EndTime$.
Since S_0 **sat** φ_0 is valid, we have $\langle \sigma_3, begin(\sigma_3)\rangle \models \varphi_0$.
Thus we obtain $\langle \sigma_2\sigma_3, begin(\sigma_2)\rangle \models (Wait\ \mathcal{U}\ EndTime)\ \mathcal{C}\ \ \varphi_0$, i.e., $\langle \sigma_2\sigma_3, begin(\sigma_2)\rangle \models TimeOut$.
By $\sigma = \sigma_1\sigma_2\sigma_3$, we have $\langle \sigma, begin(\sigma)\rangle \models Eval\ \mathcal{C}\ \ TimeOut$.
Hence we obtain $\langle \sigma, begin(\sigma)\rangle \models \bar{g} \rightarrow (Eval\ \mathcal{C}\ \ TimeOut)$;

4. If $\sigma \in SEQ(\mathcal{M}(\textbf{delay}\ K_g), AnyWait(G), Comm(G))$, then there exist models $\sigma_1 \in \mathcal{M}(\textbf{delay}\ K_g)$, $\sigma_2 \in AnyWait(G)$, and $\sigma_3 \in Comm(G)$ such that $\sigma = \sigma_1\sigma_2\sigma_3$.
$\sigma_1 \in \mathcal{M}(\textbf{delay}\ K_g)$ implies $\langle \sigma_1, begin(\sigma_1)\rangle \models Eval$.
$\sigma_2 \in AnyWait(G)$ implies $\mathcal{G}(\neg g_0)(\sigma_2^b.s)$ and then we have $\langle \sigma_2, begin(\sigma_2)\rangle \models \neg g_0$. Thus we have $\langle \sigma_2, begin(\sigma_2)\rangle \models g_0 \rightarrow T < start + max(0, e)$. From the semantics, we obtain $\mathcal{G}(\bar{g})(\sigma_2^b.s)$ and then $\langle \sigma_2, begin(\sigma_2)\rangle \models \bar{g}$, i.e., $\langle \sigma, begin(\sigma)\rangle \models \bar{g}$. Similar to previous cases, we can derive that, for any τ_2, $begin(\sigma_2) \leq \tau_2 < end(\sigma_2)$, $\langle \sigma_2, \tau_2\rangle \models empty(dch(G) \setminus \{c_1?, \ldots, c_n?\}) \wedge \bigwedge_{i=1}^n (g_i \leftrightarrow wait(c_i?))$, for any τ_2', $begin(\sigma_2) \leq \tau_2' \leq end(\sigma_2)$, $\langle \sigma_2, \tau_2'\rangle \models inv(wvar(G)) \wedge (g_0 \rightarrow T < start + max(0, e))$, and $\langle \sigma_2, end(\sigma_2)\rangle \models T = term$. If $end(\sigma_2) = \infty$, we have $\langle \sigma_2, begin(\sigma_2)\rangle \models \Box Wait$. If $end(\sigma_2) < \infty$, we obtain $\langle \sigma_2, begin(\sigma_2)\rangle \models Wait\ \mathcal{U}\ InTime$. Hence we have

$\langle \sigma_2, begin(\sigma_2)\rangle \models Wait\ \mathbf{U}\ InTime$.
$\sigma_3 \in Comm(G)$ implies $\langle \sigma_3, begin(\sigma_3)\rangle \models \bigvee_{i=1}^{n} g_i \wedge \varphi_i \wedge comm(c_i)$.
Thus we obtain $\langle \sigma_2\sigma_3, begin(\sigma_2)\rangle \models (Wait\ \mathbf{U}\ InTime)\ \mathcal{C}\ \bigvee_{i=1}^{n} g_i \wedge \varphi_i \wedge comm(c_i)$,
i.e., $\langle \sigma_2\sigma_3, begin(\sigma_2)\rangle \models Comm$.
By $\sigma = \sigma_1\sigma_2\sigma_3$, we have $\langle \sigma, begin(\sigma)\rangle \models Eval\ \mathcal{C}\ Comm$.
Hence we have $\langle \sigma, begin(\sigma)\rangle \models \bar{g} \rightarrow (Eval\ \mathcal{C}\ Comm)$.

Hence rule 2.4.5 preserves validity.

Iteration

We prove that the iteration rule 2.4.6 preserves validity.

Assume G **sat** φ is valid. We prove that $\star G$ **sat** $(\bar{g} \wedge \varphi)\ \mathcal{C}^*\ (\neg\bar{g} \wedge \varphi)$ is also valid. Consider any $\sigma \in \mathcal{M}(\star G)$. There are two possibilities:

1. either there exist a $k \in I\!N$, $k \geq 1$, and models $\sigma_1, \sigma_2, \ldots, \sigma_k$ such that $\sigma = \sigma_1\sigma_2 \ldots \sigma_k$, for all i, $1 \leq i \leq k$, $\sigma_i \in \mathcal{M}(G)$, for all j, $1 \leq j \leq k-1$, $end(\sigma_j) < \infty$, $\mathcal{G}(\bar{g})(\sigma_j^b.s)$, and if $end(\sigma_k) < \infty$ then $\mathcal{G}(\neg\bar{g})(\sigma_k^b.s)$ otherwise $\mathcal{G}(\bar{g})(\sigma_k^b.s)$,

2. or there exist an infinite sequence of models $\sigma_1, \sigma_2, \ldots$ such that $\sigma = \sigma_1\sigma_2\ldots$, for all $i \geq 1$, $\sigma_i \in \mathcal{M}(G)$, $end(\sigma_i) < \infty$, and $\mathcal{G}(\bar{g})(\sigma_i^b.s)$.

Since G **sat** φ is valid, we obtain $\langle \sigma_i, begin(\sigma_i)\rangle \models \varphi$, for all $\sigma_i \in \mathcal{M}(G)$. Then,

1. either there exist a $k \in I\!N$, $k \geq 1$, and models $\sigma_1, \sigma_2, \ldots, \sigma_k$ such that $\sigma = \sigma_1\sigma_2\ldots\sigma_k$, for all j, $1 \leq j \leq k-1$, $\langle \sigma_j, begin(\sigma_j)\rangle \models \varphi$, $end(\sigma_j) < \infty$. From $\mathcal{G}(\bar{g})(\sigma_j^b.s)$, by lemma 2.6.2, $\langle \sigma_j, begin(\sigma_j)\rangle \models \bar{g}$. Then $\langle \sigma_j, begin(\sigma_j)\rangle \models \bar{g} \wedge \varphi$. If $end(\sigma_k) = \infty$, from $\mathcal{G}(\bar{g})(\sigma_k^b.s)$, we obtain $\langle \sigma_k, begin(\sigma_k)\rangle \models \bar{g}$. By $\langle \sigma_k, begin(\sigma_k)\rangle \models \varphi$, we obtain $\langle \sigma_k, begin(\sigma_k)\rangle \models \bar{g} \wedge \varphi$. If $end(\sigma_k) < \infty$, by $\mathcal{G}(\neg\bar{g})(\sigma_k^b.s)$, we have $\langle \sigma_k, begin(\sigma_k)\rangle \models \neg\bar{g} \wedge \varphi$;

2. Or there exist an infinite sequence of models $\sigma_1, \sigma_2, \ldots$ such that $\sigma = \sigma_1\sigma_2\ldots$, for all $i \geq 1$, $\langle \sigma_i, begin(\sigma_i)\rangle \models \varphi$, $end(\sigma_i) < \infty$, and $\langle \sigma_i, begin(\sigma_i)\rangle \models \bar{g}$. Thus, for all $i \geq 1$, we obtain $\langle \sigma_i, begin(\sigma_i)\rangle \models \bar{g} \wedge \varphi$.

By the definition of the $\mathcal{C}^*$ operator, we obtain $\langle \sigma, begin(\sigma)\rangle \models (\bar{g} \wedge \varphi)\ \mathcal{C}^*\ (\neg\bar{g} \wedge \varphi)$, i.e., rule 2.4.6 preserves validity.

Parallel Composition

We prove that the general parallel composition rule 2.4.8 preserves validity. Then the simple parallel composition rule 2.4.7 preserves validity as well.

Assume S_i **sat** φ_i, $\psi_i \equiv \Box[inv(var(S_i)) \wedge empty(dch(S_i))]$, $dch(\varphi_i) \subseteq dch(S_i)$, and $var(\varphi_i) \subseteq var(S_i)$, for $i = 1, 2$. We show the validity of $S_1 \| S_2$ **sat** $(\varphi_1 \wedge (\varphi_2 \, \mathcal{C} \, \psi_2)) \vee (\varphi_2 \wedge (\varphi_1 \, \mathcal{C} \, \psi_1))$. Consider any $\sigma \in \mathcal{M}(S_1 \| S_2)$. Then $dch(\sigma) \subseteq dch(S_1) \cup dch(S_2)$, and for $i \in \{1, 2\}$, there exist $\sigma_i \in \mathcal{M}(S_i)$ such that $begin(\sigma) = begin(\sigma_1) = begin(\sigma_2)$, $end(\sigma) = max(end(\sigma_1), end(\sigma_2))$. Suppose $end(\sigma_1) \geq end(\sigma_2)$. Then $end(\sigma) = end(\sigma_1)$. We prove $\langle \sigma, begin(\sigma) \rangle \models \varphi_1 \wedge (\varphi_2 \, \mathcal{C} \, \psi_2)$.

- First we prove $\langle \sigma, begin(\sigma) \rangle \models \varphi_1$. From the semantics, we have that, for any τ, $begin(\sigma_1) \leq \tau < end(\sigma_1)$, $[\sigma \downarrow var(S_1)]_{dch(S_1)}(\tau).c = \sigma_1(\tau).c$, for any τ', $begin(\sigma_1) \leq \tau' \leq end(\sigma_1)$, $[\sigma \downarrow var(S_1)]_{dch(S_1)}(\tau').s = \sigma_1(\tau').s$. Since $begin([\sigma \downarrow var(S_1)]_{dch(S_1)}) = begin(\sigma) = begin(\sigma_1)$, $end([\sigma \downarrow var(S_1)]_{dch(S_1)}) = end(\sigma) = end(\sigma_1)$, we obtain $[\sigma \downarrow var(S_1)]_{dch(S_1)} = \sigma_1$. Since $\sigma_1 \in \mathcal{M}(S_1)$ and S_1 **sat** φ_1, we have $\langle [\sigma \downarrow var(S_1)]_{dch(S_1)}, begin(\sigma) \rangle \models \varphi_1$. Since $dch(\varphi_1) \subseteq dch(S_1)$ and $var(\varphi_1) \subseteq var(S_1)$, lemma 2.6.7 and lemma 2.6.8 lead to $\langle \sigma, begin(\sigma) \rangle \models \varphi_1$.

- Next we prove $\langle \sigma, begin(\sigma) \rangle \models \varphi_2 \, \mathcal{C} \, \psi_2$.

  - If $end(\sigma_2) = \infty$, since $end(\sigma) = end(\sigma_1) \geq end(\sigma_2)$, we have $end(\sigma_2) = end(\sigma) = \infty$. Similarly, we can derive $\langle \sigma, begin(\sigma) \rangle \models \varphi_2$. By the definition of the $\mathcal{C}$ operator, we obtain $\langle \sigma, begin(\sigma) \rangle \models \varphi_2 \, \mathcal{C} \, \psi_2$;

  - If $end(\sigma_2) < \infty$, from S_2 **sat** φ_2 and $\sigma_2 \in \mathcal{M}(S_2)$, we obtain $\langle \sigma_2, begin(\sigma_2) \rangle \models \varphi_2$. We define a model σ_3 such that $begin(\sigma_3) = end(\sigma_2)$, $end(\sigma_3) = end(\sigma)$, for any τ, $begin(\sigma_3) \leq \tau < end(\sigma_3)$, $\sigma_3(\tau).c = [\sigma]_{dch(S_2)}(\tau).c$, for any τ', $begin(\sigma_3) \leq \tau' \leq end(\sigma_3)$, $\sigma_3(\tau).s = \sigma_2^e.s$. Then we have $\langle \sigma_3, \tau' \rangle \models inv(var(S_2))$. For any $\tau_1' > end(\sigma_3)$, we also have $\langle \sigma_3, \tau_1' \rangle \models inv(var(S_2))$. Hence we obtain $\langle \sigma_3, begin(\sigma_3) \rangle \models \Box \, inv(var(S_2))$. From the semantics, for any τ, $end(\sigma_2) \leq \tau < end(\sigma)$, $[\sigma]_{dch(S_2)}(\tau).c = \emptyset$. That is, for any τ, $begin(\sigma_3) \leq \tau < end(\sigma_3)$, $\sigma_3(\tau).c = \emptyset$. Thus we have $\langle \sigma_3, \tau \rangle \models empty(dch(S_2))$. For any $\tau_1 > end(\sigma_3)$, we also have $\langle \sigma_3, \tau_1 \rangle \models empty(dch(S_2))$. Then we obtain $\langle \sigma_3, begin(\sigma_3) \rangle \models \Box \, empty(dch(S_2))$. Thus we have $\langle \sigma_3, begin(\sigma_3) \rangle \models \Box[inv(var(S_2)) \wedge empty(dch(S_2))]$, i.e., $\langle \sigma_3, begin(\sigma_3) \rangle \models \psi_2$. By the definition of the $\mathcal{C}$ operator, we obtain $\langle \sigma_2 \sigma_3, begin(\sigma_2) \rangle \models \varphi_2 \mathcal{C} \psi_2$. Next we prove $[\sigma \downarrow var(S_2)]_{dch(S_2)} = \sigma_2 \sigma_3$. Let $\bar{\sigma} \equiv [\sigma \downarrow var(S_2)]_{dch(S_2)}$. By definitions, we have

$$\bar{\sigma}(\tau).s = (\sigma \downarrow var(S_2))(\tau).s = \begin{cases} \sigma_2(\tau).s & begin(\sigma_2) \leq \tau \leq end(\sigma_2) \\ \sigma_3(\tau).s & end(\sigma_2) < \tau \leq end(\sigma) \end{cases}$$

$$\bar{\sigma}(\tau).c = [\sigma]_{dch(S_2)}(\tau).c = \begin{cases} \sigma_2(\tau).c & begin(\sigma_2) \leq \tau < end(\sigma_2) \\ \sigma_3(\tau).c & end(\sigma_2) \leq \tau < end(\sigma) \end{cases}$$

Hence $\bar{\sigma} = \sigma_2\sigma_3$. Thus $\langle[\sigma \downarrow var(S_2)]_{dch(S_2)}, begin(\sigma_2)\rangle \models \varphi_2 \, \mathcal{C} \, \psi_2$. Since $dch(\varphi_2) \subseteq dch(S_2)$ and $var(\varphi_2) \subseteq var(S_2)$, we have $dch(\varphi_2 \, \mathcal{C} \, \psi_2) \subseteq dch(S_2)$ and $var(\varphi_2 \, \mathcal{C} \, \psi_2) \subseteq var(S_2)$. Then lemma 2.6.7 and lemma 2.6.8 lead to $\langle \sigma, begin(\sigma)\rangle \models \varphi_2 \, \mathcal{C} \, \psi_2$.

Therefore we have proved $\langle \sigma, begin(\sigma)\rangle \models \varphi_1 \wedge (\varphi_2 \, \mathcal{C} \, \psi_2)$.
Similarly, for $end(\sigma_1) < end(\sigma_2)$, we can show $\langle \sigma, begin(\sigma)\rangle \models \varphi_2 \wedge (\varphi_1 \, \mathcal{C} \, \psi_1)$.
Hence the general parallel composition rule 2.4.8 preserves validity.

Appendix C

Preciseness of the Proof System in Chapter 2

To prove the preciseness theorem 2.6.2, we show that for any statement S we can prove S **sat** φ where φ is precise for S, namely,

1. S **sat** φ holds, i.e., $\langle \sigma, begin(\sigma)\rangle \models \varphi$, for any $\sigma \in \mathcal{M}(S)$;

2. If σ is a well-formed model, $dch(\sigma) \subseteq dch(S)$, for any variable $x \notin wvar(S)$, x is invariant with respect to σ, and $\langle \sigma, begin(\sigma)\rangle \models \varphi$, then $\sigma \in \mathcal{M}(S)$; and

3. $dch(\varphi) = dch(S)$ and $var(\varphi) = var(S)$.

By induction on the structure of S, we show that, for any statement S, S **sat** φ holds where φ is precise for S.

For all the cases, the proof of the first requirement follows from the soundness theorem (Theorem 2.6.1) and the proof of the third requirement is easy. Hence we only give here the proof of the second requirement.

Skip

By the skip axiom, **skip** **sat** $term = start$. We show that $term = start$ is precise for statement **skip**. Consider a well-formed model σ such that $\langle \sigma, begin(\sigma)\rangle \models term = start$. Then we have $end(\sigma) = begin(\sigma)$ and hence $\sigma \in \mathcal{M}(\textbf{skip})$. Hence $term = start$ is precise for **skip**.

Assignment

Let $\varphi \equiv (x = first(x))\ \mathcal{U}\ (T = term = start + K_a \wedge x = e[first(x)/x])$. By the assignment axiom, $x := e$ **sat** φ. We show that φ is a precise specification for $x := e$.

Consider a well-formed model σ such that $dch(\sigma) \subseteq dch(x := e)$ and any variable $y \notin wvar(x := e)$ is invariant with respect to σ. Thus we obtain $dch(\sigma) = \emptyset$, i.e., for any τ_1, $begin(\sigma) \leq \tau_1 < end(\sigma)$, $\sigma(\tau_1).c = \emptyset$. Furthermore, for any variable $y \not\equiv x$, for any τ_2, $begin(\sigma) \leq \tau_2 \leq end(\sigma)$, we have $\sigma(\tau_2).s(y) = \sigma^b.s(y)$. Assume $\langle \sigma, begin(\sigma)\rangle \models \varphi$. Then we obtain $end(\sigma) = begin(\sigma) + K_a$ and, for any τ_1, $begin(\sigma) \leq \tau_1 < end(\sigma)$, $\sigma(\tau_1).s(x) = \sigma^b.s(x)$, and $\sigma^e.s(x) = \mathcal{V}(e[first(x)/x])(\sigma, end(\sigma))$. By definition, we have $\mathcal{V}(e[first(x)/x])(\sigma, end(\sigma)) = \mathcal{V}(e[first(x)/x])(\sigma, begin(\sigma)) = \mathcal{V}(e)(\sigma, begin(\sigma)) = \mathcal{E}(e)\sigma^b.s$. Thus, for any τ_1, $begin(\sigma) \leq \tau_1 < end(\sigma)$, $\sigma(\tau_1).s = \sigma^b.s$, $\sigma^e.s = (\sigma^b.s : x \mapsto \mathcal{E}(e)\sigma^b.s)$. Hence $\sigma \in \mathcal{M}(x := e)$. Thus φ is a precise specification for $x := e$.

Delay

Let $\varphi \equiv term = start + max(0, e)$. By the delay axiom, **delay** e **sat** φ. We show that φ is a precise specification for **delay** e. Consider a well-formed model σ such that $dch(\sigma) \subseteq dch(\textbf{delay}\ e)$ and any variable $y \notin wvar(\textbf{delay}\ e)$ is invariant with respect to σ. Thus we obtain $dch(\sigma) = \emptyset$, i.e., for any τ_1, $begin(\sigma) \leq \tau_1 < end(\sigma)$, $\sigma(\tau_1).c = \emptyset$. Furthermore, for any τ_2, $begin(\sigma) \leq \tau_2 \leq end(\sigma)$, we have $\sigma(\tau_2).s = \sigma^b.s$. Assume $\langle \sigma, begin(\sigma)\rangle \models \varphi$. Thus $end(\sigma) = begin(\sigma) + max(0, \mathcal{V}(e)(\sigma, begin(\sigma))) = begin(\sigma) + max(0, \mathcal{E}(e)(\sigma^b.s))$. Hence $\sigma \in \mathcal{M}(\textbf{delay}\ e)$. Therefore φ is a precise specification for **delay** e.

Output

Let $\varphi \equiv wait(c!)\ \mathbf{U}\ (T = term - K_c \wedge (comm(c, e)\ \mathcal{U}\ T = term))$. By the output axiom, $c!e$ **sat** φ. We show that φ is precise for $c!e$. Consider a well-formed model σ such that $dch(\sigma) \subseteq dch(c!e)$ and any variable $y \notin wvar(c!e)$ is invariant with respect to σ. Then we obtain $dch(\sigma) \subseteq \{c, c!\}$ and, for any variable y, any τ, $begin(\sigma) \leq \tau \leq end(\sigma)$, $\sigma(\tau).s(y) = \sigma^b.s(y)$. Hence $\sigma(\tau).s = \sigma^b.s$. Assume $\langle \sigma, begin(\sigma)\rangle \models \varphi$. Then there are two possibilities:

- either $\langle \sigma, begin(\sigma)\rangle \models \Box\, wait(c!)$,
- or $\langle \sigma, begin(\sigma)\rangle \models wait(c!)\ \mathcal{U}\ (T = term - K_c \wedge (comm(c, e)\ \mathcal{U}\ T = term))$.

That is,

- either for any $\tau \geq begin(\sigma)$, $\langle \sigma, \tau\rangle \models wait(c!)$, i.e., $\tau < end(\sigma)$ and thus $end(\sigma) = \infty$. By definition, for any $\tau \geq begin(\sigma)$, $c! \in \sigma(\tau).c$. Since σ is a well-formed model, for any value $\vartheta \in VAL$ and any τ, $begin(\sigma) \leq \tau < end(\sigma)$, $\neg(c! \in \sigma(\tau).c \wedge c? \in \sigma(\tau).c$ and $\neg(c! \in \sigma(\tau).c \wedge (c, \vartheta) \in \sigma(\tau).c)$ are valid. Then we obtain $\sigma(\tau).c = \{c!\}$. Together with $\sigma(\tau).s = \sigma^b.s$, we have $\sigma \in \mathcal{M}(c!e)$;

- or there exists a $\tau \geq begin(\sigma)$, $\tau \in TIME$, such that, for any τ_1, $begin(\sigma) \leq \tau_1 < \tau$, $\langle \sigma, \tau_1 \rangle \models wait(c!)$ and $\langle \sigma, \tau \rangle \models T = term - K_c \wedge (comm(c, e)\ \mathcal{U}\ T = term)$. We split σ into two models σ_1 and σ_2 such that $\sigma = \sigma_1 \sigma_2$ with $end(\sigma_1) = \tau$. Thus $begin(\sigma_2) = end(\sigma_1) = \tau$. Then we obtain that, for any τ_1, $begin(\sigma_1) \leq \tau_1 < end(\sigma_1)$, $\sigma_1(\tau_1).c = \{c!\}$. Together with $\sigma(\tau).s = \sigma^b.s$, for any τ, $begin(\sigma) \leq \tau \leq end(\sigma)$, we obtain $\sigma_1 \in Wait(c!)$. From $\langle \sigma, \tau \rangle \models T = term - K_c$, we obtain $\tau = end(\sigma) - K_c$ and then $end(\sigma_2) = \tau + K_c = begin(\sigma_2) + K_c$. From $\langle \sigma, \tau \rangle \models comm(c, e)\ \mathcal{U}\ T = term$, we can derive that, for any τ_2, $begin(\sigma_2) \leq \tau_2 < end(\sigma_2)$, $(c, \mathcal{V}(e)(\sigma_2, \tau_2)) \in \sigma_2(\tau_2).c$. By the well-formedness of σ and the invariance of variables, $\sigma_2(\tau_2).c = \{(c, \mathcal{V}(e)(\sigma_2, begin(\sigma_2)))\} = \{(c, \mathcal{E}(e)\sigma_2^b.s)\}$. Together with $\sigma(\tau).s = \sigma^b.s$, for any τ, $begin(\sigma) \leq \tau \leq end(\sigma)$, we obtain $\sigma_2 \in Send(c, e)$ and hence $\sigma \in \mathcal{M}(c!e)$.

Therefore φ is precise for $c!e$.

Input

Let $\varphi \equiv (x = first(x) \wedge wait(c?))\ \mathbf{U}\ (T = term - K_c \wedge ((x = first(x) \wedge comm(c, last(x)))\ \mathcal{U}\ T = term))$. By the input axiom, $c?x$ **sat** φ. We show that φ is precise for $c?x$. Consider a well-formed model σ such that $dch(\sigma) \subseteq dch(c?x)$ and any variable $y \notin wvar(c?x)$ is invariant with respect to σ. Then $dch(\sigma) \subseteq \{c, c?\}$ and, for any τ, $begin(\sigma) \leq \tau \leq end(\sigma)$, for any variable $y \not\equiv x$, $\sigma(\tau).s(y) = \sigma^b.s(y)$. Assume $\langle \sigma, begin(\sigma) \rangle \models \varphi$. There are two possibilities:

- either $\langle \sigma, begin(\sigma) \rangle \models \Box (x = first(x) \wedge wait(c?))$;
- or $\langle \sigma, begin(\sigma) \rangle \models (x = first(x) \wedge wait(c?))\ \mathcal{U}\ [T = term - K_c \wedge ((x = first(x) \wedge comm(c, last(x)))\ \mathcal{U}\ T = term)]$.

That is,

- either $end(\sigma) = \infty$, for any $\tau \geq begin(\sigma)$, $\sigma(\tau).s(x) = \sigma^b.s(x)$, and $c? \in \sigma(\tau).c$. From the invariance of variables different from x and the well-formedness of σ, we obtain, for any $\tau \geq begin(\sigma)$, $\sigma(\tau).s = \sigma^b.s$ and $\sigma(\tau).c = \{c?\}$. Hence $\sigma \in \mathcal{M}(c?x)$;
- or there exists a $\tau \geq begin(\sigma)$, $\tau \in TIME$, such that, for any τ_1, $begin(\sigma) \leq \tau_1 < \tau$, $\langle \sigma, \tau_1 \rangle \models x = first(x) \wedge wait(c!)$ and $\langle \sigma, \tau \rangle \models T = term - K_c \wedge ((x = first(x) \wedge comm(c, last(x)))\ \mathcal{U}\ T = term)$. We split σ into two models σ_1 and σ_2 such that $\sigma = \sigma_1 \sigma_2$ with $end(\sigma_1) = \tau$. Then $begin(\sigma_2) = end(\sigma_1) = \tau$. We obtain that, for any τ_1, $begin(\sigma_1) \leq \tau_1 < end(\sigma_1)$, $\sigma_1(\tau_1).s = \sigma_1^b.s$, $\sigma_1(\tau_1).c = \{c?\}$. From $\langle \sigma, \tau \rangle \models T = term - K_c$, we have $\tau = end(\sigma) - K_c$ and thus $end(\sigma_2) = begin(\sigma_2) + K_c$.

We can also derive that, for any τ_2, $begin(\sigma_2) \leq \tau_2 < end(\sigma_2)$, $\langle\sigma_2, \tau_2\rangle \models x = first(x) \wedge comm(c, last(x))$. Together with the invariance of variables different from x, we then have $\sigma_2(\tau_2).s = \sigma_2^b.s$. Since $\sigma = \sigma_1\sigma_2$ and $\sigma_1^e.s(x) = \sigma_2^b.s(x)$, we obtain $\sigma_1^e.s = \sigma_1^b.s$. Thus $\sigma_1 \in Wait(c?)$. By definition, $\mathcal{V}(last(x))(\sigma_2, \tau_2) = \sigma_2^e.s(x)$. Let $\vartheta = \sigma_2^e.s(x)$. Hence by the well-formedness of σ, we obtain, for any τ_2, $begin(\sigma_2) \leq \tau_2 < end(\sigma_2)$, $\sigma_2(\tau_2).c = \{(c, \vartheta)\}$. Furthermore, we also have $\sigma_2^e.s = (\sigma_2^b.s : x \mapsto \vartheta)$. Hence $\sigma_2 \in Receive(c, x)$ and then $\sigma \in \mathcal{M}(c?x)$.

Hence φ is precise for $c?x$.

Sequential Composition

Consider $S \equiv S_1; S_2$. By the induction hypothesis, we can derive S_1 **sat** φ_1 and S_2 **sat** φ_2, where φ_1 and φ_2 are precise for S_1 and S_1, respectively. By the communication invariance axiom, we obtain
S_1 **sat** $\Box\, empty(dch(S_2) \setminus dch(S_1))$ and S_2 **sat** $\Box\, empty(dch(S_1) \setminus dch(S_2))$.
By the variable invariance axiom, we obtain
S_1 **sat** $\Box\, inv(wvar(S_1; S_2) \setminus wvar(S_1))$ and S_2 **sat** $\Box\, inv(wvar(S_1; S_2) \setminus wvar(S_2))$.
Then, using the conjunction rule, we have
S_1 **sat** $\varphi_1 \wedge \Box\,(empty(dch(S_2) \setminus dch(S_1)) \wedge inv(wvar(S_1; S_2) \setminus wvar(S_1)))$ and
S_2 **sat** $\varphi_2 \wedge \Box\,(empty(dch(S_1) \setminus dch(S_2)) \wedge inv(wvar(S_1; S_2) \setminus wvar(S_2)))$.
Hence, by the sequential composition rule, $S_1; S_2$ **sat** φ with
$\varphi \equiv [\varphi_1 \wedge \Box\,(empty(dch(S_2) \setminus dch(S_1)) \wedge inv(wvar(S_1; S_2) \setminus wvar(S_1)))]\ C$
$[\varphi_2 \wedge \Box\,(empty(dch(S_1) \setminus dch(S_2)) \wedge inv(wvar(S_1; S_2) \setminus wvar(S_2)))]$.
We prove that φ is precise for $S_1; S_2$.
Consider a well-formed model σ such that $dch(\sigma) \subseteq dch(S_1; S_2)$ and any variable $y \notin wvar(S_1; S_2)$ is invariant with respect to σ. Assume $\langle\sigma, begin(\sigma)\rangle \models \varphi$. There exist σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $end(\sigma_1) > begin(\sigma)$,
$\langle\sigma_1, begin(\sigma_1)\rangle \models \varphi_1 \wedge \Box\,(empty(dch(S_2) \setminus dch(S_1)) \wedge inv(wvar(S_1; S_2) \setminus wvar(S_1)))$, and
$\langle\sigma_2, begin(\sigma_2)\rangle \models \varphi_2 \wedge \Box\,(empty(dch(S_1) \setminus dch(S_2)) \wedge inv(wvar(S_1; S_2) \setminus wvar(S_2)))$.
From $\langle\sigma_1, begin(\sigma_1)\rangle \models \Box\, empty(dch(S_2) \setminus dch(S_1))$, lemma 2.6.10 leads to
$[\sigma]_{dch(S_1) \cup dch(S_2)} = [\sigma]_{dch(S_1)}$. From $dch(\sigma) \subseteq dch(S_1; S_2) = dch(S_1) \cup dch(S_2)$ and $\sigma = \sigma_1\sigma_2$, we obtain $dch(\sigma_1) \subseteq dch(S_1) \cup dch(S_2)$. Thus, by lemma 2.6.9, we have
$\sigma_1 = [\sigma_1]_{dch(S_1) \cup dch(S_2)} = [\sigma_1]_{dch(S_1)}$. By lemma 2.6.9 again, we obtain $dch(\sigma_1) \subseteq dch(S_1)$.
From $\langle\sigma_1, begin(\sigma_1)\rangle \models \Box\, inv(wvar(S_1; S_2) \setminus wvar(S_1))$, we know that any variable $x \in wvar(S_1; S_2) \setminus wvar(S_1)$ is invariant with respect to σ_1. By the assumption, any variable $y \notin wvar(S_1; S_2)$ is invariant with respect to σ. Thus any variable $z \notin wvar(S_1)$ is invariant with respect to σ_1. Since σ is well-formed, both σ_1 and σ_2 are also well-

formed. Together with $\langle\sigma_1, begin(\sigma_1)\rangle \models \varphi_1$ and the preciseness of φ_1 for S_1, we obtain $\sigma_1 \in \mathcal{M}(S_1)$. Similarly, $\sigma_2 \in \mathcal{M}(S_2)$. By $\sigma = \sigma_1\sigma_2$ and the definition of SEQ, $\sigma \in \mathcal{M}(S_1; S_2)$. Then φ is precise for $S_1; S_2$.

Guarded Command with Purely Boolean Guards

Consider $G \equiv [\|_{i=1}^{n} g_i \rightarrow S_i]$. By the induction hypothesis we can derive S_i **sat** φ_i, $i = 1, ...n$, where φ_i is precise for S_i. By the variable invariance axiom,
S_i **sat** $\Box\, inv(wvar(G) \setminus wvar(S_i))$. By the communication invariance axiom,
S_i **sat** $\Box\, empty(dch(G) \setminus dch(S_i))$. Then by the conjunction rule, we have
S_i **sat** $\varphi_i \wedge \Box\, (inv(wvar(G) \setminus wvar(S_i)) \wedge empty(dch(G) \setminus dch(S_i)))$.
By the guarded command evaluation axiom, the guarded command with purely boolean guards rule, and the conjunction rule, we obtain G **sat** φ with

$$\begin{aligned}\varphi \equiv\ & [(inv(wvar(G)) \wedge empty(dch(G)))\ \mathcal{U}\ (T = start + K_g \wedge inv(wvar(G)))] \wedge \\ & (\neg\bar{g} \rightarrow Eval) \wedge [\bar{g} \rightarrow (Eval\ \mathcal{C}\ \ \textstyle\bigvee_{i=1}^{n}(g_i \wedge \varphi_i \wedge \Box\, (inv(wvar(G) \setminus wvar(S_i)) \wedge \\ & empty(dch(G) \setminus dch(S_i))))))]\end{aligned}$$

We prove that φ is precise for G.
Consider a well-formed model σ such that $dch(\sigma) \subseteq dch(G)$ and any variable $y \notin wvar(G)$ is invariant with respect to σ. Assume $\langle\sigma, begin(\sigma)\rangle \models \varphi$. We prove that $\sigma \in \mathcal{M}([\|_{i=1}^{n} g_i \rightarrow S_i)$. By assumption, there exists a $\tau \geq begin(\sigma)$ such that $\langle\sigma, \tau\rangle \models T = start + K_g \wedge inv(wvar(G))$ and, for any τ_1, $begin(\sigma) \leq \tau_1 < \tau$, $\langle\sigma, \tau_1\rangle \models inv(wvar(G)) \wedge empty(dch(G))$. Then we have $\tau = begin(\sigma) + K_g$ and, for any τ_1', $begin(\sigma) \leq \tau_1' \leq \tau$, any $y \in wvar(G)$, $\sigma(\tau_1').s(y) = \sigma^b.s(y)$. Together with the invariance of variables $y \notin wvar(G)$, we obtain $\sigma(\tau_1').s = \sigma^b.s$. Since $dch(\sigma) \subseteq dch(G)$ and $\langle\sigma, \tau_1\rangle \models empty(dch(G))$, we obtain $\sigma(\tau_1).c = \emptyset$.
Next consider the validity of $\bar{g}$. There are two possibilities.

- If $\langle\sigma, begin(\sigma)\rangle \models \neg\bar{g}$, lemma 2.6.2 implies $\mathcal{G}(\neg\bar{g})(\sigma^b.s)$. By assumption, $\langle\sigma, begin(\sigma)\rangle \models term = start + K_g$ and hence $end(\sigma) = begin(\sigma) + K_g$. Thus, $end(\sigma) = \tau = begin(\sigma) + K_g$ and then $\sigma \in \mathcal{M}(\textbf{delay}\ K_g)$.

- If $\langle\sigma, begin(\sigma)\rangle \models \bar{g}$, then $\langle\sigma, begin(\sigma)\rangle \models (term = start + K_g)\ \mathcal{C}$ $\bigvee_{i=1}^{n}(g_i \wedge \varphi_i \wedge \Box\, (inv(wvar(G) \setminus wvar(S_i)) \wedge empty(dch(G) \setminus dch(S_i))))$.
  By definition of the $\mathcal{C}$ operator, there exist models σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $\langle\sigma_1, begin(\sigma_1)\rangle \models term = start + K_g$, and $\langle\sigma_2, begin(\sigma_2)\rangle \models \bigvee_{i=1}^{n}(g_i \wedge \varphi_i \wedge \Box\, (inv(wvar(G) \setminus wvar(S_i)) \wedge empty(dch(G) \setminus dch(S_i))))$. Thus $end(\sigma_1) = begin(\sigma_1) + K_g$. From $begin(\sigma) = begin(\sigma_1)$, we obtain $\sigma_1 \in \mathcal{M}(\textbf{delay}\ K_g)$.
  Since $end(\sigma_1) < \infty$, by the definition of $\sigma_1\sigma_2$, we have $end(\sigma_1) = begin(\sigma_2)$ and $\sigma_1^e.s = \sigma_2^b.s$. Furthermore, there must exist a k, $1 \leq k \leq n$, such that

$\langle\sigma_2, begin(\sigma_2)\rangle \models g_k \wedge \varphi_k \wedge \Box\,(inv(wvar(G)\setminus wvar(S_i)) \wedge empty(dch(G)\setminus dch(S_k)))$. From $\langle\sigma_2, begin(\sigma_2)\rangle \models g_k$, by lemma 2.6.2, $\mathcal{G}(g_k)(\sigma_2^b.s)$. From $\langle\sigma_2, begin(\sigma_2)\rangle \models \Box\, inv(wvar(G) \setminus wvar(S_k))$, any variable $x \in wvar(G) \setminus wvar(S_k)$ is invariant with respect to σ_2. By assumption, any variable $y \notin wvar(G)$ is invariant with respect to σ. Thus, any variable $z \notin wvar(S_k)$ is invariant with respect to σ_2. From $\langle\sigma_2, begin(\sigma_2)\rangle \models \Box\, empty(dch(G) \setminus dch(S_k))$, lemma 2.6.10 leads to $[\sigma_2]_{dch(G)\cup dch(S_k)} = [\sigma_2]_{dch(S_k)}$. Since $dch(G) \cup dch(S_k) = dch(G)$, we obtain $[\sigma_2]_{dch(G)} = [\sigma_2]_{dch(S_k)}$. From $\sigma = \sigma_1\sigma_2$ and $dch(\sigma) \subseteq dch(G)$, we have $dch(\sigma_2) \subseteq dch(G)$. By lemma 2.6.9, it implies $\sigma_2 = [\sigma_2]_{dch(G)}$ and then $\sigma_2 = [\sigma_2]_{dch(S_k)}$. By lemma 2.6.9 again, we obtain $dch(\sigma_2) \subseteq dch(S_k)$. Since σ is a well-formed model, σ_1 and σ_2 are also well-formed. Together with $\langle\sigma_2, begin(\sigma_2)\rangle \models \varphi_k$ and the preciseness of φ_k for S_k, $\sigma_2 \in \mathcal{M}(S_k)$. By $\sigma = \sigma_1\sigma_2$ and $\sigma_1 \in \mathcal{M}(\textbf{delay } K_g)$, we obtain $\mathcal{G}(g_k)(\sigma^b.s)$. By the definition of SEQ, we have $\sigma \in \mathcal{M}(\textbf{delay } K_g; S_k)$.

Both cases lead to $\sigma \in \mathcal{M}([\,[]_{i=1}^n g_i \rightarrow S_i])$. Hence φ is precise for $[\,[]_{i=1}^n b_i \rightarrow S_i]$.

Guarded Command with IO-Guards

Consider $G \equiv [\,[]_{i=1}^n g_i; c_i?x_i \rightarrow S_i \;[]\; g_0; \textbf{delay } e \rightarrow S_0]$. By the induction hypothesis, we have $c_i?x_i; S_i$ **sat** φ_i and S_0 **sat** φ_0, where φ_i is precise for $c_i?x_i; S_i$, $i = 1, 2, \ldots, n$, and φ_0 is precise for S_0. By the variable invariance axiom, the communication invariance axiom, and the conjunction rule, we obtain

$c_i?x_i; S_i$ **sat** $\varphi_i \wedge \Box\,(inv(wvar(G) \setminus wvar(c_i?x_i; S_i)) \wedge empty(dch(G) \setminus dch(c_i?x_i; S_i)))$.

Similarly, we have S_0 **sat** $\varphi_0 \wedge \Box\,(inv(wvar(G) \setminus wvar(S_0)) \wedge empty(dch(G) \setminus dch(S_0)))$.

By the guarded command evaluation axiom, the guarded command with IO-guards rule, and the conjunction rule, we obtain G **sat** ψ with

$$\psi \equiv [(inv(wvar(G)) \wedge empty(dch(G)))\ \mathcal{U}\ (T = start + K_g \wedge inv(wvar(G)))] \wedge$$
$$(\neg\bar{g} \rightarrow Eval) \wedge [\bar{g} \rightarrow (Eval\ \ \mathcal{C}\ \ (NComm \vee NTimeout))]$$

where

$NComm \equiv (Wait\ \mathbf{U}\ InTime)\ \mathcal{C}\ \psi_1, \qquad NTimeOut \equiv (Wait\ \mathcal{U}\ EndTime)\ \mathcal{C}\ \psi_2$

with

$$\psi_1 \equiv \bigvee_{i=1}^n [g_i \wedge \varphi_i \wedge comm(c_i) \wedge \Box\,(inv(wvar(G) \setminus wvar(c_i?x_i; S_i)) \wedge$$
$$empty(dch(G) \setminus dch(c_i?x_i; S_i)))]$$

$$\psi_2 \equiv \varphi_0 \wedge \Box\,(inv(wvar(G) \setminus wvar(S_0)) \wedge empty(dch(G) \setminus dch(S_0)))$$

We prove that ψ is precise for G.

Consider a well-formed model σ such that $dch(\sigma) \subseteq dch(G)$ and any variable $y \notin wvar(G)$ is invariant with respect to σ. Assume $\langle\sigma, begin(\sigma)\rangle \models \psi$. We prove $\sigma \in \mathcal{M}(G)$. Similar to the preciseness proof for $G \equiv [\,[]_{i=1}^n g_i \rightarrow S_i]$, we have that, for any τ_1,

$begin(\sigma) \leq \tau_1 < begin(\sigma) + K_g$, $\sigma(\tau_1).c = \emptyset$, and for any τ_1', $begin(\sigma) \leq \tau_1' \leq begin(\sigma) + K_g$, $\sigma(\tau_1').s = \sigma^b.s$.

Next consider the validity of $\bar{g}$. There are two possibilities.

- If $\langle \sigma, begin(\sigma)\rangle \models \neg\bar{g}$, lemma 2.6.2 leads to $\mathcal{G}(\neg\bar{g})(\sigma^b.s)$. By assumption, we have $\langle \sigma, begin(\sigma)\rangle \models term = start + K_g$ and then $end(\sigma) = begin(\sigma) + K_g$. Then we obtain $\sigma \in \mathcal{M}(\mathbf{delay}\ K_g)$. Hence $\sigma \in \mathcal{M}(G)$.

- If $\langle \sigma, begin(\sigma)\rangle \models \bar{g}$, then we have $\langle \sigma, begin(\sigma)\rangle \models (term = start + K_g)\ \mathcal{C}$ $[((Wait\ \mathbf{U}\ InTime)\ \mathcal{C}\ \psi_1) \vee ((Wait\ \mathcal{U}\ EndTime)\ \mathcal{C}\ \psi_2)]$.

For this case, consider the further three possibilities.

1. If $\langle \sigma, begin(\sigma)\rangle \models (term = start + K_g)\ \mathcal{C}\ ((Wait\ \mathcal{U}\ InTime)\,\mathcal{C}\,\psi_1)$, then there exist models σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $\langle \sigma_1, begin(\sigma_1)\rangle \models term = start + K_g$, and $\langle \sigma_2, begin(\sigma_2)\rangle \models (Wait\ \mathcal{U}\ InTime)\ \mathcal{C}\ \psi_1$. Then we have $end(\sigma_1) = begin(\sigma_1) + K_g$. By $begin(\sigma) = begin(\sigma_1)$, we obtain $\sigma_1 \in \mathcal{M}(\mathbf{delay}\ K_g)$.
Furthermore, there exist models σ_{21} and σ_{22} such that $\sigma_2 = \sigma_{21}\sigma_{22}$, $\langle \sigma_{21}, begin(\sigma_{21})\rangle \models Wait\ \mathcal{U}\ InTime$, and $\langle \sigma_{22}, begin(\sigma_{22})\rangle \models \psi_1$. We prove that $\sigma_{21} \in FinWait(G)\ \cup\ AnyWait(G)$ and $\sigma_{22} \in Comm(G)$.
By definition, there exists a $\tau_2 \geq begin(\sigma_{21})$ such that $\langle \sigma_{21}, \tau_2\rangle \models inv(wvar(G)) \wedge (T = term) \wedge (g_0 \rightarrow T < start + max(0, e))$ and for any τ_2', $begin(\sigma_{21}) \leq \tau_2' < \tau_2$, $\langle \sigma_{21}, \tau_2'\rangle \models inv(wvar(G)) \wedge empty(dch(G) \setminus \{c_1?, ..., c_n?\}) \wedge (g_0 \rightarrow T < start + max(0, e)) \wedge \bigwedge_{i=1}^{n}(g_i \leftrightarrow wait(c_i?))$. Then we obtain $end(\sigma_{21}) = \tau_2$ and, for any $y \in wvar(G)$, for any τ_2'', $begin(\sigma_{21}) \leq \tau_2'' \leq \tau_2$, $\sigma_{21}(\tau_2'').s(y) = \sigma_{21}^b.s(y)$. Together with the invariance of variables $y \notin wvar(G)$, we obtain $\sigma_{21}(\tau_2'').s = \sigma_{21}^b.s$. Since σ is a well-formed model, so are σ_{21} and σ_{22}. From above, we obtain $\sigma_{21}(\tau_2').c = \{c_i? \mid \mathcal{G}(g_i)(\sigma_{21}^b.s), 1 \leq i \leq n\}$. By assumption, $\langle \sigma, begin(\sigma)\rangle \models \bar{g}$. By lemma 2.6.2, $\mathcal{G}(\bar{g})(\sigma^b.s)$ and hence $\mathcal{G}(\bar{g})(\sigma_{21}^b.s)$.
If $\langle \sigma_{21}, begin(\sigma_{21})\rangle \models g_0$, lemma 2.6.2 leads to $\mathcal{G}(g_0)(\sigma_{21}^b.s)$. From $\langle \sigma_{21}, \tau_2\rangle \models g_0 \rightarrow T < start + max(0, e)$, we obtain $\tau_2 < begin(\sigma_{21}) + max(0, \mathcal{E}(e)(\sigma_{21}(\tau_2).s))$. Then we have $end(\sigma_{21}) < begin(\sigma_{21}) + max(0, \mathcal{E}(e)(\sigma_{21}^b.s))$ and then $\sigma_{21} \in FinWait(G)$.
If $\langle \sigma_{21}, begin(\sigma_{21})\rangle \models \neg g_0$, we obtain $\sigma_{21} \in AnyWait(G)$.
Next consider σ_{22}. Since $\langle \sigma_{22}, begin(\sigma_{22})\rangle \models \psi_1$, there exists a k, $1 \leq k \leq n$, such that $\langle \sigma_{22}, begin(\sigma_{22})\rangle \models g_k \wedge \varphi_k \wedge comm(c_k) \wedge \Box\,(inv(wvar(G) \setminus wvar(c_k?x_k; S_k)) \wedge empty(dch(G) \setminus dch(c_k?x_k; S_k)))$. From lemma 2.6.2, we have $\mathcal{G}(g_k)(\sigma_{22}^b.s)$. From $\langle \sigma_{22}, begin(\sigma_{22})\rangle \models \Box\,(inv(wvar(G) \setminus wvar(c_k?x_k; S_k)))$, any variable $x \in wvar(G) \setminus wvar(c_k?x_k; S_k)$ is invariant with respect to σ_{22}. By assumption, any variable $y \notin wvar(G)$ is invariant with respect to σ. Thus, any variable $z \notin wvar(c_k?x_k; S_k)$ is invariant with respect to σ_{22}. By lemma 2.6.10, $[\sigma_{22}]_{dch(G) \cup dch(c_k?x_k; S_k)} = [\sigma_{22}]_{dch(c_k?x_k; S_k)}$

and then $[\sigma_{22}]_{dch(G)} = [\sigma_{22}]_{dch(c_k?x_k;S_k)}$. Using $dch(\sigma) \subseteq dch(G)$, we obtain $dch(\sigma_{22}) \subseteq dch(\sigma) \subseteq dch(G)$. By lemma 2.6.9, $\sigma_{22} = [\sigma_{22}]_{dch(G)}$. Thus, $\sigma_{22} = [\sigma_{22}]_{dch(c_k?x_k;S_k)}$. By lemma 2.6.9 again, we have $dch(\sigma_{22}) \subseteq dch(c_k?x_k;S_k)$. Together with the well-formedness of σ_{22}, $\langle\sigma_{22}, begin(\sigma_{22})\rangle \models \varphi_k$, and the preciseness of φ_k for $c_k?x_k;S_k$, we obtain $\sigma_{22} \in \mathcal{M}(c_k?x_k;S_k)$. Since $\mathcal{M}(c_k?x_k;S_k) = SEQ(\mathcal{M}(c_k?x_k), \mathcal{M}(S_k))$ and $\langle\sigma_{22}, begin(\sigma_{22})\rangle \models comm(c_k)$, we have $\sigma_{22} \in SEQ(Receive(c_k, x_k), \mathcal{M}(S_k))$. Thus we obtain $\sigma_{22} \in Comm(G)$.
By $\sigma_2 = \sigma_{21}\sigma_{22}$, we obtain
$\sigma_2 \in SEQ(FinWait(G), Comm(G)) \cup SEQ(AnyWait(G), Comm(G))$.
By $\sigma = \sigma_1\sigma_2$ and $\sigma_1 \in \mathcal{M}(\textbf{delay } K_g)$, we have
$\sigma \in SEQ(\mathcal{M}(\textbf{delay } K_g), FinWait(G), Comm(G)) \cup$
$SEQ(\mathcal{M}(\textbf{delay } K_g), AnyWait(G), Comm(G))$ and hence $\sigma \in \mathcal{M}(G)$.

2. If $\langle\sigma, begin(\sigma)\rangle \models (term = start + K_g)\ \mathcal{C}\ \Box Wait$, there exist σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $\langle\sigma_1, begin(\sigma_1)\rangle \models term = start + K_g$, and $\langle\sigma_2, begin(\sigma_2)\rangle \models \Box Wait$. Then $\sigma_1 \in \mathcal{M}(\textbf{delay } K_g)$. From $\langle\sigma_2, begin(\sigma_2)\rangle \models \Box Wait$, we obtain that, for any $\tau_2 \geq begin(\sigma_2)$, $\langle\sigma_2, \tau_2\rangle \models Wait$. Hence we have $\langle\sigma_2, \tau_2\rangle \models g_0 \rightarrow T < start + max(0, e)$. If $\langle\sigma_2, \tau_2\rangle \models g_0$, we obtain $\tau_2 < begin(\sigma_2) + max(0, \mathcal{E}(e)(\sigma(\tau_2).s))$. But it can not be true. Hence $\langle\sigma_2, \tau_2\rangle \models \neg g_0$. By lemma 2.6.2, $\mathcal{G}(\neg g_0)(\sigma_2(\tau_2).s)$ and then $\mathcal{G}(\neg g_0)(\sigma_2^b.s)$. Next we prove $end(\sigma_2) = \infty$. Suppose $end(\sigma_2) < \infty$. By definition, for any $\tau_3 \geq end(\sigma_2)$, we have $\langle\sigma_2, \tau_3\rangle \models empty(dch(G))$. By assumption, $\langle\sigma, begin(\sigma)\rangle \models \bar{g}$. Since $\mathcal{G}(\neg g_0)(\sigma^b.s)$, there exists a k, $1 \leq k \leq n$, such that $\langle\sigma, begin(\sigma)\rangle \models g_k$. Then, for any $\tau_2 \geq begin(\sigma_2)$, $\langle\sigma_2, \tau_2\rangle \models wait(c_k)$ and hence $\langle\sigma_2, \tau_2\rangle \models \neg empty(dch(G))$. This contradiction leads to $end(\sigma_2) = \infty$. We also have $\sigma_2(\tau_2).s = \sigma_2^b.s$ and $\sigma_2(\tau_2).c = \{c? \mid \mathcal{G}(g_i)(\sigma_2^b.s), 1 \leq i \leq n\}$. Hence $\sigma_2 \in AnyWait(G)$.
We can easily find a model which belongs to $Comm(G)$. Let σ_3 be a model such that $\sigma_3 \in Comm(G)$. By the definition of SEQ, we have $\sigma_2\sigma_3 \in SEQ(AnyWait(G), Comm(G))$. Since $end(\sigma_2) = \infty$, we have $\sigma_2\sigma_3 = \sigma_2$. Thus
$\sigma_2 \in SEQ(AnyWait(G), Comm(G))$.
Together with $\sigma = \sigma_1\sigma_2$ and $\sigma_1 \in \mathcal{M}(\textbf{delay } K_g)$, we obtain
$\sigma \in SEQ(\mathcal{M}(\textbf{delay } K_g), AnyWait(G), Comm(G))$ and hence $\sigma \in \mathcal{M}(G)$.

3. If $\langle\sigma, begin(\sigma)\rangle \models (term = start + K_g)\ \mathcal{C}\ ((Wait\ \mathcal{U}\ EndTime)\ \mathcal{C}\ \psi_2)$, there exist σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $\langle\sigma_1, begin(\sigma_1)\rangle \models term = start + K_g$, and $\langle\sigma_2, begin(\sigma_2)\rangle \models (Wait\ \mathcal{U}\ EndTime)\ \mathcal{C}\ \psi_2$. Thus $\sigma_1 \in \mathcal{M}(\textbf{delay } K_g)$.
Furthermore, there exist models σ_{21} and σ_{22} such that $\sigma_2 = \sigma_{21}\sigma_{22}$, $\langle\sigma_{21}, begin(\sigma_{21})\rangle \models Wait\ \mathcal{U}\ EndTime$, and $\langle\sigma_{22}, begin(\sigma_{22})\rangle \models \psi_2$. We prove that

$\sigma_{21} \in TimeOut(G)$ and $\sigma_{22} \in \mathcal{M}(S)$.

By definition, there exists a $\tau_2 \geq begin(\sigma_{21})$ such that $\langle \sigma_{21}, \tau_2 \rangle \models EndTime$ and, for any τ_2', $begin(\sigma_{21}) \leq \tau_2' < \tau_2$, $\langle \sigma_{21}, \tau_2' \rangle \models Wait$. Then we have $\langle \sigma_{21}, \tau_2 \rangle \models inv(wvar(G)) \wedge g_0 \wedge T = term = start + max(0, e)$. Then $end(\sigma_{21}) = \tau_2 = begin(\sigma_{21}) + max(0, \mathcal{E}(e)(\sigma_{21}(\tau_2).s))$ and, by lemma 2.6.2, $\mathcal{G}(g_0)(\sigma_{21}(\tau_2).state)$. We also have that, for any τ_2'', $begin(\sigma_{21}) \leq \tau_2'' \leq \tau_2$, $\sigma_{21}(\tau_2'').s = \sigma_{21}^b.s$ and, for any τ_2', $begin(\sigma_{21}) \leq \tau_2' < \tau_2$, $\sigma_{21}(\tau_2').c = \{c_i? \mid \mathcal{G}(g_i)(\sigma_{21}^b.s), 1 \leq i \leq n\}$. Thus $end(\sigma_{21}) = begin(\sigma_{21}) + max(0, \mathcal{E}(e)(\sigma_{21}^b.s))$ and $\mathcal{G}(g_0)(\sigma_{21}^b.s)$. Hence $\sigma_{21} \in TimeOut(G)$.

Next consider σ_{22}. Since $\langle \sigma_{22}, begin(\sigma_{22}) \rangle \models \psi_2$, any variable $x \in wvar(G) \setminus wvar(S)$ is invariant with respect to σ_{22}. By assumption, any variable $y \notin wvar(G)$ is invariant with respect to σ. Hence, any variable $z \notin wvar(S)$ is invariant with respect to σ_{22}. By lemma 2.6.10, $[\sigma_{22}]_{dch(G) \cup dch(S)} = [\sigma_{22}]_{dch(S)}$ and then $[\sigma_{22}]_{dch(G)} = [\sigma_{22}]_{dch(S)}$. Using $dch(\sigma) \subseteq dch(G)$, we have $dch(\sigma_{22}) \subseteq dch(\sigma) \subseteq dch(G)$. By lemma 2.6.9, $\sigma_{22} = [\sigma_{22}]_{dch(G)}$ and hence $\sigma_{22} = [\sigma_{22}]_{dch(S)}$. By lemma 2.6.9 again, $dch(\sigma_{22}) \subseteq dch(S)$. Together with the well-formedness of σ_{22}, $\langle \sigma_{22}, begin(\sigma_{22}) \rangle \models \varphi_0$, and the preciseness of φ_0 for S_0, we obtain $\sigma_{22} \in \mathcal{M}(S)$.

By $\sigma_2 = \sigma_{21}\sigma_{22}$, we have $\sigma_2 \in SEQ(TimeOut(G), \mathcal{M}(S))$.

By $\sigma = \sigma_1\sigma_2$, we obtain $\sigma \in SEQ(\mathcal{M}(\mathbf{delay}\ K_g), TimeOut(G), \mathcal{M}(S))$ and hence $\sigma \in \mathcal{M}(G)$.

Therefore all the cases lead to $\sigma \in \mathcal{M}(G)$. Hence, ψ is precise for $G \equiv [[]_{i=1}^{n} g_i; c_i?x_i; S_i \rightarrow S_i \; [] \; g_0; \mathbf{delay}\ e \rightarrow S_0]$.

Iteration

Consider $\star G$. By the induction hypothesis, we can derive G **sat** φ where φ is precise for G. By the iteration rule, $\star G$ **sat** ψ with $\psi \equiv (\bar{g} \wedge \varphi)\ \mathcal{C}^*\ (\neg\bar{g} \wedge \varphi)$. We prove that ψ is precise for $\star G$.

Consider a well-formed model σ such that $dch(\sigma) \subseteq dch(\star G)$ and any variable $y \notin wvar(\star G)$ is invariant with respect to σ. Thus, $dch(\sigma) \subseteq dch(G)$ and any variable $y \notin wvar(G)$ is invariant with respect to σ. Assume $\langle \sigma, begin(\sigma) \rangle \models \psi$. By definition of the $\mathcal{C}^*$ operator, there are two possibilities:

1. either there exists a $k \geq 1$ and models $\sigma_1, \sigma_2, \ldots, \sigma_k$ such that $\sigma = \sigma_1\sigma_2\ldots\sigma_k$, for any j, $1 \leq j \leq k-1$, $end(\sigma_j) < \infty$, $\langle \sigma_j, begin(\sigma_j) \rangle \models \bar{g} \wedge \varphi$, and if $end(\sigma_k) < \infty$, then $\langle \sigma_k, begin(\sigma_k) \rangle \models \neg\bar{g} \wedge \varphi$, otherwise $\langle \sigma_k, begin(\sigma_k) \rangle \models \bar{g} \wedge \varphi$,

2. or there exist infinite models $\sigma_1, \sigma_2, \ldots$ such that $\sigma = \sigma_1\sigma_2\ldots$, for any $j \geq 1$, $end(\sigma_j) < \infty$, $\langle \sigma_j, begin(\sigma_j) \rangle \models \bar{g} \wedge \varphi$.

That is,

1. Either there exists a $k \geq 1$ and models $\sigma_1, \sigma_2, \ldots, \sigma_k$ such that $\sigma = \sigma_1\sigma_2\ldots\sigma_k$, for any j, $1 \leq j \leq k-1$, $end(\sigma_j) < \infty$, $\mathcal{G}(\bar{g})(\sigma_j^b.s)$ (by lemma 2.6.2). Since σ is well-formed, so are $\sigma_1, \sigma_2, \ldots, \sigma_k$. By $dch(\sigma) \subseteq dch(G)$, we obtain $dch(\sigma_j) \subseteq dch(G)$. Together with the invariance of variables $y \notin wvar(G)$ and the preciseness of φ for G, we have $\sigma_j \in \mathcal{M}(G)$. Similarly, we have $\sigma_k \in \mathcal{M}(G)$. If $end(\sigma_k) < \infty$, by lemma 2.6.2, we obtain $\mathcal{G}(\neg\bar{g})(\sigma_k^b.s)$, otherwise $\mathcal{G}(\bar{g})(\sigma_k^b.s)$;

2. Or there exist infinite models $\sigma_1, \sigma_2, \ldots$ such that $\sigma = \sigma_1\sigma_2\ldots$, for any $j \geq 1$, $end(\sigma_j) < \infty$, $\mathcal{G}(\bar{g})(\sigma_j^b.s)$, and $\sigma_j \in \mathcal{M}(G)$.

Both cases lead to $\sigma \in \mathcal{M}(\star G)$. Hence, $(\bar{g} \wedge \varphi)\ \mathcal{C}^*\ (\neg\bar{g} \wedge \varphi)$ is precise for $\star G$.

Parallel Composition

Consider $S \equiv S_1 \| S_2$. By the induction hypothesis, we can derive S_1 **sat** φ_1 and S_2 **sat** φ_2 with φ_1 and φ_2 precise for S_1 and S_2, respectively. From preciseness, $dch(\varphi_i) \subseteq dch(S_i)$ and $var(\varphi_i) \subseteq var(S_i)$, for $i = 1, 2$. Then we can apply the general parallel composition rule and obtain $S_1 \| S_2$ **sat** ψ with $\psi \equiv (\varphi_1 \wedge (\varphi_2\, \mathcal{C}\, \psi_2)) \vee (\varphi_2 \wedge (\varphi_1\, \mathcal{C}\, \psi_1))$ where $\psi_i \equiv \Box\,[inv(var(S_i)) \wedge empty(dch(S_i))]$, for $i = 1, 2$. We prove that ψ is precise for $S_1 \| S_2$.

Let σ be a well-formed model such that $dch(\sigma) \subseteq dch(S_1 \| S_2)$ and any variable $y \notin wvar(S_1 \| S_2)$ is invariant with respect to σ. Assume $\langle \sigma, begin(\sigma)\rangle \models \psi$. By the well-formedness of σ, for any $c \in CHAN$, any τ, $begin(\sigma) \leq \tau < end(\sigma)$, $\neg(c! \in \sigma(\tau).c \wedge c? \in \sigma(\tau).c)$ holds. Suppose $\langle \sigma, begin(\sigma)\rangle \models \varphi_1 \wedge (\varphi_2\, \mathcal{C}\, \psi_2)$. Define σ_1 as $[\sigma \downarrow var(S_1)]_{dch(S_1)}$. From $\langle \sigma, begin(\sigma)\rangle \models \varphi_1$ and $var(\varphi_1) \subseteq var(S_1)$, lemma 2.6.8 leads to $\langle \sigma \downarrow var(S_1), begin(\sigma)\rangle \models \varphi_1$. By $dch(\varphi_1) \subseteq dch(S_1)$ and lemma 2.6.7, we obtain $\langle [\sigma \downarrow var(S_1)]_{dch(S_1)}, begin(\sigma)\rangle \models \varphi_1$, i.e., $\langle \sigma_1, begin(\sigma_1)\rangle \models \varphi_1$. Since σ is well-formed, σ_1 is also well-formed. By the definition of σ and σ_1, any variable $y \notin wvar(S_1)$ is invariant with respect to σ_1. Together with the preciseness of φ_1 for S_1 and $dch(\sigma_1) \subseteq dch(S_1)$, we obtain $\sigma_1 \in \mathcal{M}(S_1)$.

Next consider $\langle \sigma, begin(\sigma)\rangle \models \varphi_2 \mathcal{C} \psi_2$. There exist models σ_3 and σ_4 such that $\sigma = \sigma_3\sigma_4$, $\langle \sigma_3, begin(\sigma_3)\rangle \models \varphi_2$, and $\langle \sigma_4, begin(\sigma_4)\rangle \models \psi_2$. Define σ_2 as $[\sigma_3 \downarrow var(S_2)]_{dch(S_2)}$. Similarly, by lemma 2.6.8 and lemma 2.6.7, we obtain $\sigma_2 \in \mathcal{M}(S_2)$.

Notice that $end(\sigma) = end(\sigma_3\sigma_4) \geq end(\sigma_3) = end(\sigma_2)$ and $end(\sigma) = end(\sigma_1)$, hence $end(\sigma) = max(end(\sigma_1), end(\sigma_2))$. It is clear that $begin(\sigma) = begin(\sigma_1) = begin(\sigma_2)$. By definitions, we have that, for $i = 1, 2$,

$$[\sigma]_{dch(S_i)}(\tau).c = \begin{cases} \sigma_i(\tau).c & begin(\sigma_i) \le \tau < end(\sigma_i) \\ \emptyset & end(\sigma_i) \le \tau < end(\sigma) \end{cases}$$

$$(\sigma \downarrow var(S_i))(\tau).s = \begin{cases} \sigma_i(\tau).s & begin(\sigma_i) \le \tau \le end(\sigma_i) \\ \sigma_i^e.s & end(\sigma_i) < \tau \le end(\sigma) \end{cases}$$

By the assumption, any variable $y \notin wvar(S_1\|S_2)$ is invariant w.r.t. to σ. Thus, any variable $x \notin var(S_1\|S_2)$ is invariant w.r.t. to σ, i.e., for any τ, $begin(\sigma) \le \tau \le end(\sigma)$, $\sigma(\tau).s(x) = \sigma^b.s(x)$. Furthermore, for any $x \notin var(S_1\|S_2)$, first assume $x \notin var(S_1)$. Then by the definition of σ_1, we have $\sigma^b.s(x) = \sigma_1^b.s(x)$. There are two possibilities:

- if $x \in var(S_2)$, then by the definition of σ_2, we have $\sigma^b.s(x) = \sigma_2^b.s(x)$,
- if $x \notin var(S_2)$, we also have $\sigma^b.s(x) = \sigma_2^b.s(x)$.

This leads to $\sigma^b.s(x) = \sigma_i^b.s(x)$, for $i = 1, 2$.
Second, when $x \notin var(S_2)$, we again have $\sigma^b.s(x) = \sigma_i^b.s(x)$.
Hence, for any variable $x \notin var(S_1\|S_2)$, for any τ, $begin(\sigma) \le \tau \le end(\sigma)$, we obtain $\sigma(\tau).s(x) = \sigma_i^b.s(x)$, for $i = 1, 2$.
Thus $\sigma \in \mathcal{M}(S_1\|S_2)$.
Similarly, if $\langle\sigma, begin(\sigma)\rangle \models \varphi_2 \wedge (\varphi_1\, \mathcal{C}\, \psi_1)$, we can also prove that $\sigma \in \mathcal{M}(S_1\|S_2)$.
Therefore ψ is indeed precise for $S_1\|S_2$.

Appendix D

Proofs of Lemmas in Chapter 3

Lemma 3.5.1 and lemma 3.5.2 can be proved similarly as in Appendix A for lemma 2.6.1 and lemma 2.6.2, respectively. Notice that adding a buffer b does not influence the proofs.

Proof of Lemma 3.5.3

For any expression $qexp$ of type QUE, any $cset \subseteq CHAN$, and any buffers b_1 and b_2, if $ich(qexp) \subseteq cset$ and for any $c \in cset$, $b_1(c) = b_2(c)$, we prove that, for any model σ and any $\tau \geq begin(\sigma)$, $\mathcal{Q}(qexp)(\sigma, b_1, \tau) = \mathcal{Q}(qexp)(\sigma, b_2, \tau)$ by induction on the structure of $qexp$.

- $qexp \equiv w$. $\mathcal{Q}(w)(\sigma, b_1, \tau) = w = \mathcal{Q}(w)(\sigma, b_2, \tau)$.
- $qexp \equiv init(c)$. $\mathcal{Q}(init(c))(\sigma, b_1, \tau) = b_1(c) = b_2(c) = \mathcal{Q}(init(c))(\sigma, b_2, \tau)$.

Proof of Lemma 3.5.4

For any expression $qexp$ of type QUE, any model σ, any buffer b, any $cset \subseteq CHAN$, and any $\tau \geq begin(\sigma)$, we prove $\mathcal{Q}(qexp)(\sigma, b, \tau) = \mathcal{Q}(qexp)([\sigma]^R_{cset}, b, \tau)$ by induction on the structure of $qexp$.

- $qexp \equiv w$. $\mathcal{Q}(w)(\sigma, b, \tau) = w = \mathcal{Q}(w)([\sigma]^R_{cset}, b, \tau)$.
- $qexp \equiv init(c)$. $\mathcal{Q}(init(c))(\sigma, b, \tau) = b(c) = \mathcal{Q}(init(c))([\sigma]^R_{cset}, b, \tau)$.

Proof of Lemma 3.5.5

For any expression $qexp$ of type QUE, any model σ, any buffer b, any $vset \subseteq VAR$, and any $\tau \geq begin(\sigma)$, we prove $\mathcal{Q}(qexp)(\sigma, b, \tau) = \mathcal{Q}(qexp)(\sigma \downarrow vset, b, \tau)$ by induction on the structure of $qexp$.

- $qexp \equiv w$. $\mathcal{Q}(w)(\sigma, b, \tau) = w = \mathcal{Q}(w)(\sigma \downarrow vset, b, \tau)$.
- $qexp \equiv init(c)$. $\mathcal{Q}(init(c))(\sigma, b, \tau) = b(c) = \mathcal{Q}(init(c))(\sigma \downarrow vset, b, \tau)$.

Proof of Lemma 3.5.6

For any expression $vexp$ of type VAL, any $cset \subseteq CHAN$, and any buffers b_1 and b_2, if $ich(vexp) \subseteq cset$ and for any $c \in cset$, $b_1(c) = b_2(c)$, we prove, by induction on the structure of $vexp$, that for any model σ and any $\tau \geq begin(\sigma)$, $\mathcal{V}(vexp)(\sigma, b_1, \tau) = \mathcal{V}(vexp)(\sigma, b_2, \tau)$.

- $vexp \equiv \vartheta$. $\mathcal{V}(\vartheta)(\sigma, b_1, \tau) = \vartheta = \mathcal{V}(\vartheta)(\sigma, b_2, \tau)$.
- $vexp \equiv x$. By definition, if $\tau \leq end(\sigma)$, then $\mathcal{V}(x)(\sigma, b_1, \tau) = \sigma(\tau).s(x)$, i.e., $\mathcal{V}(x)(\sigma, b_1, \tau) = \mathcal{V}(x)(\sigma, b_2, \tau)$. If $\tau > end(\sigma)$, then $\mathcal{V}(x)(\sigma, b_1, \tau) = \sigma^e.s(x)$, i.e., $\mathcal{V}(x)(\sigma, b_1, \tau) = \mathcal{V}(x)(\sigma, b_2, \tau)$. Hence $\mathcal{V}(x)(\sigma, b_1, \tau) = \mathcal{V}(x)(\sigma, b_2, \tau)$.
- $vexp \equiv first(x)$. $\mathcal{V}(first(x))(\sigma, b_1, \tau) = \sigma^b.s(x) = \mathcal{V}(first(x))(\sigma, b_2, \tau)$.
- $vexp \equiv first(qexp)$. $ich(vexp) = ich(qexp)$ and thus $ich(qexp) \subseteq cset$. By lemma 3.5.3, $\mathcal{Q}(qexp)(\sigma, b_1, \tau) = \mathcal{Q}(qexp)(\sigma, b_2, \tau)$. Then $\mathcal{V}(first(qexp))(\sigma, b_1, \tau) = First(\mathcal{Q}(qexp)(\sigma, b_1, \tau)) = First(\mathcal{Q}(qexp)(\sigma, b_2, \tau)) = \mathcal{V}(first(qexp))(\sigma, b_2, \tau)$.
- $vexp \equiv max(vexp_1, vexp_2)$. By the induction hypothesis, we have, for $i = 1, 2$, $\mathcal{V}(vexp_i)(\sigma, b_1, \tau) = \mathcal{V}(vexp_i)(\sigma, b_2, \tau)$. Then
$\mathcal{V}(max(vexp_1, vexp_2))(\sigma, b_1, \tau) = max(\mathcal{V}(vexp_1)(\sigma, b_1, \tau), \mathcal{V}(vexp_2)(\sigma, b_1, \tau))$
$= max(\mathcal{V}(vexp_1)(\sigma, b_2, \tau), \mathcal{V}(vexp_2)(\sigma, b_2, \tau)) = \mathcal{V}(max(vexp_1, vexp_2))(\sigma, b_2, \tau)$.
- $vexp \equiv vexp_1 \odot vexp_2$, where $\odot \in \{+, -, \times\}$. By the induction hypothesis, we have, for $i = 1, 2$, $\mathcal{V}(vexp_i)(\sigma, b_1, \tau) = \mathcal{V}(vexp_i)(\sigma, b_2, \tau)$. Thus
$\mathcal{V}(vexp_1 \odot vexp_2)(\sigma, b_1, \tau) = \mathcal{V}(vexp_1)(\sigma, b_1, \tau) \odot \mathcal{V}(vexp_2)(\sigma, b_1, \tau)$
$= \mathcal{V}(vexp_1)(\sigma, b_2, \tau) \odot \mathcal{V}(vexp_2)(\sigma, b_2, \tau) = \mathcal{V}(vexp_1 \odot vexp_2)(\sigma, b_2, \tau)$.

Proof of Lemma 3.5.7

For any expression $vexp$ of type VAL, any model σ, any buffer b, any $cset \subseteq CHAN$, and any $\tau \geq begin(\sigma)$, we prove $\mathcal{V}(vexp)(\sigma, b, \tau) = \mathcal{V}(vexp)([\sigma]^R_{cset}, b, \tau)$.

The proof is similar to the proof for lemma 2.6.3 except the following case:

- $vexp \equiv first(qexp)$. By lemma 3.5.4, $\mathcal{Q}(qexp)(\sigma, b, \tau) = \mathcal{Q}(qexp)([\sigma]^R_{cset}, b, \tau)$. Then $\mathcal{V}(first(qexp))(\sigma, b, \tau) = First(\mathcal{Q}(qexp)(\sigma, b, \tau)) = First(\mathcal{Q}(qexp)([\sigma]^R_{cset}, b, \tau)) = \mathcal{V}(first(qexp))([\sigma]^R_{cset}, b, \tau)$.

Proof of Lemma 3.5.8

For any expression $vexp$ of type VAL, any model σ, any buffer b, any $vset \subseteq VAR$, and any $\tau \geq begin(\sigma)$, if $var(vexp) \subseteq vset$, we prove
$\mathcal{V}(vexp)(\sigma, b, \tau) = \mathcal{V}(vexp)(\sigma \downarrow vset, b, \tau)$.

This proof is similar to the proof for lemma 2.6.4 except the following case:

- $vexp \equiv first(qexp)$. By lemma 3.5.5, $\mathcal{Q}(qexp)(\sigma, b, \tau) = \mathcal{Q}(qexp)(\sigma \downarrow vset, b, \tau)$. Then $\mathcal{V}(first(qexp))(\sigma, b, \tau) = First(\mathcal{Q}(qexp)(\sigma, b, \tau)) =$ $First(\mathcal{Q}(qexp)(\sigma \downarrow vset, b, \tau)) = \mathcal{V}(first(qexp))(\sigma \downarrow vset, b, \tau)$.

Proof of Lemma 3.5.9

For any expression $texp$ of type $TIME$, any $cset \subseteq CHAN$, and any buffers b_1 and b_2, if $ich(vexp) \subseteq cset$ and for any $c \in cset$, $b_1(c) = b_2(c)$, we prove, by induction on the structure of $texp$, that for any model σ and any $\tau \geq begin(\sigma)$, $\mathcal{T}(texp)(\sigma, b_1, \tau) = \mathcal{T}(texp)(\sigma, b_2, \tau)$.

- $texp \equiv \hat{\tau}$. $\mathcal{T}(\hat{\tau})(\sigma, b_1, \tau) = \hat{\tau} = \mathcal{T}(\hat{\tau})(\sigma, b_2, \tau)$.

- $texp \equiv T$. $\mathcal{T}(T)(\sigma, b_1, \tau) = \tau = \mathcal{T}(T)(\sigma, b_2, \tau)$.

- $texp \equiv start$. $\mathcal{T}(start)(\sigma, b_1, \tau) = begin(\sigma) = \mathcal{T}(start)(\sigma, b_2, \tau)$.

- $texp \equiv term$. $\mathcal{T}(term)(\sigma, b_1, \tau) = end(\sigma) = \mathcal{T}(term)(\sigma, b_2, \tau)$.

- $texp \equiv vexp$. By lemma 3.5.6, we have $\mathcal{V}(vexp)(\sigma, b_1, \tau) = \mathcal{V}(vexp)(\sigma, b_2, \tau)$. Then $\mathcal{T}(vexp)(\sigma, b_1, \tau) = \mathcal{V}(vexp)(\sigma, b_1, \tau) = \mathcal{V}(vexp)(\sigma, b_2, \tau) = \mathcal{T}(vexp)(\sigma, b_2, \tau)$.

- $texp \equiv texp_1 \odot texp_2$, where $\odot \in \{+, -, \times\}$. By the induction hypothesis, we have, for $i = 1, 2$, $\mathcal{T}(texp_i)(\sigma, b_1, \tau) = \mathcal{T}(texp_i)(\sigma, b_2, \tau)$. Then, by definition, $\mathcal{T}(texp_1 \odot texp_2)(\sigma, b_1, \tau) = \mathcal{T}(texp_1 \odot texp_2)(\sigma, b_2, \tau)$.

Lemma 3.5.10 and lemma 3.5.11 can be proved similarly as in Appendix A for lemma 2.6.5 and lemma 2.6.6, respectively.

Proof of Lemma 3.5.12

For any specification φ, any $cset \subseteq CHAN$, and any buffers b_1, b_2, if $ich(\varphi) \subseteq cset$ and for any $c \in cset$, $b_1(c) = b_2(c)$, we prove, by induction on the structure of φ, that for any model σ and any $\tau \geq begin(\sigma)$, $\langle \sigma, b_1, \tau\rangle \models \varphi$ iff $\langle \sigma, b_2, \tau\rangle \models \varphi$.

- $\varphi \equiv qexp_1 = qexp_2$. $\langle \sigma, b_1, \tau\rangle \models qexp_1 = qexp_2$ iff $\mathcal{Q}(qexp_1)(\sigma, b_1, \tau) = \mathcal{Q}(qexp_2)(\sigma, b_1, \tau)$ iff, by lemma 3.5.3, $\mathcal{Q}(qexp_1)(\sigma, b_2, \tau) = \mathcal{Q}(qexp_2)(\sigma, b_2, \tau)$ iff $\langle \sigma, b_2, \tau\rangle \models qexp_1 = qexp_2$.

- $\varphi \equiv texp_1 = texp_2$. $\langle \sigma, b_1, \tau\rangle \models texp_1 = texp_2$ iff $\mathcal{T}(texp_1)(\sigma, b_1, \tau) = \mathcal{T}(texp_2)(\sigma, b_1, \tau)$ iff, by lemma 3.5.9, $\mathcal{T}(texp_1)(\sigma, b_2, \tau) = \mathcal{T}(texp_2)(\sigma, b_2, \tau)$ iff $\langle \sigma, b_2, \tau\rangle \models texp_1 = texp_2$.

- $\varphi \equiv texp_1 < texp_2$. Similar to the proof for $\varphi \equiv texp_1 = texp_2$.

- $\varphi \equiv send(c, vexp)$. $ich(\varphi) = ich(vexp)$ and thus $ich(vexp) \subseteq cset$. Hence $\langle \sigma, b_1, \tau\rangle \models send(c, vexp)$ iff $\tau \leq end(\sigma)$ and $(c, \mathcal{V}(vexp)(\sigma, b_1, \tau)) \in \sigma(\tau).S$ iff, by lemma 3.5.6, $\tau \leq end(\sigma)$ and $(c, \mathcal{V}(vexp)(\sigma, b_2, \tau)) \in \sigma(\tau).S$ iff $\langle \sigma, b_2, \tau\rangle \models send(c, vexp)$.

- $\varphi \equiv send(c)$. $\langle \sigma, b_1, \tau\rangle \models send(c)$ iff $\tau \leq end(\sigma)$ and there exists a $\vartheta \in VAL$ such that $(c, \vartheta) \in \sigma(\tau).S$ iff $\langle \sigma, b_2, \tau\rangle \models send(c)$.

- $\varphi \equiv receive(c, vexp)$. $ich(\varphi) = \{c\} \cup ich(vexp)$ and thus $ich(vexp) \subseteq cset$. Hence $\langle \sigma, b_1, \tau\rangle \models receive(c, vexp)$ iff $\tau \leq end(\sigma)$ and $(c, \mathcal{V}(vexp)(\sigma, b_1, \tau)) \in \sigma(\tau).R$ iff, by lemma 3.5.6, $\tau \leq end(\sigma)$ and $(c, \mathcal{V}(vexp)(\sigma, b_2, \tau)) \in \sigma(\tau).R$ iff $\langle \sigma, b_2, \tau\rangle \models receive(c, vexp)$.

- $\varphi \equiv receive(c)$. $\langle \sigma, b_1, \tau\rangle \models receive(c)$ iff $\tau \leq end(\sigma)$ and there exists a $\vartheta \in VAL$ such that $(c, \vartheta) \in \sigma(\tau).R$ iff $\langle \sigma, b_2, \tau\rangle \models receive(c)$.

- $\varphi \equiv \varphi_1 \vee \varphi_2$. For $i = 1, 2$, $ich(\varphi_i) \subseteq (ich(\varphi_1) \cup ich(\varphi_2)) = ich(\varphi) \subseteq cset$. Hence $\langle \sigma, b_1, \tau\rangle \models \varphi_1 \vee \varphi_2$ iff $\langle \sigma, b_1, \tau\rangle \models \varphi_1$ or $\langle \sigma, b_1, \tau\rangle \models \varphi_2$ iff, by the induction hypothesis, $\langle \sigma, b_2, \tau\rangle \models \varphi_1$ or $\langle \sigma, b_2, \tau\rangle \models \varphi_2$ iff $\langle \sigma, b_2, \tau\rangle \models \varphi_1 \vee \varphi_2$.

- $\varphi \equiv \neg\varphi_1$ and $\varphi \equiv \varphi_1 \,\mathcal{U}\, \varphi_2$. Similar to the proof for $\varphi \equiv \varphi_1 \vee \varphi_2$.

- $\varphi \equiv \varphi_1 \;\mathcal{C}\; \varphi_2$. For $i = 1, 2$, $ich(\varphi_i) \subseteq ich(\varphi) \subseteq cset$. Hence $\langle \sigma, b_1, \tau\rangle \models \varphi_1 \;\mathcal{C}\; \varphi_2$ iff
  - either $\langle \sigma, b_1, \tau\rangle \models \varphi_1$ and $end(\sigma) = \infty$ iff, by the induction hypothesis, $\langle \sigma, b_2, \tau\rangle \models \varphi_1$ and $end(\sigma) = \infty$ iff $\langle \sigma, b_2, \tau\rangle \models \varphi_1 \;\mathcal{C}\; \varphi_2$,

- or there exist models σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $\tau \leq end(\sigma_1) < \infty$, $\langle\sigma_1, b_1, \tau\rangle \models \varphi_1$, and $\langle\sigma_2, Buf(b_1, \sigma_1), begin(\sigma_2)\rangle \models \varphi_2$ iff, since for any $c \in cset$, $b_1(c) = b_2(c)$ and thus $Buf(b_1, \sigma_1)(c) = Buf(b_2, \sigma_1)(c)$, by the induction hypothesis, there exist models σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $\langle\sigma_1, b_2, \tau\rangle \models \varphi_1$, and $\langle\sigma_2, Buf(b_2, \sigma_1), begin(\sigma_2)\rangle \models \varphi_2$ iff $\langle\sigma, b_2, \tau\rangle \models \varphi_1 \;\mathcal{C}\; \varphi_2$.

- $\varphi \equiv \varphi_1 \;\mathcal{C}^*\; \varphi_2$. Similar to the proof for $\varphi \equiv \varphi_1 \;\mathcal{C}\; \varphi_2$.

Proof of Lemma 3.5.13

For any $cset \subseteq CHAN$ and any specification φ, if $ich(\varphi) \subseteq cset$, we prove, by induction on the structure of φ, that for any model σ, any buffer b, and any $\tau \geq begin(\sigma)$, $\langle\sigma, b, \tau\rangle \models \varphi$ iff $\langle[\sigma]^R_{cset}, b, \tau\rangle \models \varphi$.

- $\varphi \equiv qexp_1 = qexp_2$. $\langle\sigma, b, \tau\rangle \models qexp_1 = qexp_2$ iff $\mathcal{Q}(qexp_1)(\sigma, b, \tau) = \mathcal{Q}(qexp_2)(\sigma, b, \tau)$ iff, by lemma 3.5.4, $\mathcal{Q}(qexp_1)([\sigma]^R_{cset}, b, \tau) = \mathcal{Q}(qexp_2)([\sigma]^R_{cset}, b, \tau)$ iff $\langle[\sigma]^R_{cset}, b, \tau\rangle \models qexp_1 = qexp_2$.

- $\varphi \equiv texp_1 = texp_2$. $\langle\sigma, b, \tau\rangle \models texp_1 = texp_2$ iff $\mathcal{T}(texp_1)(\sigma, b, \tau) = \mathcal{T}(texp_2)(\sigma, b, \tau)$ iff, by lemma 3.5.10, $\mathcal{T}(texp_1)([\sigma]^R_{cset}, b, \tau) = \mathcal{T}(texp_2)([\sigma]^R_{cset}, b, \tau)$ iff $\langle[\sigma]^R_{cset}, b, \tau\rangle \models texp_1 = texp_2$.

- $\varphi \equiv texp_1 < texp_2$. Similar to the proof for $\varphi \equiv texp_1 = texp_2$.

- $\varphi \equiv send(c, vexp)$. $\langle\sigma, b, \tau\rangle \models send(c, vexp)$ iff $\tau \leq end(\sigma)$ and $(c, \mathcal{V}(vexp)(\sigma, b, \tau)) \in \sigma(\tau).S$ iff, by definition and lemma 3.5.7, $\tau \leq end([\sigma]^R_{cset})$ and $(c, \mathcal{V}(vexp)([\sigma]^R_{cset}, b, \tau)) \in [\sigma]^R_{cset}(\tau).S$ iff $\langle[\sigma]^R_{cset}, b, \tau\rangle \models send(c, vexp)$.

- $\varphi \equiv send(c)$. $\langle\sigma, b, \tau\rangle \models send(c)$ iff $\tau \leq end(\sigma)$ and there exists a $\vartheta \in VAL$ such that $(c, \vartheta) \in \sigma(\tau).S$ iff, by definition, $\tau \leq end([\sigma]^R_{cset})$ and there exists a $\vartheta \in VAL$ such that $(c, \vartheta) \in [\sigma]^R_{cset}(\tau).S$ iff $\langle[\sigma]^R_{cset}, b, \tau\rangle \models send(c)$.

- $\varphi \equiv receive(c, vexp)$. $ich(\varphi) = \{c\} \cup ich(vexp)$ and thus $c \in cset$. Hence $\langle\sigma, b, \tau\rangle \models receive(c, vexp)$ iff $\tau \leq end(\sigma)$ and $(c, \mathcal{V}(vexp)(\sigma, b, \tau)) \in \sigma(\tau).R$ iff, by definition and lemma 3.5.7, $\tau \leq end([\sigma]^R_{cset})$ and $(c, \mathcal{V}(vexp)([\sigma]^R_{cset}, b, \tau)) \in [\sigma]^R_{cset}(\tau).R$ iff $\langle[\sigma]^R_{cset}, b, \tau\rangle \models receive(c, vexp)$.

- $\varphi \equiv receive(c)$. $ich(\varphi) = \{c\}$ and thus $c \in cset$. Hence $\langle\sigma, b, \tau\rangle \models receive(c)$ iff $\tau \leq end(\sigma)$ and there exists a $\vartheta \in VAL$ such that $(c, \vartheta) \in \sigma(\tau).R$ iff, by definition, $\tau \leq end([\sigma]^R_{cset})$ and there exists a $\vartheta \in VAL$ such that $(c, \vartheta) \in [\sigma]^R_{cset}(\tau).R$ iff $\langle[\sigma]^R_{cset}, b, \tau\rangle \models receive(c)$.

- $\varphi \equiv \varphi_1 \vee \varphi_2$. For $i = 1, 2$, $ich(\varphi_i) \subseteq (ich(\varphi_1) \cup ich(\varphi_2)) = ich(\varphi) \subseteq cset$. Hence $\langle \sigma, b, \tau \rangle \models \varphi_1 \vee \varphi_2$ iff $\langle \sigma, b, \tau \rangle \models \varphi_1$ or $\langle \sigma, b, \tau \rangle \models \varphi_2$ iff, by the induction hypothesis, $\langle [\sigma]^R_{cset}, b, \tau \rangle \models \varphi_1$ or $\langle [\sigma]^R_{cset}, b, \tau \rangle \models \varphi_2$ iff $\langle [\sigma]^R_{cset}, b, \tau \rangle \models \varphi_1 \vee \varphi_2$.

- $\varphi \equiv \neg\varphi_1$ and $\varphi \equiv \varphi_1 \ \mathcal{U} \ \varphi_2$. Similar to the proof for $\varphi \equiv \varphi_1 \vee \varphi_2$.

- $\varphi \equiv \varphi_1 \ \mathcal{C} \ \varphi_2$. For $i = 1, 2$, $ich(\varphi_i) \subseteq ich(\varphi) \subseteq cset$. Hence $\langle \sigma, b, \tau \rangle \models \varphi_1 \ \mathcal{C} \ \varphi_2$ iff
  - either $\langle \sigma, b, \tau \rangle \models \varphi_1$ and $end(\sigma) = \infty$ iff, by the induction hypothesis, $\langle [\sigma]^R_{cset}, b, \tau \rangle \models \varphi_1$ and $end([\sigma]^R_{cset}) = \infty$ iff $\langle [\sigma]^R_{cset}, b, \tau \rangle \models \varphi_1 \ \mathcal{C} \ \varphi_2$,
  - or there exist models σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $\tau \leq end(\sigma_1) < \infty$, $\langle \sigma_1, b, \tau \rangle \models \varphi_1$, and $\langle \sigma_2, Buf(b, \sigma_1), begin(\sigma_2) \rangle \models \varphi_2$ iff, by the induction hypothesis, there exist models $[\sigma_1]^R_{cset}$ and $[\sigma_2]^R_{cset}$ such that $[\sigma]^R_{cset} = [\sigma_1]^R_{cset}[\sigma_2]^R_{cset}$, $\langle [\sigma_1]^R_{cset}, b, \tau \rangle \models \varphi_1$, and $\langle [\sigma_2]^R_{cset}, Buf(b, \sigma_1), begin(\sigma_2) \rangle \models \varphi_2$ iff, since $ich(\varphi_2) \subseteq cset$ and for any $c \in cset$, $Buf(b, \sigma_1)(c) = Buf(b, [\sigma_1]^R_{cset})(c)$, by lemma 3.5.12, there exist models $[\sigma_1]^R_{cset}$ and $[\sigma_2]^R_{cset}$ such that $[\sigma]^R_{cset} = [\sigma_1]^R_{cset}[\sigma_2]^R_{cset}$, $\langle [\sigma_1]^R_{cset}, b, \tau \rangle \models \varphi_1$, and $\langle [\sigma_2]^R_{cset}, Buf(b, [\sigma_1]^R_{cset}), begin([\sigma_2]^R_{cset}) \rangle \models \varphi_2$ iff $\langle [\sigma]^R_{cset}, b, \tau \rangle \models \varphi_1 \ \mathcal{C} \ \varphi_2$.

- $\varphi \equiv \varphi_1 \ \mathcal{C}^* \ \varphi_2$. Similar to the proof for $\varphi \equiv \varphi_1 \ \mathcal{C} \ \varphi_2$.

Proof of Lemma 3.5.14

For any $vset \subseteq VAR$ and any specification φ, if $var(\varphi) \subseteq vset$, we prove, by induction on φ, that for any model σ, any buffer b, and any $\tau \geq begin(\sigma)$, $\langle \sigma, b, \tau \rangle \models \varphi$ iff $\langle \sigma \downarrow vset, b, \tau \rangle \models \varphi$.

- $\varphi \equiv qexp_1 = qexp_2$. $\langle \sigma, b, \tau \rangle \models qexp_1 = qexp_2$ iff $\mathcal{Q}(qexp_1)(\sigma, b, \tau) = \mathcal{Q}(qexp_2)(\sigma, b, \tau)$ iff, by lemma 3.5.5, $\mathcal{Q}(qexp_1)(\sigma \downarrow vset, b, \tau) = \mathcal{Q}(qexp_2)(\sigma \downarrow vset, b, \tau)$ iff $\langle \sigma \downarrow vset, b, \tau \rangle \models qexp_1 = qexp_2$.

- $\varphi \equiv texp_1 = texp_2$. For $i = 1, 2$, $var(texp_i) \subseteq var(\varphi) \subseteq vset$. Hence $\langle \sigma, b, \tau \rangle \models texp_1 = texp_2$ iff $\mathcal{T}(texp_1)(\sigma, b, \tau) = \mathcal{T}(texp_2)(\sigma, b, \tau)$ iff, by lemma 3.5.11,
  $\mathcal{T}(texp_1)(\sigma \downarrow vset, b, \tau) = \mathcal{T}(texp_2)(\sigma \downarrow vset, b, \tau)$ iff
  $\langle \sigma \downarrow vset, b, \tau \rangle \models texp_1 = texp_2$.

- $\varphi \equiv texp_1 < texp_2$. Similar to the proof for $\varphi \equiv texp_1 = texp_2$.

- $\varphi \equiv send(c, vexp)$. $var(\varphi) = var(vexp)$ and thus $var(vexp) \subseteq vset$. Hence $\langle \sigma, b, \tau \rangle \models send(c, vexp)$ iff $\tau \leq end(\sigma)$ and $(c, \mathcal{V}(vexp)(\sigma, b, \tau)) \in \sigma(\tau).S$ iff,

by definition and lemma 3.5.8, $\tau \leq end(\sigma \downarrow vset)$ and
$(c, \mathcal{V}(vexp)(\sigma \downarrow vset, b, \tau)) \in (\sigma \downarrow vset)(\tau).S$ iff $\langle \sigma \downarrow vset, b, \tau \rangle \models send(c, vexp)$.

- $\varphi \equiv send(c)$. $\langle \sigma, b, \tau \rangle \models send(c)$ iff $\tau \leq end(\sigma)$ and there exists a $\vartheta \in VAL$ such that $(c, \vartheta) \in \sigma(\tau).S$ iff, by definition, $\tau \leq end(\sigma \downarrow vset)$ and there exists a $\vartheta \in VAL$ such that $(c, \vartheta) \in (\sigma \downarrow vset)(\tau).S$ iff $\langle \sigma \downarrow vset, b, \tau \rangle \models send(c)$.

- $\varphi \equiv receive(c, vexp)$. $var(\varphi) = var(vexp)$ and thus $var(vexp) \subseteq vset$. Hence $\langle \sigma, b, \tau \rangle \models receive(c, vexp)$ iff $\tau \leq end(\sigma)$ and $(c, \mathcal{V}(vexp)(\sigma, b, \tau)) \in \sigma(\tau).R$ iff, by definition and lemma 3.5.7, $\tau \leq end(\sigma \downarrow vset)$ and
$(c, \mathcal{V}(vexp)(\sigma \downarrow vset, b, \tau)) \in (\sigma \downarrow vset)(\tau).R$ iff $\langle \sigma \downarrow vset, b, \tau \rangle \models receive(c, vexp)$.

- $\varphi \equiv receive(c)$. $\langle \sigma, b, \tau \rangle \models receive(c)$ iff $\tau \leq end(\sigma)$ and there exists a $\vartheta \in VAL$ such that $(c, \vartheta) \in \sigma(\tau).R$ iff, by definition, $\tau \leq end(\sigma \downarrow vset)$ and there exists a $\vartheta \in VAL$ such that $(c, \vartheta) \in (\sigma \downarrow vset)(\tau).R$ iff $\langle \sigma \downarrow vset, b, \tau \rangle \models receive(c)$.

- $\varphi \equiv \varphi_1 \vee \varphi_2$. For $i = 1, 2$, $var(\varphi_i) \subseteq (var(\varphi_1) \cup var(\varphi_2)) = var(\varphi) \subseteq vset$. Hence $\langle \sigma, b, \tau \rangle \models \varphi_1 \vee \varphi_2$ iff $\langle \sigma, b, \tau \rangle \models \varphi_1$ or $\langle \sigma, b, \tau \rangle \models \varphi_2$ iff, by the induction hypothesis, $\langle \sigma \downarrow vset, b, \tau \rangle \models \varphi_1$ or $\langle \sigma \downarrow vset, b, \tau \rangle \models \varphi_2$ iff $\langle \sigma \downarrow vset, b, \tau \rangle \models \varphi_1 \vee \varphi_2$.

- $\varphi \equiv \neg\varphi_1$ and $\varphi \equiv \varphi_1 \ \mathcal{U} \ \varphi_2$. Similar to the proof for $\varphi \equiv \varphi_1 \vee \varphi_2$.

- $\varphi \equiv \varphi_1 \ \mathcal{C} \ \varphi_2$. For $i = 1, 2$, $var(\varphi_i) \subseteq var(\varphi) \subseteq vset$. Hence $\langle \sigma, b, \tau \rangle \models \varphi_1 \ \mathcal{C} \ \varphi_2$ iff
  - either $\langle \sigma, b, \tau \rangle \models \varphi_1$ and $end(\sigma) = \infty$ iff, by the induction hypothesis, $\langle \sigma \downarrow vset, b, \tau \rangle \models \varphi_1$ and $end(\sigma \downarrow vset) = \infty$ iff $\langle \sigma \downarrow vset, b, \tau \rangle \models \varphi_1 \ \mathcal{C} \ \varphi_2$,
  - or there exist models σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $\tau \leq end(\sigma_1) < \infty$, $\langle \sigma_1, b, \tau \rangle \models \varphi_1$, and $\langle \sigma_2, Buf(b, \sigma_1), begin(\sigma_2) \rangle \models \varphi_2$ iff, by the induction hypothesis, there exist models $\sigma_1 \downarrow vset$ and $\sigma_2 \downarrow vset$ such that $\sigma \downarrow vset = (\sigma_1 \downarrow vset)(\sigma_2 \downarrow vset)$, $\langle \sigma_1 \downarrow vset, b, \tau \rangle \models \varphi_1$, and $\langle \sigma_2 \downarrow vset, Buf(b, \sigma_1), begin(\sigma_2) \rangle \models \varphi_2$ iff, by definition, $Buf(b, \sigma_1) = Buf(b, \sigma_1 \downarrow vset)$, there exist models $\sigma_1 \downarrow vset$ and $\sigma_2 \downarrow vset$ such that $\sigma \downarrow vset = (\sigma_1 \downarrow vset)(\sigma_2 \downarrow vset)$, $\langle \sigma_1 \downarrow vset, b, \tau \rangle \models \varphi_1$, and $\langle \sigma_2 \downarrow vset, Buf(b, \sigma_1 \downarrow vset), begin(\sigma_2 \downarrow vset) \rangle \models \varphi_2$ iff $\langle \sigma \downarrow vset, b, \tau \rangle \models \varphi_1 \ \mathcal{C} \ \varphi_2$.

- $\varphi \equiv \varphi_1 \ \mathcal{C}^* \ \varphi_2$. Similar to the proof for $\varphi \equiv \varphi_1 \ \mathcal{C} \ \varphi_2$.

Lemma 3.5.15, lemma 3.5.16, lemma 3.5.17, and lemma 3.5.18 can be proved similarly as in Appendix A for lemma 2.6.9, lemma 2.6.10, lemma 2.6.11, and lemma 2.6.12, respectively.

Appendix E

Soundness of the Proof System in Chapter 3

To prove the soundness of a proof system, we must show that every axiom in the proof system is indeed valid and every inference rule preserves validity.

To prove that S **sat** φ for some S and φ, we have to show that, for any buffer b and any model $\sigma \in \mathcal{M}(S)(b)$, $\langle \sigma, b, begin(\sigma) \rangle \models \varphi$.

Here we only give the proofs for receiving invariance, send, receive, sequential compostion, and parallel composition. The others can be proved sound silimarly as in Appendix B.

Receiving Invariance

Consider any process S and any channel $c \in cset$ with $cset \subseteq CHAN$ and $cset \cap ich(S) = \emptyset$. We prove that the receiving invariance axiom 3.4.2 is valid.

For any buffer b, any $\sigma \in \mathcal{M}(S)(b)$, by the theorem 3.2.1, we obatin $ich(\sigma) \subseteq ich(S)$ and then $cset \cap ich(\sigma) = \emptyset$. For any $c \in cset$, any $\vartheta \in VAL$, and any τ, $begin(\sigma) \leq \tau \leq end(\sigma)$, by definition, $(c, \vartheta) \notin \sigma(\tau).R$. Thus we obtain $\langle \sigma, b, \tau \rangle \models \neg receive(c)$. For any $\tau' > end(\sigma)$, by definition again, we have $\langle \sigma, b, \tau' \rangle \models \neg receive(c)$. Hence for any $\tau \geq begin(\sigma)$, we have $\langle \sigma, b, \tau \rangle \models \neg receive(c)$, i.e., $\langle \sigma, b, begin(\sigma) \rangle \models \Box \neg receive(c)$. From $c \in cset$, we have $\langle \sigma, b, begin(\sigma) \rangle \models \bigwedge_{c \in cset} \Box \neg receive(c)$, i.e., $\langle \sigma, b, begin(\sigma) \rangle \models \Box \bigwedge_{c \in cset} \neg receive(c)$. Thus we obtain $\langle \sigma, b, begin(\sigma) \rangle \models \Box\, norecv(cset)$ and then axiom 3.4.2 is valid.

Send

We prove that the send axiom 3.4.3 is valid.

For any buffer b and any $\sigma \in \mathcal{M}(c!!e)(b)$, we have $end(\sigma) = begin(\sigma)+K_c$, for any $\sigma' \prec \sigma$, $Idle(\sigma')$, $Nomsg(\sigma', \{c\})$, $\sigma^e.s = \sigma^b.s$, $\sigma^e.R = \emptyset$, and $([\sigma]^S_{\{c\}})^e.S = \{(c, \mathcal{E}(e)(\sigma^b.s))\}$. By definition, we obtain $\langle \sigma, b, begin(\sigma)\rangle \models term = start + K_c$. Furthermore, for any τ, $begin(\sigma) \leq \tau < end(\sigma)$, any $\vartheta \in VAL$, we have $(c, \vartheta) \notin \sigma(\tau).S$, i.e., $\langle \sigma, b, \tau\rangle \models \neg send(c)$. By lemma 3.5.1, we also obtain $\langle \sigma, b, end(\sigma)\rangle \models send(c, e)$. Thus we have $\langle \sigma, b, begin(\sigma)\rangle \models \neg send(c)\ \mathcal{U}\ (T = term = start + K_c \wedge send(c, e))$ and then the axiom 3.4.3 is valid.

Receive

We prove that the receive axiom 3.4.4 is valid.

For any buffer b and any $\sigma \in \mathcal{M}(c??x)(b)$, there exist models σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $\sigma_1 \in WRead(c??x)(b)$, and $\sigma_2 \in Read(c??x)(Buf(b, \sigma_1))$. From $\sigma_1 \in WRead(c??x)(b)$, we obtain $Idle(\sigma_1)$ and thus $\langle \sigma_1, b, begin(\sigma_1)\rangle \models \Box[x = first(x) \wedge \neg receive(c)]$. We also have $Buf(b, \sigma_1')(c) = \langle\rangle$, for any $\sigma_1' \prec \sigma_1$. That is, for any τ, $begin(\sigma_1) \leq \tau < end(\sigma_1)$, and any $\vartheta \in VAL$, $b(c) = \langle\rangle$ and $(c, \vartheta) \notin \sigma_1(\tau).S$. Thus we have $\langle \sigma_1, b, \tau\rangle \models init(c) = \langle\rangle \wedge \neg send(c)$. If $end(\sigma_1) = \infty$, then we obtain $\langle \sigma_1, b, begin(\sigma_1)\rangle \models \Box[init(c) = \langle\rangle \wedge \neg send(c)]$. If $end(\sigma_1) < \infty$, by the semantics, we have $b(c) \neq \langle\rangle$ or $(c, \vartheta) \in \sigma_1^e.S$, for some $\vartheta \in VAL$. Thus we have $\langle \sigma_1, b, end(\sigma_1)\rangle \models T = term \wedge (init(c) \neq \langle\rangle \vee send(c))$. Hence we have $\langle \sigma_1, b, begin(\sigma_1)\rangle \models [init(c) = \langle\rangle \wedge \neg send(c)]\ \mathcal{U}\ [T = term \wedge (init(c) \neq \langle\rangle \vee send(c))]$. Thus we obtain $\langle \sigma_1, b, begin(\sigma_1)\rangle \models Await[init(c) \neq \langle\rangle \vee send(c)]$ and thus $\langle \sigma_1, b, begin(\sigma_1)\rangle \models WRecv(c??x)$.

Let $b' \equiv Buf(b, \sigma_1)$. From $\sigma_2 \in Read(c??x)(Buf(b, \sigma_1))$, i.e., $\sigma_2 \in Read(c??x)(b')$, we obtain $end(\sigma_2) = begin(\sigma_2) + K_c$, for any $\sigma_2' \prec \sigma_2$, $Idle(\sigma_2')$, $\sigma_2^e.R = \{(c, First(b'(c)))\}$, and $\sigma_2^e.s = (\sigma_2^b.s : x \to First(b'(c)))$. Thus, for any τ, $begin(\sigma_2) \leq \tau < end(\sigma_2)$, we have $\sigma_2(\tau).s = \sigma_2^b.s$ and $\sigma_2(\tau).R = \emptyset$. We also have $\sigma_2^e.s(x) = First(b'(c))$. Then we obtain $\langle \sigma_2, b', end(\sigma_2)\rangle \models receive(c, x) \wedge x = first(init(c))$. Hence we have $\langle \sigma_2, b', begin(\sigma_2)\rangle \models [x = first(x) \wedge \neg receive(c)]\ \mathcal{U}\ [T = term = start + K_c \wedge receive(c, x) \wedge x = first(init(c))]$, i.e., $\langle \sigma_2, Buf(b, \sigma_1), begin(\sigma_2)\rangle \models Recv(c??x)$.

Since $\sigma = \sigma_1\sigma_2$, by the definition of the $\mathcal{C}$ operator, we obtain
$\langle \sigma, b, begin(\sigma)\rangle \models WRecv(c??x)\ \mathcal{C}\ Recv(c??x)$. Hence the receive axiom 3.4.4 is valid.

Sequential Composition

We prove that the sequential composition rule 3.4.1 preserves validity.

Assume that S_1 **sat** φ_1 and S_2 **sat** φ_2 are valid. Let $\psi_1 \equiv \Box\, nosend(och(S_2) \setminus och(S_1))$ and $\psi_2 \equiv \Box\, nosend(och(S_1) \setminus och(S_2))$. We show that $S_1; S_2$ **sat** $(\varphi_1 \wedge \psi_1)\ \mathcal{C}\ (\varphi_2 \wedge \psi_2)$ is also valid.

For any buffer b, consider any $\sigma \in \mathcal{M}(S_1; S_2)(b)$. Then there exist σ_1 and σ_2 such that $\sigma = \sigma_1\sigma_2$, $\sigma_1 \in \mathcal{M}(S_1)(b)$, $\sigma_2 \in \mathcal{M}(S_2)(Buf(b, \sigma_1))$, and $Agree(\sigma_1, \sigma_2, S_1, S_2)$. By definition, $Agree(\sigma_1, \sigma_2, S_1, S_2) \equiv Nomsg(\sigma_1, och(S_2) \setminus och(S_1)) \wedge Nomsg(\sigma_2, och(S_1) \setminus och(S_2))$. From $Nomsg(\sigma_1, och(S_2) \setminus och(S_1))$, we have, for any τ, $begin(\sigma_1) \leq \tau \leq end(\sigma_1)$, any $c \in och(S_2) \setminus och(S_1)$, and any $\vartheta \in VAL$, $(c, \vartheta) \notin \sigma_1(\tau).S$. Thus we obtain $\langle \sigma_1, b, \tau \rangle \models \neg send(c)$. For any $\tau' > end(\sigma_1)$, by definition, we also have $\langle \sigma_1, b, \tau' \rangle \models \neg send(c)$. Then we obtain $\langle \sigma_1, b, begin(\sigma_1) \rangle \models \Box \neg send(c)$. Since $c \in och(S_2) \setminus och(S_1)$, we have $\langle \sigma_1, b, begin(\sigma_1) \rangle \models \bigwedge_{c \in och(S_2) \setminus och(S_1)} \Box \neg send(c)$, i.e., $\langle \sigma_1, b, begin(\sigma_1) \rangle \models \Box \bigwedge_{c \in och(S_2) \setminus och(S_1)} \neg send(c)$. Hence we obtain $\langle \sigma_1, b, begin(\sigma_1) \rangle \models \Box\, nosend(och(S_2) \setminus och(S_1))$ and then $\langle \sigma_1, b, begin(\sigma_1) \rangle \models \psi_1$. From S_1 **sat** φ_1, we obtain $\langle \sigma_1, b, begin(\sigma_1) \rangle \models \varphi_1$. Thus we have $\langle \sigma_1, b, begin(\sigma_1) \rangle \models \varphi_1 \wedge \psi_1$. Similarly, we can derive $\langle \sigma_2, Buf(b, \sigma_1), begin(\sigma_2) \rangle \models \varphi_2 \wedge \psi_2$. By the definition of the $\mathcal{C}$ operator, we have $\langle \sigma_1\sigma_2, b, begin(\sigma_1) \rangle \models (\varphi_1 \wedge \psi_1)\ \mathcal{C}\ (\varphi_2 \wedge \psi_2)$, i.e., $\langle \sigma, b, begin(\sigma) \rangle \models (\varphi_1 \wedge \psi_1)\ \mathcal{C}\ (\varphi_2 \wedge \psi_2)$. Hence the rule 3.4.1 preserves validity.

Parallel Composition

Assume S_i **sat** φ_i, $IBuf \equiv \bigwedge_{c \in ch(S_1) \cap ch(S_2)} init(c) = \langle\rangle$, $\psi_i \equiv \Box\, [inv(var(S_i)) \wedge norecv(ich(S_i)) \wedge nosend(och(S_i))]$, $ich(\varphi_i) \subseteq ich(S_i)$, and $var(\varphi_i) \subseteq var(S_i)$, for $i = 1, 2$. We show the validity of $S_1 \| S_2$ **sat** $IBuf \wedge [(\varphi_1 \wedge (\varphi_2\ \mathcal{C}\ \psi_2)) \vee (\varphi_2 \wedge (\varphi_1\ \mathcal{C}\ \psi_1))]$.

For any buffer b, consider any $\sigma \in \mathcal{M}(S_1 \| S_2)(b)$. Then $ich(\sigma) \subseteq ich(S_1) \cup ich(S_2)$, and for $i \in \{1, 2\}$, there exist $\sigma_i \in \mathcal{M}(S_i)(b)$ such that $begin(\sigma) = begin(\sigma_1) = begin(\sigma_2)$, $end(\sigma) = max(end(\sigma_1), end(\sigma_2))$, for any $c \in ch(S_1) \cap ch(S_2)$, $b(c) = \langle\rangle$. By definition, we have $\langle \sigma, b, begin(\sigma) \rangle \models IBuf$. Suppose $end(\sigma_1) \geq end(\sigma_2)$. Then $end(\sigma) = end(\sigma_1)$. We prove $\langle \sigma, b, begin(\sigma) \rangle \models \varphi_1 \wedge (\varphi_2\ \mathcal{C}\ \psi_2)$.

- First we prove $\langle \sigma, b, begin(\sigma) \rangle \models \varphi_1$. From the semantics, we have that, for any τ, $begin(\sigma_1) \leq \tau \leq end(\sigma_1)$, $[\sigma \downarrow var(S_1)]^R_{ich(S_1)}(\tau).S = \sigma(\tau).S = \sigma_1(\tau).S$, $[\sigma \downarrow var(S_1)]^R_{ich(S_1)}(\tau).R = [\sigma]^R_{ich(S_1)}(\tau).R = \sigma_1(\tau).R$, $[\sigma \downarrow var(S_1)]^R_{ich(S_1)}(\tau).s = (\sigma \downarrow var(S_1))(\tau).s = \sigma_1(\tau).s$. Since $begin([\sigma \downarrow var(S_1)]^R_{ich(S_1)}) = begin(\sigma) = begin(\sigma_1)$, $end([\sigma \downarrow var(S_1)]^R_{ich(S_1)}) = end(\sigma) = end(\sigma_1)$, we obtain $[\sigma \downarrow var(S_1)]^R_{ich(S_1)} = \sigma_1$. Since $\sigma_1 \in \mathcal{M}(S_1)(b)$ and S_1 **sat** φ_1, we have $\langle [\sigma \downarrow var(S_1)]^R_{ich(S_1)}, b, begin(\sigma) \rangle \models \varphi_1$. Since $ich(\varphi_1) \subseteq ich(S_1)$ and $var(\varphi_1) \subseteq var(S_1)$, lemma 3.5.13 and lemma 3.5.14 lead to $\langle \sigma, b, begin(\sigma) \rangle \models \varphi_1$.

- Next we prove $\langle \sigma, b, begin(\sigma) \rangle \models \varphi_2\ \mathcal{C}\ \psi_2$.

- If $end(\sigma_2) = \infty$, since $end(\sigma) = end(\sigma_1) \geq end(\sigma_2)$, we have $end(\sigma_2) = end(\sigma) = \infty$. Similarly, we can derive $\langle \sigma, b, begin(\sigma) \rangle \models \varphi_2$. By the definition of the $\mathcal{C}$ operator, we obtain $\langle \sigma, b, begin(\sigma) \rangle \models \varphi_2 \ \mathcal{C} \ \psi_2$;

- If $end(\sigma_2) < \infty$, from S_2 **sat** φ_2 and $\sigma_2 \in \mathcal{M}(S_2)(b)$, we obtain $\langle \sigma_2, b, begin(\sigma_2) \rangle \models \varphi_2$. We define a model σ_3 such that $begin(\sigma_3) = end(\sigma_2)$, $end(\sigma_3) = end(\sigma)$, for any τ, $begin(\sigma_3) < \tau \leq end(\sigma_3)$, $\sigma_3(\tau).s = \sigma_2^e.s$, $\sigma_3(\tau).R = [\sigma]^R_{ich(S_2)}(\tau).R$, $\sigma_3(\tau).S = \sigma_1(\tau).S$, $\sigma_3^b.s = \sigma_2^e.s$, $\sigma_3^b.R = ([\sigma]^R_{ich(S_2)})^b.R$, and for any $c \in och(S_2)$, any $\vartheta \in VAL$, $(c, \vartheta) \notin \sigma_3^b.S$. By the semantics, $[\sigma]^R_{ich(S_2)}(\tau).R = \emptyset$ and thus $\sigma_3(\tau).R = \emptyset$. Since $end(\sigma_2) \leq end(\sigma_1)$, by $Cons(\sigma_1, \sigma_2, S_1, S_2)$, for any τ', $end(\sigma_2) < \tau' \leq end(\sigma_1)$, any $c \in och(S_2)$, and any $\vartheta \in VAL$, $(c, \vartheta) \notin \sigma_1(\tau').S$. That is, for any τ, $begin(\sigma_3) < \tau \leq end(\sigma_3)$, $(c, \vartheta) \notin \sigma_3(\tau).S$. Then we obtain
  $\langle \sigma_3, Buf(b, \sigma_2), \tau \rangle \models inv(var(S_2)) \wedge norecv(ich(S_2)) \wedge nosend(och(S_2))$.
  For any $\tau' > end(\sigma_3)$, we also have
  $\langle \sigma_3, Buf(b, \sigma_2), \tau' \rangle \models inv(var(S_2)) \wedge norecv(ich(S_2)) \wedge nosend(och(S_2))$.
  Thus we obtain
  $\langle \sigma_3, Buf(b, \sigma_2), begin(\sigma_3) \rangle \models \Box\, [inv(var(S_2)) \wedge norecv(ich(S_2)) \wedge nosend(och(S_2))]$, i.e., $\langle \sigma_3, Buf(b, \sigma_2), begin(\sigma_3) \rangle \models \psi_2$. By the $\mathcal{C}$ operator, we obtain $\langle \sigma_2\sigma_3, b, begin(\sigma_2) \rangle \models \varphi_2 \ \mathcal{C} \ \psi_2$.

  Now we prove $\sigma_2\sigma_3 = [\sigma \downarrow var(S_2)]^R_{ich(S_2)}$. Let $\bar{\sigma} \equiv [\sigma \downarrow var(S_2)]^R_{ich(S_2)}$.
  By definition,

$$\bar{\sigma}(\tau).s = (\sigma \downarrow var(S_2))(\tau).s = \begin{cases} \sigma_2(\tau).s & begin(\sigma_2) \leq \tau \leq end(\sigma_2) \\ \sigma_3(\tau).s & end(\sigma_2) < \tau \leq end(\sigma) \end{cases}$$

$$\bar{\sigma}(\tau).R = [\sigma]^R_{ich(S_2)}(\tau).R = \begin{cases} \sigma_2(\tau).R & begin(\sigma_2) \leq \tau \leq end(\sigma_2) \\ \sigma_3(\tau).R & end(\sigma_2) < \tau \leq end(\sigma) \end{cases}$$

$$\bar{\sigma}(\tau).S = \sigma(\tau).S = \begin{cases} \sigma_2(\tau).S & begin(\sigma_2) \leq \tau \leq end(\sigma_2) \\ \sigma_1(\tau).S = \sigma_3(\tau).S & end(\sigma_2) < \tau \leq end(\sigma) \end{cases}$$

  Hence $\bar{\sigma} = \sigma_2\sigma_3$ and then we have $\langle \bar{\sigma}, b, begin(\sigma) \rangle \models \varphi_2 \ \mathcal{C} \ \psi_2$. Since $ich(\varphi_2) \cup ich(\psi_2) \subseteq ich(S_2)$ and $var(\varphi_2) \cup var(\psi_2) \subseteq var(S_2)$, by lemma 3.5.13 and lemma 3.5.14, we obtain $\langle \sigma, b, begin(\sigma) \rangle \models \varphi_2 \ \mathcal{C} \ \psi_2$.

Hence we have proved $\langle \sigma, b, begin(\sigma) \rangle \models \varphi_1 \wedge (\varphi_2 \ \mathcal{C} \ \psi_2)$.
Similarly, for $end(\sigma_1) < end(\sigma_2)$, we can show $\langle \sigma, b, begin(\sigma) \rangle \models \varphi_2 \wedge (\varphi_1 \ \mathcal{C} \ \psi_1)$.
Thus the parallel composition rule 3.4.5 preserves validity.

Appendix F

Precise Specifications for Statements in Chapter 3

The preciseness theorem 3.5.2 can be proved similarly as in Appendix C for the theorem 2.6.2. Here we only give a precise specification for each statement from the programming language in section 3.1.

The precise specifications for skip, assignment, and delay statements are the same as those given in Appendix C, respectively.

Send

A precise specification for statement $c!!e$ is
$\neg send(c)\ \mathcal{U}\ (T = term = start + K_c \wedge send(c, e))$.

To prove that this is a precise specification for $c!!e$, we need to use the general assumption on the S-fields of a model which is given in section 3.2.2.

Receive

A precise specification for statement $c??x$ is $WRecv(c??x)\ \mathcal{C}\ \ Recv(c??x)$ with

$WRecv(c??x) \equiv \Box\,[x = first(x) \wedge \neg receive(c)] \wedge Await[init(c) \neq \langle\rangle \vee send(c)]$

and

$$Recv(c??x) \equiv [x = first(x) \wedge \neg receive(c)]\ \ \mathcal{U}$$
$$[T = term = start + K_c \wedge receive(c, x) \wedge x = first(init(c))].$$

Sequential Composition

Assume that φ_i is precise for S_i, for $i = 1, 2$. A precise specification for $S_1; S_2$ is

$$[\varphi_1 \wedge \Box\,(inv(wvar(S_1;S_2) \setminus wvar(S_1)) \wedge norecv(ich(S_2) \setminus ich(S_1)) \wedge$$
$$nosend(och(S_2) \setminus och(S_1)))]\ \mathcal{C}$$
$$[\varphi_2 \wedge \Box\,(inv(wvar(S_1;S_2) \setminus wvar(S_2)) \wedge norecv(ich(S_1) \setminus ich(S_2)) \wedge$$
$$nosend(och(S_1) \setminus och(S_2)))].$$

Guarded Command with Purely Boolean Guards

Assume that φ_i is precise for S_i, for $i = 1, \ldots, n$.
A precise specification for $G_1 \equiv [\,[]_{i=1}^{n} g_i \rightarrow S_i]$ is

$$[Quiet(G_1)\ \mathcal{U}\ (T = start + K_g \wedge Quiet(G_1))] \wedge [\neg\bar{g} \rightarrow Eval] \wedge$$
$$[\bar{g} \rightarrow (Eval\ \mathcal{C}\ \ \textstyle\bigvee_{i=1}^{n} g_i \wedge \varphi_i \wedge \Box\,(inv(wvar(G_1) \setminus wvar(S_i)) \wedge norecv(ich(G_1) \setminus ich(S_i)) \wedge$$
$$nosend(och(G_1) \setminus och(S_i))))].$$

Guarded Command with IO-Guards

Assume that φ_0 is precise for S_0 and φ_i is precise for $c_i??x_i; S_i$, for $i = 1, \ldots, n$.
A precise specification for $G_2 \equiv [\,[]_{i=1}^{n} g_i; c_i??x_i \rightarrow S_i\,[]\, g_0; \textbf{delay}\ e \rightarrow S_0]$ is

$$[Quiet(G_2)\ \mathcal{U}\ (T = start + K_g \wedge Quiet(G_2))] \wedge [\neg\bar{g} \rightarrow Eval] \wedge$$
$$[\bar{g} \rightarrow (Eval\ \mathcal{C}\ \ (NFinComm\ \vee\ NTimeOut\ \vee\ NAnyComm))]$$

where

$$NFinComm \equiv (g_0 \wedge term < start + max(0,e) \wedge Wait)\ \mathcal{C}\ \ NComm$$
$$NComm \equiv \textstyle\bigvee_{i=1}^{n} g_i \wedge \varphi_i \wedge \Box\,(inv(wvar(G_2) \setminus wvar(c_i??x_i; S_i)) \wedge$$
$$norecv(ich(G_2) \setminus ich(c_i??x_i; S_i)) \wedge nosend(och(G_2) \setminus och(c_i??x_i; S_i)))$$
$$NTimeOut \equiv [g_0 \wedge \Box\,(\textstyle\bigwedge_{c_i \in \bar{c}} init(c_i) = \langle\rangle \wedge \neg send(c_i)) \wedge term = start + max(0,e) \wedge$$
$$\Box\,Quiet(G_2)]\ \mathcal{C}$$
$$[\varphi_0 \wedge \Box\,(inv(wvar(G_2) \setminus wvar(S_0)) \wedge norecv(ich(G_2) \setminus ich(S_0)) \wedge$$
$$nosend(och(G_2) \setminus och(S_0)))]$$
$$NAnyComm \equiv (\neg g_0 \wedge Wait)\ \mathcal{C}\ \ NComm$$

Iteration

Assume that φ is precise for G. A precise specification for $\star G$ is $(\bar{g} \wedge \varphi)\ \mathcal{C}^*\ (\neg\bar{g} \wedge \varphi)$.

Parallel Composition

Assume that φ_i is precise for S_i, for $i = 1, 2$. A precise specification for $S_1 \| S_2$ is
$IBuf \wedge [(\varphi_1 \wedge (\varphi_2 \;\mathcal{C}\; \psi_2)) \vee (\varphi_2 \wedge (\varphi_1 \;\mathcal{C}\; \psi_1))]$,
where
$IBuf \equiv \bigwedge_{c \in ch(S_1) \cap ch(S_2)} init(c) = \langle\rangle$,
$\psi_i \equiv \Box\, [inv(var(S_i)) \wedge norecv(ich(S_i)) \wedge nosend(och(S_i))]$, for $i = 1, 2$.

Bibliography

[Ada83] LNCS 155. *The Programming Language Ada, Reference Manual*, 1983.

[AH89] R. Alur and T.A. Henzinger. A really temporal logic. In *Proc. of the 30th Annual Symposium on Foundations of Computer Science*, pages 164–169. IEEE Computer Society Press, 1989.

[AH92] R. Alur and T.A. Henzinger. Logics and models of real-time: A survey. In *Real-Time: Theory in Practice*, pages 74–106. J.W. de Bakker, C. Huizing, W.-P. de Roever, and G. Rozenberg (Eds.), LNCS 600, 1992.

[AL90] M. Abadi and L. Lamport. Composing specifications. In *Stepwise Refinement of Distributed Systems*, pages 1–41. J.W. de Bakker, W.-P. de Roever, and G. Rozenberg (Eds.), LNCS 430, Springer-Verlag, 1990.

[AL92] M. Abadi and L. Lamport. An old-fashioned recipe for real time. In *Real-Time: Theory in Practice*, pages 1–27. J.W. de Bakker, C. Huizing, W.-P. de Roever, and G. Rozenberg (Eds.), LNCS 600, 1992.

[Apt81] K.R. Apt. Ten years of hoare's logic: A survey—part I. *ACM Transactions on Programming Languages and Systems*, 3:431–483, 1981.

[Bak80] J. de Bakker. *Mathematical Theory of Program Correctness.* Prentice Hall, 1980.

[BB91] J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Formal Aspects of Computing 3(2)*, pages 142–188, 1991.

[BB92] J.C.M. Baeten and J.A. Bergstra. Asynchronous communication in real space process algebra. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 473–492. J. Vytopil (Ed.), LNCS 571, Springer-Verlag, 1992.

[BCG92] F.S. de Boer, J. Coenen, and R. Gerth. Exception handling in process algebra. In *Proc. of the North American Process Algebra Workshop*, 1992.

[BD85] O. Babaoglu and R. Drumond. Streets of byzantium: Network architectures for fast reliable broadcast. *IEEE Transactions on Software Engineering 11(6)*, 1985.

[BH92] F. de Boer and J. Hooman. The real-time behaviour of asynchronously communicating processes. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 451–472. J. Vytopil (Ed.), LNCS 571, Springer-Verlag, 1992.

[BHRR91] J.W. de Bakker, C. Huizing, W.-P. de Roever, and G. Rozenberg(Eds.). *Real-Time: Theory in Practice.* LNCS 600, Springer-Verlag, 1991.

[BJ87] K. Birman and T. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems 5(1)*, pages 47–76, 1987.

[BK84] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information & Control*, 60:109–137, 1984.

[BKP84] H. Barringer, R. Kuiper, and A. Pnueli. Now you may compose temporal logic specifications. In *Proc. of the 16th Annual ACM Symposium on Theory of Computing*, pages 51–63, 1984.

[BKP85] H. Barringer, R. Kuiper, and A. Pnueli. A compositional temporal approach to a CSP-like language. In *Formal Models in Programming*, pages 207–227. E.J. Neuhold and G. Chroust (Eds.), 1985.

[BKT85] J.A. Bergstra, J.W. Klop, and J.V. Tucker. Process algebra with asynchronous communication mechanisms. In *Seminar on Concurrency*, pages 76–95. S.D. Brookes, A.W. Roscoe, and G. Winskel (Eds.), LNCS 197, Springer-Verlag, 1985.

[CAS86] F. Cristian, H. Aghili, and R. Strong. Approximate clock synchronization despite omission and performance failures and processor joins. In *The 16th International Symposium on Fault-Tolerant Computing.* Wien, Austrian, 1986.

[CASD85] F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to Byzantine agreement. In *The 15th Annual International Symposium on Fault-Tolerant Computing*, pages 200 – 206. Ann Arbor, USA, 1985.

[CASD89] F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to Byzantine agreement. Research Report RJ 5244, IBM Almaden Research Center, 1989.

[CH92] J. Coenen and J. Hooman. A compositional semantics for fault-tolerant real-time systems. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 33–51. J. Vytopil (Ed.), LNCS 571, Springer-Verlag, 1992.

[CHR91] Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40:269 - 276, 1991.

[CHRR92] Zhou Chaochen, M.R. Hansen, A.P. Ravn, and H. Rischel. Duration specifications for shared processors. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 21–32. J. Vytopil (Ed.), LNCS 571, Springer-Verlag, 1992.

[CM84] J.M. Chang and N. Maxemchuck. Reliable broadcast protocols. *ACM Transactions on Computer Systems 2(3)*, pages 251–273, 1984.

[Coe92] J. Coenen. Specifying fault tolerant programs in deontic logic. In *Proc. of the First International Workshop on Deontic Logic in Computer Science*, 1992.

[Cri85] F. Cristian. A rigorous approach to fault-tolerant programming. *IEEE Trans. on Software Engineering SE-11(1)*, pages 23–31, 1985.

[Cri90] F. Cristian. Synchronous atomic broadcast for redundant broadcast channels. *The Journal of Real-Time Systems 2*, pages 195–212, 1990.

[Cri93] F. Cristian. Comments. *Private Correspondence*, 1993.

[CW92] P. Coesmans and M.J. Wieczorek. Formal specification of fault-tolerant real-time systems using minimal 3-sorted modal logic. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 571–590. J. Vytopil (Ed.), LNCS 571, Springer-Verlag, 1992.

[Dij76] E.W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[DS89] J. Davies and S. Schneider. Factorizing proofs in timed CSP. In *Mathematical Foundations of Programming Semantics*, pages 129–159. M. Main, A. Melton, M. Mislove, and D. Schmidt (Eds.), LNCS 442, Springer-Verlag, 1989.

[Ger84] R. Gerth. Transition logic: How to reason about temporal properties in a compositional way. In *Proc. of the 16th Annual ACM Symposium on Theory of Computing*, 1984.

[GJ88] A. Goswami and M. Joseph. Semantics of real-time distributed programs. In *Formal Techniques in Real-Time and Fault-Tolerant Systems.* M. Joseph (Ed.), LNCS 331, 1988.

[Gou88] Ronald Gould. *Graph Theory.* The Benjamin/Cummings Publishing Company, Inc., 1988.

[Har88] E. Harel. Temporal analysis of real-time systems. Master's thesis, The Weizmann Institute of Science, Rehovot, Israel, 1988.

[Hen91] T.A. Henzinger. *The Temporal Specification and Verification of Real-Time Systems.* PhD thesis, Stanford University, 1991.

[HGR87] C. Huizing, R. Gerth, and W.-P. de Roever. Full abstraction of a real-time denotational semantics for an OCCAM-like language. In *Proc. of the 14th ACM Symposium on Principles of Programming Languages*, pages 223–237, 1987.

[HKZ91] J. Hooman, R. Kuiper, and P. Zhou. A compositional proof system for real-time systems based on explicit clock temporal logic. In *Proc. of the 6th International Workshop on Software Specification and Design*, pages 110–117. IEEE Computer Society Press, 1991.

[HLP90] E. Harel, O. Lichtenstein, and A. Pnueli. Explicit clock temporal logic. In *Proc. Symposium on Logic in Computer Science*, pages 402–413, 1990.

[HMP92] T.A. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In *Real-Time: Theory in Practice*, pages 226–251. J.W. de Bakker, C. Huizing, W.-P. de Roever, and G. Rozenberg (Ed.), LNCS 600, Springer-Verlag, 1992.

[HO83] B.T. Hailpern and S.S. Owicki. Modular verification of computer communication protocols. *IEEE Transactions on Communications*, COM-31(1):56–68, 1983.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[Hoo91] J. Hooman. *Specification and Compositional Verification of Real-Time Systems.* LNCS 558, Springer-Verlag, 1991.

[Hoo93] J. Hooman. Compositional verification of a distributed real-time arbitration protocol. *Real-Time Systems*, to appear, 1993.

[JJH90] He Jifeng, M.B. Josephs, and C.A.R. Hoare. A theory of synchrony and asynchrony. In *Proc. of IFIP Working Conference on Programming Concepts and Methods*, pages 459–478, 1990.

[JMS87] M. Joseph, A. Moitra, and N. Soundararajan. Proof rules for fault tolerant distributed programs. *Science of Computer Programming 8*, pages 43–67, 1987.

[Jon80] C.B. Jones. *Software Development A Rigorous Approach.* Prentice Hall, 1980.

[Jon85] B. Jonsson. A model and proof system for asynchronous networks. In *Proc. of the 4th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 49–58, 1985.

[Jon90] C.B. Jones. *Systematic Software Development using VDM.* Prentice Hall, 1990.

[Koy92] R. Koymans. *Specifying Message Passing and Time-Critical Systems with Temporal Logic.* LNCS 651, Springer-Verlag, 1992.

[KSR+88] R. Koymans, R.K. Shyamasundar, W.-P. de Roever, R. Gerth, and S. Arun-Kumar. Compositional semantics for real-time distributed computing. *Information and Computation*, 79(3):210–256, 1988.

[KVR83] R. Koymans, J. Vytopyl, and W.-P. de Roever. Real-time programming and asynchronous message passing. In *Proc. of the 2nd ACM Symposium on Principles of Distributed Computing*, pages 187–197, 1983.

[LA90] P.A. Lee and T. Anderson. *Fault Tolerance - Principles and Practice.* Springer-Verlag, 1990.

[Lam83a] L. Lamport. Specifying concurrent program modules. *ACM Transactions on Programming Languages and Systems*, 5(2):190–222, 1983.

[Lam83b] L. Lamport. What good is temporal logic. In *Information Processing*, pages 657–668. R.E. Manson (Ed.), North Holland, 1983.

[Lar90] K.G. Larsen. Compositional theories based on an operational semantics of contexts. In *Stepwise Refinement of Distributed Systems*, pages 487–518. J.W. de Bakker, W.-P. de Roever, and G. Rozenberg (Eds.), LNCS 430, Springer-Verlag, 1990.

[Mil83] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25, 1983.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[MP82] Z. Manna and A. Pnueli. Verification of concurrent programs: a temporal proof system. In *Foundations of Computer Science IV, Distributed Systems: Part 2*, volume 159 of *Mathematical Centre Tracts*, pages 163–255, 1982.

[MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer, Berlin, 1991.

[MT90] F. Moller and C. Tofts. A temporal calculus of communicating systems. In *CONCUR '90*, pages 401–415. J.C.M. Baeten and J.W. Klop (Eds), LNCS 458, Springer-Verlag, 1990.

[NDGO86] V. Nguyen, A. Demers, D. Gries, and S. Owicki. A model and temporal proof system for networks of processes. *Distributed Computing*, 1(1):7–25, 1986.

[NRSV90] X. Nicollin, J.-L. Richier, J. Sifakis, and J. Voiron. ATP: an algebra for timed processes. In *Programming Concepts and Methods*, pages 415–442. M. Broy and C.B. Jones (Eds.), 1990.

[Occ88] INMOS Limited. OCCAM *2 Reference Manual*, 1988.

[OL82] S. Owicki and L. Lamport. Proving liveness poperties of concurrent programs. *ACM Transactions on Programming Languages and Systems*, 4(3):455–495, 1982.

[Ost89] J. Ostroff. *Temporal Logic for Real-Time Systems.* Advanced Software Development Series. Research Studies Press, 1989.

[Pel91] J. Peleska. Design and verification of fault tolerant systems with CSP. *Distributed Computing 5*, pages 95–106, 1991.

[PH88] A. Pnueli and E. Harel. Applications of temporal logic to the specification of real-time systems. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 84–98. M. Joseph (Ed.), LNCS 331, 1988.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. of the 18th Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[PWT90] P. Place, W. Wood, and M. Tudball. Survey of formal specification techniques for reactive systems. Technical Report CMU/SEI-90-TR-5, Software Engineering Institute, Carnegie-Mellon University, 1990.

[Ree89] G.M. Reed. A hierarchy of domains for real-time distributed computing. In *Mathematical Foundations of Programming Language Semantics*, pages 80–128. M. Main, A. Melton, M. Mislove, and D. Schmidt (Eds.), LNCS 442, Springer-Verlag, 1989.

[RH91] J. Rushby and F. von Henke. Formal verification of algorithms for critical systems. *ACM SIGSOFT Engineering Notes*, 16(5):1–15, 1991.

[RLT78] B. Randell, P.A. Lee, and P.C. Treleaven. Reliability issues in computing system design. *ACM Computing Surveys*, 10(2):123–165, 1978.

[Roe85] W.-P. de Roever. The quest for compositionality - a survey of assertion-based proof systems for concurrent programs, Part I: concurrency based on shared variables. In *Proc. of the IFIP Working Conference 1985: The role of abstract models in computer science*, pages 181–207. North-Holland, 1985.

[RP86] R. Rosner and A. Pnueli. A choppy logic. In *Proc. Symposium on Logic in Computer Science*, pages 306–313, 1986.

[RR86] G. Reed and A. Roscoe. A timed model for Communicating Sequential Processes. In *Proc. of ICALP '86: Automata, Languages, and Programming*, pages 314–323. LNCS 226, Springer-Verlag, 1986.

[SC93] H. Schepers and J. Coenen. Trace-based compositional refinement of fault tolerant distributed systems. In *Proc. of the 4th IFIP Working Conference on Dependable Computing for Critical Applications*. to appear, 1993.

[SH93] H. Schepers and J. Hooman. Trace-based compositional reasoning about fault tolerant systems. In *PARLE: Parallel Architectures and Languages Europe*. Springer-Verlag, 1993.

[Sha92] N. Shankar. Mechanical verification of a generalized protocol for Byzantine fault tolerant clock synchronization. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 217–236. J. Vytopil(Ed.), LNCS 571, Springer-Verlag, 1992.

[SM81] A. Salwicki and T. Muldner. On the algorithmic properties of concurrent programs. In *Logic of Programs*, pages 169–197. E. Engeler (Ed.), LNCS 125, 1981.

[Yi91] Wang Yi. CCS + time = an interleaving model for real time systems. In *Automata, Languages and Programming*, pages 217–228. J. Leach Albert, B. Monien, and M. Rodriguez (Eds.), LNCS 510, Springer-Verlag, 1991.

[Yod92] V. Yodaiken. Verification of a reliable net protocol. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 193–215. J. Vytopil (Ed.), LNCS 571, Springer-Verlag, 1992.

[ZH92] P. Zhou and J. Hooman. A proof theory for asynchronously communicating real-time systems. In *Proc. of the 13th IEEE Real-Time Systems*, pages 177–186. IEEE Computer Society Press, 1992.

[ZH93a] P. Zhou and J. Hooman. Formal specification and compositional verification of an atomic broadcast protocol. *submitted*, 1993.

[ZH93b] P. Zhou and J. Hooman. Specification and verification of an atomic broadcast protocol. In *Proc. of the 4th IFIP Working Conference on Dependable Computing for Critical Applications*. to appear, 1993.

[ZHK93] P. Zhou, J. Hooman, and R. Kuiper. Compositional verification of real-time systems with explicit clock temporal logic. *Formal Aspects of Computing, to appear*, 1993.

[Zwi89] J. Zwiers. *Compositionality, Concurrency and Partial Correctness.* LNCS 321, Springer-Verlag, 1989.

Samenvatting

In dit proefschrift onderzoeken we formalismen waarin de correctheid van real-time en fout-tolerante systemen bewezen kan worden. Real-time systemen worden gekarakteriseerd door quantitatieve tijdseisen betreffende het optreden van gebeurtenissen. Typische voorbeelden van zulke systemen zijn te vinden in nucleaire energie centrales, industrieële procesbesturing en vliegtuig systemen. De correctheid van deze real-time systemen hangt niet alleen af van hun functionele gedrag maar ook van hun timing. Gezien de complexiteit van veel real-time systemen is het niet eenvoudig om te garanderen dat aan hun functionele en timing eisen is voldaan. Nog moeilijker is het om correctheid te garanderen als componenten kunnen falen. In real-time systemen worden vaak fout-tolerante technieken toegepast om een zekere service te kunnen blijven leveren bij het optreden van fouten. Technieken om fout-tolerantie te bereiken zijn in het algemeen gebaseerd op het efficiënt benutten van redundantie. De introductie van redundantie zal echter het tijdsgedrag van een systeem beïnvloeden. Dit wijst op een sterke relatie tussen real-time en fout-tolerantie.

Om het ontwerpen van een real-time en fout-tolerant systeem te formaliseren is een specificatietaal een eerste vereiste. Zo'n taal moet in staat zijn de eisen van een systeem precies te beschrijven. Een formele beschrijving van de eisen wordt een specificatie genoemd. Een mogelijke aanpak voor het verifiëren dat een programma aan een specificatie voldoet is het ontwerpen van een bewijssysteem bestaande uit axioma's en afleidingsregels. In dit proefschrift ligt de nadruk op het ontwerpen van bewijssystemen die compositioneel zijn. Een compositioneel bewijssysteem stelt ons in staat een systeem te verifiëren door alleen de specificaties van de componenten te gebruiken, zonder kennis van hun interne structuur, en zo te abstraheren van hun implementatie.

Dit proefschrift bestaat ruwweg uit twee delen die hieronder beschreven worden.

Real-Time Formalismen

Om een compositioneel bewijssysteem te ontwikkelen beschouwen we twee versies van een real-time programmeertaal waarin parallelle processen communiceren door middel van het sturen van boodschappen. In de eerste versie is communicatie synchroon, dat wil zeggen dat zowel zender als ontvanger wachten met communiceren totdat er een communicatie partner beschikbaar is. In de tweede versie is communicatie asynchroon, hetgeen betekent dat de zender zijn boodschap onmiddellijk verstuurt zonder op een partner te wachten, terwijl een ontvanger nog steeds moet wachten als er geen boodschap beschikbaar is. Als startpunt voor de ontwikkeling van een compositioneel bewijssysteem geven we een compositionele semantiek voor elk van deze twee versies van de programmeertaal.

De compositionele semantiek zal gebruikt worden als basis voor de interpretatie van de specificatietaal. In de hoofdstukken 2 en 3 van dit proefschrift is de specificatietaal gebaseerd op Explicit Clock Temporal Logic (ECTL). ECTL is een uitbreiding van lineaire tijd temporele logica met een speciale tijdsvariabele die expliciet refereert aan waarden van een globale klok. Overeenkomstig de programmeertaal zijn er van de specificatietaal ook twee versies, een synchrone en een asynchrone versie.

We ontwikkelen een compositioneel bewijssysteem voor elk van de twee versies van de programmeertaal en de specificatietaal. Er wordt bewezen dat beide bewijsmethoden gezond zijn met betrekking tot de semantiek (dat wil zeggen, alle in het bewijssysteem afleidbare formules zijn geldig) en relatief volledig zijn met betrekking tot een bewijssysteem voor ECTL (dat wil zeggen, alle geldige formules kunnen in het bewijssysteem afgeleid worden, mits alle geldige ECTL formules axioma's van het bewijssysteem zijn). De synchrone versie van het formalisme wordt in dit proefschrift toegepast bij het specificeren en verifiëren van een klein deel van een vliegtuig besturingssysteem.

Real-Time en Fout-Tolerante Toepassing

Na deze meer theoretische studie, waarbij het formalisme gebaseerd is op ECTL, onderzoeken we de specificatie en verificatie van realistische toepassingen. Omdat atomic broadcast een van de fundamentele concepten is in fout-tolerantie, kiezen we voor de bestudering van een atomic broadcast protocol. Dit protocol wordt uitgevoerd in een netwerk van processoren en communicatieverbindingen daartussen, en kan gekarakteriseerd worden door drie eigenschappen: terminatie, atomiciteit en ordening. Deze eigenschappen kunnen als volgt geformuleerd worden: als een correcte processor een boodschap broadcast dan dienen alle correcte processoren deze boodschap te ontvangen binnen een bepaalde tijdslimiet (terminatie), als een correcte processor een boodschap ontvangt op een bepaald tijdstip dan dienen alle correcte processoren deze boodschap op ongeveer het zelfde tijdstip te ontvangen (atomiciteit), en alle correcte processoren dienen boodschappen in dezelfde volgorde te ontvangen (ordening). De atomic broadcast service wordt geïmplementeerd in een netwerk van gedistribueerde processoren door het repliceren van een speciaal server proces op elke processor in het netwerk. Parallelle executie van de server processen dient te leiden tot deze drie eigenschappen van het protocol.

Een processor of een communicatieverbinding is correct als het zich gedraagt zoals gespecificeerd. Anders faalt het. Het gekozen protocol is ontworpen om omission fouten te tolereren. Als een processor een omission fout vertoont dan kan het geen boodschappen versturen naar andere processoren. Als een communicatieverbinding te lijden heeft van een omission fout dan kunnen boodschappen die via de link verstuurd worden verloren gaan. Boodschappen die door een processor ontvangen worden zijn echter correct betreffende timing en inhoud. Elke processor heeft toegang tot een lokale klok. Er wordt verondersteld dat lokale klokken van correcte processoren gesynchroniseerd zijn binnen een zekere marge.

De specificatietaal in de hoofdstukken 2 en 3 is gebaseerd op ECTL waarin de speciale tijdsvariabele kan refereren aan waarden van een globale klok. Gezien de complexiteit van ECTL formules en het streven om de formele verificatie nauw te laten aansluiten bij de intuitieve correctheidsargumenten, kiezen we in hoofdstuk 4 een andere specificatietaal gebaseerd op eerste-orde logica.

De verificatie van het protocol geschied als volgt. Allereerst worden de eigenschappen van het protocol beschreven. Ten tweede worden het onderliggende communicatie mechanisme, de kloksynchronisatie aanname en de aannames over het optreden van fouten geaxiomatiseerd. Ten derde wordt het server proces gekarakteriseerd door een formele specificatie. Ten vierde bewijzen we dat parallelle executie van de server processen tot de gewenste protocol eigenschappen leidt. Het protocol wordt compositioneel geverifiëerd door gebruik te maken van specificaties waarin de timing van componenten uitgedrukt wordt met behulp van lokale klok waarden. Dit in tegenstelling tot gebruikelijke real-time verificatiemethoden, inclusief onze bewijssystemen van de hoofdstukken 2 en 3, waarin timing uitgedrukt wordt met behulp van waarden van een globale klok.

Een natuurlijke voortzetting van dit werk is het implementeren van het server proces in een bepaalde programmeertaal en het verifiëren dat een implementatie inderdaad correct is. Dit wordt echter niet in dit proefschrift gedaan en behoort tot toekomstig werk.

Curriculum Vitae

The author of this thesis was born on May 22, 1964 at JianYang, Sichuan province, China. In 1980, she finished her secondary education and entered Wuhan University to study at the Department of Computer Science. In July 1984, her university education was completed with a project named "Design and Implementation of University Personnel Management System" and she was awarded a Bachelor's degree in Computer Science. From September 1984 to July 1987, she undertook her postgraduate study and research at the same department in Wuhan University and finished it with a Master's degree in Computer Science. Her master thesis was supervised by Prof. Qiongzhang Li and was entitled "A Temporal Semantics for a Distributed Programming Language". She was awarded a *Young Scientist Prize* by the 1st National Conference in Theoretical Computer Science held in Beijing, China in 1985 and an *Outstanding Postgraduate Research Prize* by Wuhan University in 1986. From August 1987 to April 1989, she worked as an assistant researcher at the Institute of Computer Application of Chengdu Branch of Chinese Academy of Sciences, and was awarded a *Young Scientist Prize* by the institute in 1988.

In October 1988 she met Prof. Willem-Paul de Roever who was invited to China by her master thesis external examiner Prof. Chaochen Zhou. This meeting resulted in an offer for her to work at Eindhoven University of Technology (Technische Universiteit Eindhoven, TUE). From May 1989 to January 1992, she was employed by the Department of Mathematics and Computing Science of TUE as a researcher in the Esprit-BRA project 3096 "Formal Methods and Tools for the Development of Distributed and Real-Time Systems" (SPEC). Since February 1992, she has been working as an "assistent in opleiding" for her Ph.D at the same department of TUE. When Prof. W.-P. de Roever left Eindhoven in 1990, her daily supervision was taken over by Dr. Jozef Hooman, who suggested the topics worked out in this thesis and helped her with the resulting research.

Stellingen

behorende bij het proefschrift

Clocks, Communications, and Correctness

van

P. Zhou

1. Consider the following two versions of a real-time programming language in which parallel processes communicate by message passing along unidirectional channels. In the first version, the communication is synchronous, i.e., both sender and receiver have to wait until a communication partner is available. In the second version, the communication is asynchronous, namely, a sender does not wait for a receiver, but a receiver still has to wait for a message arriving if there are no messages in the buffer for a specific channel. To obtain a compositional semantics for the synchronous version of the language, the model of computation should record the information that a process is waiting to send or to receive on a particular channel. For the asynchronous version, however, such waiting information is not needed, but explicit assumptions about the environment are contained in the model.

   See chapters 2 and 3 of this thesis.

2. *Maximal Parallelism* [KSR+88] means that each parallel process runs at a distinct processor. Therefore each process is executed without unnecessary waiting. When applied to the two versions of the programming language mentioned above, it has different implications. For the synchronous version, it implies that a process only waits when it tries to execute an input or output statement but the communication partner is not available. In the asynchronous case, however, it enforces that a process only waits when it tries to receive a message along a channel while the buffer for that channel is empty.

   See chapters 2 and 3 of this thesis.

   [KSR+88] R. Koymans, R.K. Shyamasundar, W.-P. de Roever, R. Gerth, and S. Arun-Kumar. Compositional semantics for real-time distributed computing. *Information and Computation*, 79(3):210–256, 1988.

3. ECTL (this thesis), RTTL ([Ost89]), XCTL ([HLP90]), and TPTL ([Hen91]) are real-time extensions of linear temporal logic. A comparison between them can be made according to their use of the time variable, global variables, universal quantification, and freeze quantification (which binds the value of the clock to the quantified variable):

	time var.	*global var.*	*universal quan.*	*freeze quan.*
ECTL	*yes*	*no*	*no*	*no*
RTTL	*yes*	*yes*	*yes*	*no*
XCTL	*yes*	*yes*	*no*	*no*
TPTL	*no*	*yes*	*no*	*yes*

[Ost89] J. Ostroff. *Temporal Logic for Real-Time Systems.* Advanced Software Development Series. Research Studies Press, 1989.

[HLP90] E. Harel, O. Lichtenstein, and A. Pnueli. Explicit clock temporal logic. In *Proceedings Symposium on Logic in Computer Science*, pages 402–413, 1990.

[Hen91] T. Henzinger. *The Temporal Specification and Verification of Real-Time Systems.* PhD thesis, Stanford University, 1991.

4. The atomic broadcast protocol in chapter 4 of this thesis is verified compositionally by using specifications about the protocol in which timing is expressed by local clock values. This is new in real-time specification and verification, since until now most methods for program verification use only global clock values, see e.g. [BHRR91].

   [BHRR91] J.W. de Bakker, C. Huizing, W.-P. de Roever, and G. Rozenberg(Eds.). *Real-Time: Theory in Practice, REX Workshop Proceedings.* LNCS 600, Springer-Verlag, 1991.

5. In Western society, Chinese names are usually transformed into English spellings consisting of letters. Such a transformation is possible for any Chinese name. On the other hand, an English spelling corresponding to a possible Chinese name can also be converted into a Chinese name. This conversion, however, is not a function in the mathematical sense, as many different Chinese names have the same English spelling.

6. A possible topic for future work is to develop a fault-tolerant proof system. Such a proof system can be formulated similarly to [CH92] where the behavior of a process is partitioned into the normal behavior and the fault behavior (that describes the behavior if a fault occurs).

   [CH92] J. Coenen and J. Hooman. A compositional semantics for fault-tolerant real-time systems. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 33–51. J. Vytopil (Ed.), LNCS 571, Springer-Verlag, 1992.

7. A key point to a compositional semantics is that the semantics of a component should contain all the possible executions of the component in any environment. A dictionary, which gives meanings to the words of a language, can be considered as a semantics. In reality, most of the dictionaries are not compositional, because they usually do not list all the meanings of a word in any context.

8. From the amount of verification steps in chapters 2 and 3 of this thesis and especially of the verification of the atomic broadcast protocol in chapter 4, it follows that the only future for this field is in supporting it by mechanical verification.

9. The semantics of a syntactic construct is not always uniquely defined. For instance, Tangram is an ancient Chinese game [Elf76], but it is also a VLSI-programming language [Ber92]. Nevertheless, we have to tolerate this phenomenon.

   [Elf76] J. Elfers. *Tangram: the Ancient Chinese Shapes Game.* Penguin Books, 1976.

   [Ber92] K. van Berkel. *Handshake Circuits: an Intermediary between Communicating Processes and VLSI.* PhD thesis, Eindhoven University of Technology, the Netherlands, 1992.

10. A highly educated woman around thirty is usually on the horns of a dilemma: to pursue her career or to have children. In Western society, these two cannot be carried out in parallel: choosing one implies that the other has to be delayed.