



Development of Communication Protocols with Algebraic-Temporal Specifications

vorgelegt von
Diplom-Informatiker
Mohamed Jmaiel

Vom Fachbereich Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades
Doktor-Ingenieur
genehmigte Dissertation

Promotionsausschuß:

Vorsitzender: Prof. Dr.-Ing. Stefan Jähnichen
Berichter: Prof. Dr. Peter Pepper
Prof. Dr. Bernd Mahr
Prof. Dr. Fred Kröger

Tag der wissenschaftlichen Aussprache: 26 Januar 1996

Berlin 1995
D 83

Abstract

In this thesis we are aiming at two main goals. First, we intend to provide a theory for the formal specification and verification of communication protocols. Algebraic specifications and temporal formulas are combined to specify data structures and data flows in a unified framework. The algebraic approach is used to specify the structure of data which will be exchanged between the components of a distributed system. The communication rules that control the data flow in the system will be described in terms of temporal logic. The second goal is to provide a methodology which supports the hierarchical development of protocols in a modular style. The application of the theory and the development methodology are illustrated through the specification and verification of some selected protocols.

Contents

1	Introduction	3
2	The Specification Language	6
2.1	How to Specify	7
2.2	Algebraic Specifications: An Overview	9
2.3	The Unified Algebraic Framework	11
2.3.1	Specification of Messages	11
2.3.2	Specification of the Behavior	12
2.3.3	Algebraic-Temporal Specifications	14
2.4	The Semantic Model	15
3	Temporal Logic	17
3.1	The History of Temporal Logic	17
3.2	The Temporal Logic	20
3.3	Indexing Temporal Operators	23
3.3.1	Semantics	24
3.4	The Proof System	27
3.5	Proving Temporal Rules and Theorems	29
4	Extending the Temporal Logic	39
4.1	The Inexpressiveness of Temporal Logic	39
4.2	Introducing Colors	40
4.3	Specification of Failures	42
4.4	Composition of Agents	44
4.5	Extending Operations w.r.t. Colors	45
4.6	The Extended Model	47

<i>CONTENTS</i>	2
4.7 The Predicate Calculus	48
4.8 A Specification Example: The Internet Protocol	49
5 Development Methodology	51
5.1 Processes and the Satisfaction Relation	52
5.2 Composition	53
5.3 Refinement	56
5.4 How to Develop Protocols	59
6 Applications	62
6.1 The Alternating Bit Protocol	62
6.1.1 Specification of the AB Protocol	62
6.1.2 Specification of the Transmission Medium	63
6.1.3 Formal Derivation of the AB Protocol	64
6.2 The CSMA/CD Protocol	71
6.2.1 Specification of the CSMA/CD Protocol	72
6.2.2 Specification of the Transmission Medium	73
6.2.3 Formal Derivation of the Protocol Specification	75
6.2.4 Considering Failures	79
6.3 A Transport Protocol	80
6.3.1 Specification of the Transport Protocol	80
6.3.2 Specification of the Transmission Medium	81
6.3.3 Formal Derivation of the Transport Protocol	82
6.3.4 Preserving the Order of Messages	83
6.3.5 The Three-Way Handshake	86
7 Conclusions and Related Works	94

Chapter 1

Introduction

This thesis presents a formal method for the specification and verification of communication protocols based on two well-known techniques: algebraic specification and temporal logic. It has long been evident that neither informal methods nor testing are reliable enough to establish the correctness of programs, especially concurrent programs which generally exhibit extremely complicated behavior. The application of formal methods to establish the correctness of programs began with the work of Floyd [Flo67], using flow charts, and Hoare [Hoa69], using a compositional proof system for while-programs. The problem of proving that a program should satisfy its specification was first recognized by Turing [Tur49] in 1949. In general, the verification of a program's correctness, that is the program meets its specification, is a very difficult task. In order to simplify this task, it is mandatory that the verification is started in the early stages of the design process of a program. A promising approach to guarantee the correctness of a program is to develop it step by step from its specification using suitable transformation rules. The method presented here deals with the verification of protocols in the early stages of development.

A communication protocol consists of a set of rules designed to enable data exchange between the agents of a distributed system. In this context, the formal specification of a communication protocol requires the description of two aspects of communication: data aspect and behavioral aspect. There are many well-established formal methods for the specification and verification of data types, e.g. VDM [BJ87], Z [Abr85], and algebraic specifications [EM85]. As a rule, behavioral aspects are specified using process description methods such as CSP [Hoa85], Petri nets [Pet62], finite-state machines [BS80], and temporal logic [Krö87, MP91]. All these methods have a solid theoretical background and have been applied successfully in different kinds of practical applications. In the majority of protocol specifications, data aspects and behavioral aspects are treated separately. However, with an eye to practical applications in the development of protocols, it turned out to be important to apply a specification technique which supports process description as well as data-type specifications. One of the prominent examples is the specification language LOTOS [Inf87], which is a combination of the algebraic specification language ACT ONE [EFH83] and the behavioral description calculus CCS [Mil80]. Other approaches are the algebraic high-level nets [EPR93], which combine algebraic specifications and Petri nets, and temporal Z [DS89], which integrates temporal formulas in the framework of Z.

In our opinion, applying a specification technique that describes data aspects as well as behavioral aspects in a unified framework facilitates the development of protocols in the sense that the user has to follow a single development process instead of two separate processes. Moreover, we believe that protocols have to be specified in a complete manner such that several layers can be described within the same specification. For example, it is very important to specify the connection-establishment protocol and the data-transfer protocol together, because they constitute the overall transport protocol. In addition, the use of a separate specifications requires a suitable notion for composing them which may be very intricate. For these reasons, we apply a combination of algebraic specifications and temporal logic as a language for specifying protocols; we import these two well-established techniques in order to describe data aspects and behavioral aspects of protocols in a unified framework. Apart from the advantages offered by formal languages, such as unambiguity and precision, the adequate integration of temporal logic in algebraic specifications has the following characteristics which make it a suitable method for specifying and verifying communication protocols.

- It allows us to abstract from implementation details. It is widely known that algebraic specifications are suitable for defining abstract data types. Temporal logic provides a more abstract way to describe protocol properties than transition systems and behavioral description techniques such as CCS and CSP.
- It supports formal reasoning about protocol properties and thus the formal verification of protocols because it is based on the mathematical foundations of algebra and on the rigorous proof systems of temporal logic.
- It provides a powerful formalism which is capable of expressing a wide range of protocol properties. It is widely recognized that the formalism of temporal logic is an appropriate tool for capturing interesting *safety* and *liveness* properties of communication protocols. In comparison to other techniques, temporal logic is one of the few in which liveness properties can be expressed in a simple and elegant way.
- It offers simplicity and usability. Temporal formulas as well as algebraic specifications are easy to use and understand. Moreover, in contrast to other techniques, such as finite-state machines, where the size of the proofs increases exponentially with the number of states, temporal logic proofs remain compact and clear.

A major goal of this work is to provide a methodology for developing communication protocols. Such a methodology will enable us to develop design specifications from requirement specifications using transformations and refinements. In the context of distributed systems, it is important that the methodology allows the independent development of system components. In order to be able to construct such a methodology, it is necessary to have a modular specification language. Unfortunately, temporal logic in its elementary form is non-modular. In general, a system component can neither be specified nor verified independently of its environment [BKP84]. The reason is that temporal formulas are global, i.e. temporal formulas in a system specification refer to all system components. In this approach, the modularity of temporal logic is achieved by indexing temporal operators with identifiers for system components. In describing systems, one can select the particular system component which should satisfy a temporal formula. In this way, one system component can be specified independently of the other components.

Apart from the lack of modularity, we have to deal with some expressiveness problems: we will show that a large class of protocol properties cannot be specified using classical temporal logic. The lack of expressiveness is due to the inability of temporal logic to uniquely identify messages on streams. In order to overcome these expressiveness problems, we extend the semantic model by providing a mechanism that enables unique identification of messages on streams. We introduce “colors” (they may be viewed as an abstraction of time-stamps) which serve as identifiers on streams. It turns out that we need a hierarchy of equivalence classes of colors in order to express desired properties of protocols.

Based on the modular temporal logic, we will provide a methodology for developing protocol specifications. The methodology enables the stepwise development of a design specification from an abstract specification describing the requirements put on the protocol. In order to ensure the correctness of the final product, we define a set of rigorous rules for composing and refining specifications. Moreover, we define a satisfaction relation between the specifications involved during the development activities. We will show that our refinements are relations which are transitive and compatible with the composition of specifications. Accordingly, we obtain a methodology that supports the hierarchical development of protocols in a modular style. It should be stressed that in our methodology we follow principles and strategies successfully applied in the **KORSO**-methodology [PW94].

In order to demonstrate the applicability of the theory and the development process, some protocols such as the Alternating Bit protocol, a transport protocol including the data-transfer and the connection-establishment parts, and a CSMA/CD protocol are specified and verified.

This thesis is organized as follows. Chapter 2 defines the language that we use to specify protocols. First, we describe the aspects considered in a specification. Next, we show how algebraic specifications and temporal formulas can be embedded into a common algebraic framework. A semantic model for the specification language is also given. The temporal logic that we apply in our specifications is introduced in Chapter 3. We begin this chapter with a brief overview of the history and the development stages of temporal logic. After presenting an informal interpretation of the usual temporal operators, we consider an indexed version of temporal logic. Then, a formal interpretation of the indexed temporal operators and a proof system are given.

In Chapter 4 an extension of the temporal logic is considered, with a view to augmenting the expressive power of the language. First, we discuss the causes of the expressiveness problems. Then, we extend the logic step by step. We illustrate the extended temporal logic by specifying some services provided by the Internet protocol. Chapter 5 presents a development methodology for communication protocols. We first introduce a notion of processes, in order to be able to reason about compositionality, modularity, and refinement in a rigorous way. Next, composition of specifications is investigated and a justification rule is given. We then tackle refinements and present a proof rule for verifying them. Finally, we give some guidelines for developing design protocol specifications from requirement specifications.

We illustrate the methodology presented in this thesis in Chapter 6. Three protocols are considered. First, we develop a specification for the Alternating Bit protocol. Then, we specify a CSMA/CD protocol. Finally, we develop a specification of a transport protocol.

Chapter 2

The Specification Language

A distributed system consists of a set of agents that execute independently and interact with each other in order to exchange data. The programs which are responsible for the realization of a reliable data exchange within a distributed system are called *communication protocols*. Usually, communication protocols exhibit extremely intricate behavior, because they must cope with the possibility of failures in the physical components of the system. Due to their complexity, the development of protocols is considered a difficult task and should follow rigorous formal techniques in order to ensure correct implementations. In this context, the formal specification of protocols is of particular importance, because it forms the basis of a correct implementation.

In specifying communication protocols there are two aspects of communication that should be considered, namely the communication rules and the structure of the data to be transmitted within the system. Many different formal languages have been developed for and applied to the description of protocols. The most important approaches are finite-state machines [BS80], CSP [Hoa85], CCS [Mil80], Petri nets [Pet62], and temporal logic [Krö87, MP91]. However, the majority of existing languages consider only the description of the behavioral aspect of a protocol, whereas the aspect of data representation is treated separately using other specification methods, such as Z [Abr85], VDM [BJ87], or algebraic specifications [EM85]. In our opinion, this separation may lead to some difficulties during the development process, because in order to be able to derive the correctness of the protocol implementation from the correctness of the implementations of the data part and the behavioral part, a suitable notion of composition of the specification languages is required. Moreover, the user has to follow two development processes based on different kinds of transformation rules, rather than just one process based on one sort of rules. For these reasons, we think that a formal specification language for protocols should be complete, in the sense that it covers all relevant aspects of a protocol.

In this chapter, we propose a *unified framework* for specifying protocols in a complete fashion. A specification includes a description of the data types, the network topology and the description of the system behavior. To specify the data types and the network topology we apply algebraic techniques, while the behavior is supported by a suitable temporal language. Our formalism is obtained by combining temporal logic and algebraic specifications. The major difference to LOTOS is that our language is not only a disjoint union of two formalisms, but



Figure 2.1: A Single Agent

also an integration of the one into the other; we integrate temporal formulas into the algebraic framework. Moreover, because temporal logic describes the behavior in a more abstract way than CCS, the integration of temporal specifications in the algebraic framework, as presented in this chapter, provides a formalism by which protocols can be specified in a more abstract way than by LOTOS.

This chapter is organized as follows. Section 2.1 presents the abstract model of distributed systems that we use to describe protocols. In Section 2.2 we give a brief overview of algebraic specifications, as described by Ehrig and Mahr [EM85] and Wirsing [Wir90]. Then, in Section 2.3, the specification language for protocols is defined: we show how specification of data types and temporal formulas can be embedded into a common algebraic framework. Finally, the semantic model based on algebras and event structures is given.

2.1 How to Specify

In our approach, a distributed system is modeled by a family of interacting *agents* which represent the logical units of the system, e.g. sender, receiver, or transmission medium. One of the main characteristics of an agent is that it does not operate in isolation, but exchanges data with its environment via unidirectional *channels*. A channel is considered to be an abstract interface of an agent to its environment. So each agent is associated with a finite set of distinctly named input and output channels (see Figure 2.1). An agent may be viewed as an entity which receives and sends data on its input and output channels. Basically, this representation corresponds to the CSP model proposed by Hoare [Hoa85].

A network of agents is formed by linking some input channels of some agents to some output channels of other agents in one-to-one manner. A network may be pictorially represented as in Figure 2.2. In such a network, we distinguish two kinds of channels, namely *internal* and *external* channels. An internal channel is a connection between exactly two agents; it is used by the one for input and by the other for output. External channels provide a boundary between the system and the outside world. They may be used to embed one system into another or to extend the original system by adding further agents.

However, it is important not to confuse channels and physical transmission lines; channels are *conceptual* representations of communications between agents. Semantically, channels are considered as streams of data elements. Thus, they are absolutely reliable, in contrast to physical transmission media which, in general, may lose, duplicate, or permute messages. In our approach, we model a transmission medium by an active agent, in order to describe how it transmits messages. A comparison between a “real-world” communication net and its abstract model can be shown by the following simple example. Consider two computers *A* and *B* connected by a wire, which is the actual transmission medium (see Figure 2.3). This

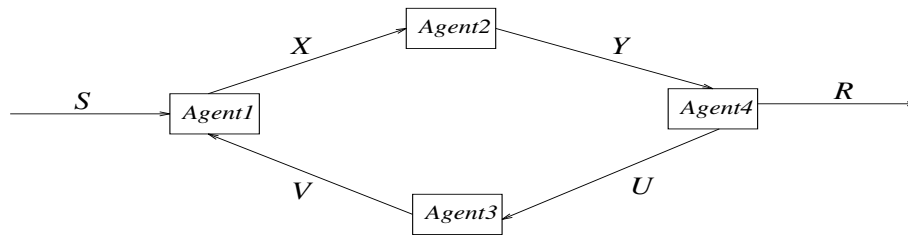


Figure 2.2: A Network of Agents

computer network is modeled by three active agents, the computers A and B , and the agent $Wire$ representing the transmission medium. The interfaces between the computers A and B and the wire are represented by the channels S and R , which reflect the data flow over time from the computer A into the wire and from the wire into the computer B , respectively.

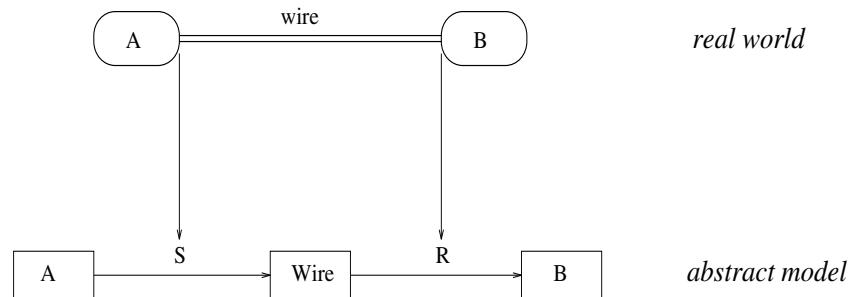


Figure 2.3: Modeling a Real-world Situation

For any given network, we have to describe the communication rules that control the data exchange between the agents. These rules must be respected by the agents, in order to achieve the desired communication service. Accordingly, we specify the behavior of the agents such that the rules of communication are respected and the desired service is provided. In essence, a specification consists of a set of requirements put on the behavior of the agents. However, we do not consider any details relating to the internal structure of the agents, that is, an agent is viewed as a black box that passes messages from its input to its output channels. From this point of view, the specification of a system is restricted to the description of its observable input-output behavior, characterized by a relation between the communication actions which occur during a system run; this relation is often called the *causality* relation. This behavioral aspect will be covered by temporal logic. Thus the behavior of a system comprises a set of temporal formulas which express *safety* and *liveness* properties of the system. We will study the syntax and the semantics of our temporal language in detail in the next chapter.

Apart from the behavioral aspects of a system, we will also describe the structure of messages carried along the channels, as well as the operations available to the agents to treat them, i.e. we will specify the data types of messages. In order to achieve this in an abstract way, we apply the formalism of algebraic specifications, because this is particularly suitable for the specification of abstract data types.

To this end, a protocol specification should take into account the two aspects mentioned above, viz. the data types of messages and the behavior of the agents. Conceptually, these aspects are

quite different, and the choice of different techniques to specify them follows good principles of software engineering. In order to describe these aspects in a unified framework, we need to relate these formalisms. Therefore, we also integrate temporal formulas into algebraic specifications. The specifications obtained are called algebraic-temporal specifications.

2.2 Algebraic Specifications: An Overview

Algebraic specifications provide a formalism for describing data types in

an abstract way without consideration of any implementation details. The basic idea of the algebraic approach is to describe data types by giving the names of the data sets, the names of the operations, and the

properties that should be satisfied by the operations. The combination of the names of the data sets (*sorts*) and the names of the operations with their functionalities constitutes the signature of this data type. Formally, a *signature* Σ is a pair $\langle S, F \rangle$ where S is a set of *sorts* and F is a set of *function symbols*, each of which

is associated with a *functionality*: for any $f \in F$ the

functionality of f is an element from $S^* \times S$ expressing the type of f . The properties of the operations are formulas expressed in a specific logical system. In the classical algebraic specifications (e.g. as defined by the ADJ-group [GTW78]) the operation properties are given by a set of equations over the signature. In this approach, a *specification* is a pair $\langle \Sigma, E \rangle$, where Σ is a signature and E is a set of equations over Σ . One of the prominent algebraic languages based on

equational logic is ACT ONE [EFH83]. But there are also algebraic specification languages based on other logical formalisms, such as Horn-clause equational logic, first-order logic [Wir90], or higher-order equational logic [Möl85]. For instance, the specification language SPECTRUM, defined in [BFG⁺93], is based on

first-order logic.

The semantics of algebraic specifications is based on the mathematical concept of algebras. An algebra consists of a collection of *carriers* (sets of data) and operations on them. To any signature Σ one can associate an algebra

(called Σ -algebra) by assigning a carrier set to each sort in Σ , and an operation on these sets to each operation symbol in Σ . Formally, let $\Sigma = \langle S, F \rangle$ be a signature. A Σ -algebra A consists of a family of carrier sets $\{A_s\}_{s \in S}$ and a function $f^A : A_{s_1} \times \dots \times A_{s_n} \rightarrow A_s$ for each $f : s_1, \dots, s_n \rightarrow s \in F$. In the classical algebraic approach the functions in a Σ -algebra are considered to be total. However, in

describing data types of programs it is important to be able to specify partial functions too because, in general, we have to deal with error cases and non-terminating algorithms (e.g. recursive functions). Although total

approaches are simple to work with, they pose difficulties if data types with partial functions have to be specified. Many approaches have been developed to provide a formalism for partial function handling. For instance, *partial algebras* deal with error situations and *continuous*

algebras (based on fixpoint theory) provide a semantics for non-terminating algorithms.

For any algebraic (data) specification $Dspec = \langle \Sigma, E \rangle$ a *Dspec-algebra* is a Σ -algebra that satisfies the properties E . Looking to the semantic models of algebraic specifications, we distinguish two main approaches: *loose* and *initial* semantics. In the loose approach, the semantics of a specification is given by the class of all *Dspec*-algebras, denoted by $Alg(Dspec)$. In the initial approach only particular algebras are considered; those which are initial in the class $Alg(Dspec)$. Roughly speaking, an initial algebra of a specification *Dspec* satisfies only formulas which are valid in all *Dspec*-algebras. Initial algebras are characterized uniquely up to isomorphism by the property that there is a unique homomorphism to each algebra in $Alg(Dspec)$ [EM85]. The initial approach provides a simple and elegant basis for the semantics of abstract data types. However, it is not always possible to associate an initial semantics to algebraic specifications; it is possible to show that, taking an arbitrary first-order logic as a formalism to express the properties of the operations, an initial algebra in the class of *Dspec*-algebras does not always exist [EM85]. The main difference between loose and initial semantics is that in the initial semantics a specification is represented by a unique (up to isomorphism) model, whereas the loose semantics allows different models for a specification. This makes loose semantics more appropriate for the development of programs using specification transformations and refinements, because the loose approach allows the user to add design decisions during a development process which starts from an abstract specification.

Apart from the extensions of the classical algebraic approach to model important features of sequential programs, algebraic specifications have been extended in order that they can support the description of concurrent systems as well. From a methodological point of view, we distinguish two main approaches. The first approach is a pure algebraic technique, where processes are considered as special data types [AGR88]. In the second approach, the specification language is a combination of techniques like CCS [Mil80] and Petri nets for the description of the behavioral aspect of processes, with algebraic languages like ACT ONE [EFH83] used for the specification of abstract data types, for example the language LOTOS [Inf87] and the algebraic high-level nets [EPR93].

As mentioned above, the logical formalisms applied to express the properties of systems specified in an algebraic framework are not restricted to equational logic. In general, the axioms of a specification may be formulated in first-order logic, Horn-clause logic, higher-order logic, or temporal logic. Usually, the choice of the logic depends on the particular area of application. For example, equational logic is appropriate for describing abstract data types, Horn clauses are usually applied in connection with logic programming, and temporal logic is suitable for capturing safety and liveness properties of concurrent systems. In this context, it is very useful to have a relation between logical systems, such that interesting results (such as proving program properties) which are achieved in one logic can be translated to another one. For this reason, the concept of *institutions* was introduced by Goguen and Burstall [GB84] to provide a foundation for making interesting results completely independent of the underlying logical system. Informally, an institution consists of a collection of signatures with signature morphisms together with, for any signature Σ , a set of Σ -sentences, a set of Σ -models, and a satisfaction relation between Σ -sentences and Σ -models, such that when one changes signatures (by a signature morphism), satisfaction of sentences by models changes consistently [GB84]. That is, using institutions one can translate formulas from one logic to another logic

in such a way that soundness is preserved. Furthermore, institutions make program development concepts such as composition, verification, and refinement of specifications independent of the logic in which the specifications are expressed (for these and other theoretical results, see [GB92]).

2.3 The Unified Algebraic Framework

As mentioned in Section 2.1, a protocol specification includes the specification of the messages that will be exchanged between the agents as well as the behavior of these agents. In specifying the behavior of agents, we also consider the network topology of the system so that we have a communication structure telling us how agents are related to each other. This section presents a *unified algebraic framework* incorporating all these aspects.

2.3.1 Specification of Messages

Algebraic specifications, as introduced in Section 2.2, can also be applied in the context of communications. We make use of the classical algebraic approach to describe the data types of messages and use the following schema for representing algebraic specification. Our terminology is influenced by the schema used for representing specifications of OPAL programs (see, e.g. [DFG⁺94]).

```

SPECIFICATION << name >>
IMPORT << other specifications >>
SIGNATURE << collection of sorts and operation declarations >>
PROPERTIES << list of axioms (usually equations) describing the operations >>

```

The import list makes some sorts and operations defined in other specifications available in the new specification. This is very useful in describing data types, because repetitions are avoided and specifications are more structured. Actually, the import part enriches the sorts, the operations, and the equations of the current specification by the sorts, the operations, and the equations of the imported one.

In our context, the signature comprises the sorts of messages to be transmitted within the system, as well as operations, such as splitting messages into packets, adding (or selecting) fragments to packets (e.g. destination address, sequence number or error detection code), and building complete messages from individual packets.

In this algebraic framework, we can deal with topics relevant in the context of protocols, such as error detection, segmenting, reassembling, address resolution, routing, and so forth. Many approaches handle these topics at a very low level dealing with bit positions and encoding schemes. Obviously, these issues include many implementation details which do not need to be considered at the highest level of protocol specification. Indeed, algebraic specifications provide the appropriate formalism for describing these topics independently of implementation details. Moreover, using a suitable development methodology, implementation details (e.g. encoding schemes and bit positions) can be added in later refinement steps.

As an example, we consider the following situation: we have to realize a message transmission over a packet transmission medium. On the sender side, an incoming message should be split into packets, whereas on the receiver side, packets should be recombined into messages. Accordingly, we have to specify two operations: `split` and `compose`. The main property of interest is that the original message is yielded by the recombined packets. In our terminology, this is given by the following specification:

```

SPECIFICATION Split-Compose
IMPORT Messages ONLY message
IMPORT Packets ONLY packet
SIGNATURE
  FUN  split : message → Set[packet]
  FUN  compose : Set[packet] → message
PROPERTIES m : message
  AXM  compose(split(m)) = m

```

In general, packets have, in contrast to messages, a fixed size. Thus, when messages are split into packets, the number of yielded packets is not constant; it depends on the size of the message. For this reason, we specify `split` as a function which assigns to each message a (finite) set of packets, and not a list of a fixed length of packets. Moreover, in splitting a message into packets, information - other than the data part - needed for reassembling should be added to the packets, i.e. a sequence number, or a code identifying the last packet of a message. All these details are irrelevant at the algebraic level. Indeed, the property (AXM) is sufficient to ensure the proper functioning of the “split-recombine” routines.

2.3.2 Specification of the Behavior

In specifying the behavior of a system we have to take the network topology, which describes how the agents are connected to each other, into consideration. In general, networks are represented graphically. But in our algebraic framework we have to deal with a more formal representation of networks. Because graphical representations are simpler (than their formal description) to work with, in our specifications we will use the graphical representation of networks as an abbreviation for their formal description, because (as we will see) there is a one-to-one correspondence between them.

In order to fit graphical representations into algebraic specifications, we adopt a model proposed, among others, by Broy (see [Bro88]) in which agents are viewed as stream-processing functions and channels are seen as streams of messages. Based on this setting, an agent is (algebraically) represented by an equation of the form $A(S_1, \dots, S_n) = (R_1, \dots, R_m)$, where A is the agent, S_1, \dots, S_n are the input, and R_1, \dots, R_m the output channels of A . From this point of view, a network of agents is defined by a *set of recursive stream equations*. For example, the following network:

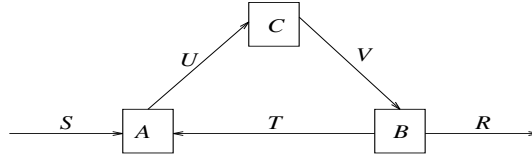


Figure 2.4: Graphical Representation of a Network

corresponds to the following three equations:

$$\begin{aligned} A(S, T) &= U \\ B(V) &= (R, T) \\ C(U) &= V \end{aligned}$$

Note that such a network corresponds exactly to one graphical representation in which external channels occur only on one side of the equations, and internal channels occur on both sides.

Because we also consider the data type of messages carried on the channels of a system, a network is defined in connection with a given signature specifying the sorts of messages exchanged within the network. Formally, we define a network as follows:

Definition 2.3.1 [Network]

Let Σ be a signature. A Σ -network $Net = (Ag, St, Eq)$ consists of a collection Ag of agent names, a collection St of stream variables over the sorts of Σ , and a set Eq of equations of the form $A(S_1, \dots, S_n) = (R_1, \dots, R_m)$. The set of equations includes exactly one equation for each agent. A network has to satisfy a completeness property which states that each name in St is either an input or output channel of an agent, or a connection between two agents.

In a network of agents, the basic actions that may occur are transmission of messages; their occurrences are expressed by predicates such as

$[A \text{ rcv } m \text{ on } S]$ The agent A receives the message m on the channel S .

$[U \text{ xmt } m]$ The channel U transmits the message m .

$[B \text{ snd } m \text{ on } T]$ The agent B sends the messages m on the channel T .

We omit the **on**-part if this is clear from the context. For example, we write $[A \text{ rcv } m]$ if the agent A has only one input channel. Moreover, we use the abbreviation $[A \text{ rcv }]$ if no reference to the message is needed (“ A receives some message”).

These predicates constitute the atomic formulas of the specification language; their formal meaning will be given later in connection with temporal logic. We should mention that the networks we are considering exhibit *strongly coupled communication*: such networks are characterized by the equalities (referring to the network in Figure 2.4)

$$[A \text{ snd } m \text{ on } U] \equiv [U \text{ xmt } m] \equiv [C \text{ rcv } m \text{ on } U]$$

We use the discipline of temporal logic as the formalism for specifying the behavior of a system. The specification of the behavior is given by a set of temporal formulas expressing *safety* and *liveness* properties of the system. By combining temporal operators and transmission predicates, we are able to express interesting properties of communication protocols. For example, the liveness property saying that an agent A eventually sends what it has received can be expressed by the formula $([A \text{ rcv } m] \Rightarrow \diamond [A \text{ snd } m])$, where ‘ \diamond ’ is a temporal operator meaning “eventually”. The temporal logic applied here will be presented in detail in the next chapter.

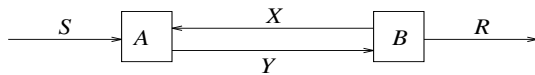
2.3.3 Algebraic-Temporal Specifications

The specification of the behavioral aspects of protocols is now given in algebraic-temporal specification. Following the idea that agents are represented by stream-processing functions, and channels by streams, we embed temporal formulas in the framework of algebraic specification¹ in the following way:

```

SPECIFICATION << name >>
IMPORT << other specifications >>
SIGNATURE << collection of agents and their functionalities, such as >>
  FUN A: stream × stream → stream
  FUN B: stream → stream × stream
NETWORK << a communication network for the given agents, such as >>

```



```

PROPERTIES << temporal formulas describing the behavior of the agents >>

```

Actually, the graphical representation of the network is merely syntactic sugar for the temporal formulas. As mentioned above, a graphical representation corresponds to a specific system of equations. The above network is therefore a shorthand version of the following extension of the property part:

```

PROPERTIES ∀ S, X, Y, R: stream
  A(S, X) = Y ∧ B(Y) = (X, R)
  ⇒ << temporal properties >>

```

This entails that our specifications usually introduce and characterize a certain set of agents, whereas the *names of the streams are bound and thus exchangeable*.

Using parametrized algebraic specifications (see, e.g. [EM85]), one can deal with typed streams, such as streams of messages, packets, or frames. In this way, and by using the above algebraic framework, one can easily integrate the specification of messages into the specification of the behavior. We obtain a *unified framework* in which several aspects of protocols can be described at a high level of abstraction.

¹In the terminology of Goguen and Burstall [GB84], we use the institution of temporal logic.

In order to reason about specifications in a rigorous way, we formally define specifications as follows:

Definition 2.3.2 [Specification]

A specification \mathcal{S} consists of a specification $Dspec = (\Sigma, E)$ of the data part, a Σ -Network $Net = (Ag, St, Eq)$ (as defined in 2.3.1), and a set Pr of temporal formulas. The network together with the set of temporal formulas represents the following axiom:

$$(\forall S_1 : stream[s_1], \dots, S_n : stream[s_n] : (\bigwedge_{e \in Eq} e)) \Rightarrow Pr$$

where $\{S_1, \dots, S_n\} = St$ and $(\bigwedge_{e \in Eq} e)$ is the conjunction of all equations in Eq .

This definition, in which we separate the temporal formulas from the network representation, facilitates the definition of the semantics and it is suitable for defining development steps.

2.4 The Semantic Model

In this section, we define the semantic model for algebraic-temporal specifications. Like the syntax, the semantics refers to two aspects: it combines a model for data types and a model for system behavior. Semantic models for both algebraic specifications and temporal logic have already been established and shown to be suitable for reasoning about the aspects of data types and system behavior, respectively.

As mentioned in Section 2.2, models for algebraic specifications are based on the notion of algebra. Models for distributed systems are usually based on either states or events. In a state-based approach, a system execution is a sequence of states, where each state is an assignment of values to the variables of the distributed program that implements the system. Many models for temporal logic are based on states: the most popular is the logic presented by Manna and Pnueli in [MP91]. Generally, states are global, i.e. a state comprises the values of all variables of the distributed program. Reasoning about distributed systems using state-based temporal logic forces us to attach temporal formulas to the global states of distributed programs. *In our opinion, the globality of states prevents the compositionality of temporal logics based on such models.*

In the event-based approach, one is interested in the communications performed by the system components without taking program variables into consideration. This approach has been used e.g. by Hoare [Hoa85] to model CSP processes. Here, we make use of the event-based approach to model distributed systems. Generally, the occurrences of communications during a system run are partially ordered by a *causality* relation, because two communications in separate locations may occur concurrently. Usually, the causality is modeled by a linear order, in other words: concurrency is modeled by non-deterministic interleaving. Obviously, taking partial orderings as the “time structure” for the interpretation of temporal formulas yields a logic that is more expressive than linear-time logic. For some applications, partial orders are needed to express desired properties that cannot be expressed in linear models; examples

are the concurrency properties investigated in [Rei88]. However, as long as such properties are not relevant for the description of the intended systems there is no loss of expressiveness in considering linear-ordered sets of events. This is because a partial ordering is completely equivalent to the set of total orderings consistent with it.

In this work, we restrict our attention to the interleaving semantics for two reasons. First, linear temporal logic is powerful enough to express properties of the protocols we intend to specify. Second, we think that linear-time formulas are better understood and, therefore, easier to work with. In the interleaving semantics, the behavior of a distributed system consists of the set of its possible runs, where each run is a sequence (maybe infinite) of events. An *event* represents the occurrence of a send, receive, or transmission action during a system run. Since each send or receive action is “equivalent” to a transmission action on the corresponding channel, we may model an event by a pair (c, m) , where c is a channel name and m the message.

A model $\mathcal{M} = (Alg, (\mathcal{E}, \leq))$ consists of an algebra Alg and a linear-ordered set \mathcal{E} of events, satisfying the property that the messages in the events of \mathcal{E} are data elements of the algebra. Given a specification $\mathcal{S} = (Dspec, Net, Pr)$, we say \mathcal{M} is a model for \mathcal{S} iff Alg is a *Dspec*-algebra and the temporal properties in Pr are satisfied by \mathcal{M} . The validity relation between temporal formulas and models will be defined in the next chapter.

For a given specification $\mathcal{S} = (Dspec, Net, Pr)$ and a *Dspec*-algebra Alg , we define the semantics of \mathcal{S} w.r.t. the algebra Alg as the class of all models over this algebra, that is the class of all event sequences over Alg which satisfy the properties Pr . Formally,

$$\llbracket \mathcal{S} \rrbracket_{Alg} \stackrel{\text{def}}{=} \{ \mathcal{M} = (Alg, (\mathcal{E}, \leq)) \mid \mathcal{M} \models Pr \}$$

The semantics of a specification \mathcal{S} is given by the class of all models which satisfy the specification \mathcal{S} , this corresponds to the union of all fixed-algebra semantics of \mathcal{S} :

$$\llbracket \mathcal{S} \rrbracket \stackrel{\text{def}}{=} \{ \mathcal{M} \mid \mathcal{M} \models Pr \} = \bigcup_{Alg \in Alg(Dspec)} \llbracket \mathcal{S} \rrbracket_{Alg}$$

That is, we consider *loose semantics*. As mentioned before, in the context of algebraic specifications, we distinguish another kind of semantics, namely the *initial semantics*. In contrast to loose semantics, which allows different models for a specification, initial semantics considers only one model (up to isomorphism) as semantics for a specification, which can be uniquely (up to isomorphism) determined. One of the major advantages of initial semantics is that it makes reasoning about systems intuitive and simple. However, taking one model as semantics is not adequate with specification-building operations, such as enrichment and refinement. Loose semantics, in contrast, allows the user to add design decisions during a development process. Because, we are aiming at a development methodology that supports stepwise refinement, we adopt loose semantics. In a development process, we starts with an abstract specification allowing all possible models (implementations) that fulfill all requirements placed on the desired system, and then we construct step by step a design specification by adding axioms that correspond to design decisions. Obviously, these axioms constrain the class of models and, therefore, reduce the number of models at each step.

Chapter 3

Temporal Logic

There are basically two classes of computer programs. In the first class, a program is viewed as a function from an initial to a final state, a relation between initial and final states in the case of nondeterminism. This class includes, for example, programs which accept inputs at the beginning of their execution and yield outputs at the termination. In reasoning about this kind of programs, it suffices to refer to the initial and the final state of a program. One of the prominent formalisms for the specification and verification of such programs is the Hoare logic. The other class consists of programs which are designed to run permanently, such as communication protocols, control programs, operating systems, etc. These programs are not intended to yield final results, but rather to maintain some interaction with their environment. Clearly, such programs cannot be adequately described by referring only to the initial and final state. An adequate description must refer to all states that can arise during a program execution. Temporal logic is one of the suitable techniques for the specification and verification of this kind of program, because it provides a formalism for reasoning about states (or events) of a program execution and their relation in time. Its semantics distinguishes between the static aspects of a program, represented by states, and the dynamic aspects, represented by the relation (in time) between states. Temporal logic is a simple and elegant extension of propositional logic (predicate logic in the case of first-order temporal logic) which is still powerful enough to express interesting properties of the programs differentiated above.

This chapter is organized as follows. The next section presents a brief overview of the history and the development stages of temporal logic. It also gives an overview of the various models of temporal logic and their application areas. In Section 3.2, we define the temporal logic that we use to specify protocols, and investigate how interesting properties of protocols can be formulated in this logic. In Section 3.3, we consider an extension of our temporal logic needed to support modular specification and verification. A proof system for the propositional part of this temporal logic is presented in Section 3.4. We conclude this chapter by proving some temporal theorems and rules.

3.1 The History of Temporal Logic

Temporal logic belongs to the class of *modal logics*, since long a subject of research. In modal logic, we distinguish basically between propositions that must be true and propositions that

may be true. A proposition that must be true is called *necessary*, one that may be true is called *possible*. The notion of possibility and necessity are dual, i.e. if a proposition is necessarily true then its negation is not possible and vice versa. There are two operators to denote the modalities¹: \Box interpreted as “it is necessary that” and \Diamond interpreted as “it is possible that”. In the modal logic, the modal operators \Box and \Diamond are used in addition to the usual propositional operators, such as \neg , \wedge , \vee , \Rightarrow , and \Leftrightarrow .

The theory of the modalities necessary and possibly was even studied by the ancient Greeks, e.g. by Megarian and Stoic, and later was developed by medieval Arabian logicians, e.g. Avicenna (see [RU71]). The modern revival of modal logic in the 1920s can be attributed to C.I. Lewis. Lewis was dissatisfied with the *material implication* (the \Rightarrow of the propositional logic); in his opinion, “ p implies q ”, as used in ordinary discourse, meant more than just “it is not the case that (p and not q)”. He proposed a stronger implication, called *strict implication* (\prec); $p \prec q$ means “it is impossible that (p and not q)”. To formulate the strict implication he used the modal operator \Diamond . The strict implication is defined as:

$$p \prec q \stackrel{\text{def}}{=} \neg \Diamond (p \wedge \neg q)$$

In 1932, Lewis proposed five axiom systems for reasoning about strict implication: $S1$, $S2$, $S3$, $S4$, and $S5$. These and other modal systems are described by Hughes and Cresswell [HC68].

A formal semantics for modal logic was first proposed by Kripke in 1963 [Kri63]. Kripke gave the most famous concept of models for modal logic: *the possible world semantics*. He defined a *model structure* as a triple $\langle G, K, R \rangle$, where K is a set of all possible worlds, $G \in K$ is the “real world”, and R is a reachability relation between worlds. Given a model structure $\langle G, K, R \rangle$, a *model* is a function that assigns to each atomic formula (proposition variable) p a truth-value (*true* or *false*) in each world $w \in K$. The model proposed by Kripke enables us to formalize the interpretation of the modalities. The formula $\Box p$ is true in a world w iff the formula p is true for all worlds w' reachable from w . Similarly, $\Diamond p$ is true in a world w iff p is true in at least one world w' reachable from w (see [Kri63]). Kripke required that R has to be reflexive: every world should be reachable from itself. Depending on the axiom system in question, further conditions are required of R , e.g. working with the axiom system $S4$, the reachability relation should also be transitive. A relationship between several modal systems and the properties of the reachability relation can be found in [HC68].

A temporal interpretation of the modal operators \Box and \Diamond was first given by Prior in 1957. He defined a time structure as a linear-ordered set of time instants such that at each time instant a proposition is assigned a truth value (*true* or *false*). Prior interpreted \Box and \Diamond as “always” and “eventually”, respectively [Pri57]. The axiomatization of Prior’s logic yields two axiom systems, $S4.3$ and D , depending on whether the time structure is continuous or discrete. The system $S4.3$ models continuous time, whereas the system D models discrete time (see [HC68]). Considerations about discrete time structures gave rise to the introduction of a new temporal operator: “*next-time*”. A complete axiom system for a temporal logic that includes the operators “always” and “next-time” was given in [Pri67]. Another binary operator, “*until*”, was introduced by Kamp in [Kam68]. A complete axiomatization of this operator can be found in [Bur82].

¹Many authors, e.g. Hughes and Cresswell (see [HC68]), use L and M for \Box and \Diamond , respectively.

In 1977, Pnueli proposed the application of modal logic in reasoning about concurrent and nondeterministic programs in his paper “*The Temporal Logic of Programs*” [Pnu77]. The idea of applying modal logic in program specification and verification had already been suggested by Burstall 1974 [Bur74]. This idea was later elaborated by other authors, e.g. Kröger [Krö76]. Kröger, in 1978 [Krö78], investigated a temporal logic for program verification with three operators: “*next-time*”, “*always*”, and “*eventually*”. Since that time, numerous attempts have been made to demonstrate the usefulness of temporal logic in specifying and verifying concurrent programs. Today, temporal logic is one of the most widely used formal techniques in the development of computerized systems; it has been applied in the specification and verification of a large class of systems, such as reactive, real-time, and hybrid systems, (see, e.g. [MP91], [MP93], and [AH89]).

The application of temporal logic in the field of program verification is based on the idea of considering program executions as time structures to interpret the temporal operators. Due to the different views of how the states of a program are related (in time) to each other, we distinguish different types of temporal logics. The most common are linear-time and branching-time logics. The linear approach adopts a linear ordering as a model for time, whereas the branching-time approach uses tree-structured time. The main difference between linear- and branching-time logics is that the former reasons about detached program executions, while in the branching-time approach all executions are taken in a single structure (tree) distinguishing the states at which a nondeterministic choice has been made. In the literature we find different variants of branching-time logics. A simple variant is the unified system of branching-time (UB) presented in [BAPM81]. The most expressive branching-time logic is *The Computational Tree Logic* (CTL^*), defined by Emerson and Halpern in [EH86]. This logic subsumes the other branching-time logics and the linear-time logic as well. The answer to the question whether linear time or branching time should be applied depends on the type of programs considered and on the properties one wishes to formalize. Linear temporal logic, for example, is more suitable (and simpler) for specifying and verifying properties that must be satisfied by all executions. However, if one is interested in formulas that must be satisfied only by some computations (due to nondeterministic choices), then branching-time logic is recommended. For a comparison between linear-time and branching-time temporal logic, see [Lam80, EH86]. Apart from linear-time and branching-time logic, there are temporal logics based on partially ordered structures, for example, the partial-order temporal logic presented in [PW84] and the temporal logic for subclasses of event structures presented in [LT87]. Partial-order temporal logics are required when properties dealing with true concurrency which cannot be expressed in linear-time or branching-time logics are to be specified [Rei88].

The temporal logics listed above are designed for quantitative reasoning about concurrent and distributed systems; they are inadequate for reasoning about time-critical systems where quantitative reasoning is required. Temporal logic has been extended so that quantitative properties can be expressed as well. A simple approach is the temporal logic of Alur and Henzinger [AH89], which uses explicit clock variables to express time constraints. Other approaches adopt temporal frames based on intervals instead of points, for example *The Interval Temporal Logic* defined by Moszkowski (see [MM84, Mos85] for hardware verification, and the *Interval Logic* of Schwarz et al. (see [SMSV83]) for protocol specification. Moreover, there exist temporal logics which deal with continuous temporal frames, such as the *The Duration Calculus* [CHR91] and *The Hybrid Temporal Logic* [KHMP94].

3.2 The Temporal Logic

The logic we wish to consider in this thesis is a temporal logic for quantitative reasoning about communication protocols. This temporal logic is an event-based linear-time logic; a time frame is considered to be an infinite, discrete and linear-ordered set of events. Looking in the literature, we find a wide variety of temporal operators for linear-time logic that can be used for specifications. In general, these operators can be defined in terms of two basic ones. For our language we adopt temporal operators taken from several sources [LPZ85, Krö87, MP91]. We consider temporal operators that refer to the past and the future. The future operators include the following unary operators:

\square (*always or henceforth*), \diamond (*eventually*), \circ (*nexttime*)

and the binary operator **until** (*until operator*). Informally, these temporal operators have the following meaning (where P and Q are again temporal or propositional formulas):

$\circ P$ P holds at the next point in time
 $\diamond P$ P holds at some future point in time (incl. the present)
 $\square P$ P holds at every future point in time (incl. the present)
 P **until** Q P holds at all following time-points up to a point in time at which Q holds

Among the past operators are symmetric counterparts to each of the future operators; they include the unary operators:

\blacksquare (*always in the past*), \blacklozenge (*once*), \bullet (*previous*)

and the binary operator **since** (*since operator*). These operators have the following informal meaning:

$\bullet P$ P held at the previous point in time
 $\blacklozenge P$ P held at some point in time in the past (incl. the present)
 $\blacksquare P$ P held throughout the past up to (and including) the present time-point
 P **since** Q P holds since the last point in time when Q held

A temporal formula which includes only future operators is called a *future formula*. A temporal formula that at most contains past operators but no future operators is called a *past formula*. Future formulas describe properties that hold at the future fragment, whereas past formulas describe properties that hold at the past fragment. The main difference between past and future is that the past fragment is considered to be finite. That is, we consider an initial point in time which corresponds to the start of a computation. This point can be syntactically characterized by the formula “ \bullet *false*”: this means that the proposition “*false*” held at the point in time preceding the initial point.

Note that as long as the past fragment is finite, past operators do not increase the expressive power of the temporal logic; this fact has been proved by Lichtenstein, Pnueli, and Zuck in

[LPZ85] where they introduced the notion of *initial equivalence* of formulas; two formulas P and Q are said to be initially equivalent if the following holds: ($\bullet \text{ false} \Rightarrow (P \Leftrightarrow Q)$). They proved that for any formula P with past operators there is a purely future formula Q , such that P and Q are initially equivalent.

Although past operators do not increase the expressive power of the specification language, their application has some advantages. There are many properties which have a more natural expression when past operators are used. For example, the fact that every P is preceded by a Q is easily expressed using the past operator (*once*): $\square(P \Rightarrow \blacklozenge Q)$. Moreover, past operators enable a smooth syntactic characterization of the basic notions of safety and liveness properties.

There follows a list of formulas which are widely used and an informal interpretation of each. They are intended to acquaint the reader with the style of expression in temporal logic. Assume that P and Q are temporal formulas:

$\square(P \Rightarrow \lozenge Q)$	Every P is followed by a Q .
$\square \lozenge P$	P holds infinitely often.
$\lozenge \square P$	Eventually permanently P ; $\neg P$ holds only finitely many times.
$\blacklozenge \bullet \text{ false}$	There is an instant in the past which has no predecessor. This is the initial point in time.
$\lozenge \blacklozenge P$	There is a point in time in the past, present, or future at which P holds.
$\square \blacksquare P$	P holds throughout the past, present, and future.
$\square \lozenge P \Rightarrow \square \lozenge Q$	An infinite occurrence of P entails an infinite occurrence of Q . This formula is often used to express a <i>strong fairness</i> property.
$\square P \Rightarrow \square \lozenge Q$	A permanent occurrence of P implies an infinite occurrence of Q . This formula expresses a <i>weak fairness</i> property.
$P \Rightarrow \circ \neg \lozenge P$	If once P then never P . Such a formula is often used to express that a message is uniquely transmitted.

If we combine temporal operators with predicates that refer to the activity of transmitting messages, we obtain formulas like:

$\lozenge [S \text{ xmt } m]$	Eventually the channel S transmits the message m .
$\square \lozenge [S \text{ xmt } m]$	The message m will be infinitely often transmitted by S .
$[T \text{ xmt } m] \Rightarrow \circ [S \text{ xmt } m]$	If the channel T transmits a message m , then the next time S transmits m .

Safety and Liveness Properties: From a methodological point of view, it is important to classify the properties one wishes to specify. Such a classification facilitates the process of checking the completeness of a specification, in the sense that all (informal) requirements

are formulated in the specification [MP89]. One important classification of properties of systems distinguishes between *safety* and *liveness* properties. An intuitive definition of safety and liveness properties has been given by Lamport [Lam77]: a *safety property* states that something bad never happens; a *liveness property* states that some good thing eventually happens. Many attempts have been made to give a formal definition of safety and liveness properties. A characterization based on semantic models can be found in [Lam85] and [AS85]. In the context of temporal logic, safety and liveness properties can also be syntactically characterized. Siestla defined safety properties in [Sis85] as formulas that can be constructed from atomic formulas by applying positive boolean connections (\wedge , \vee) and the temporal operators \square and **unless** (*weak until*). He also gave a characterization of a subclass of liveness properties which does not, however, apply to the full class. A significant syntactical characterization of the safety and liveness classes is given by Lichtenstein, Pnueli, and Zuck in [LPZ85]. They suggest the following classification:

- A formula represents a *safety* property iff it is initially equivalent to a formula of the form:

$$\square P \quad \text{for some past formula } P$$

- A formula represents a *liveness* property if it is representable in one of the forms:

$$\diamond P, \quad \square \diamond P, \quad \diamond \square P \quad \text{for some past formula } P$$

Note that any positive boolean combination of such formulas again yields safety and liveness formulas, respectively.

However, the various formal definitions of safety and liveness properties are not all equivalent. Comparing, for example, the characterization given above to the one given in [AS85], we observe that the definition of the liveness properties according to [AS85] is stronger than the definition given above. For instance, in [LPZ85], the property (P **until** Q) is considered a liveness property, while according to [AS85], it represents neither a safety nor a liveness property. Here, we adopt the classification proposed in [LPZ85].

In the following, we list some safety and liveness properties often used in the specification of communication protocols. According to the definition in [LPZ85], the formula

$$([A \text{ snd } m] \Rightarrow \blacklozenge[A \text{ rcv } m])$$

is considered a safety property because it is representable as $\square([A \text{ snd } m] \Rightarrow \blacklozenge[A \text{ rcv } m])$. It states that the agent A only sends what it has received,

The following formula is also considered a safety property stating that the agent A sends messages in the same order as they have been received:

$$(\blacklozenge[A \text{ snd } m] \wedge [A \text{ snd } n]) \Rightarrow \blacklozenge(\blacklozenge[A \text{ rcv } m] \wedge [A \text{ rcv } n])$$

According to [LPZ85], the formula ($[A \text{ rcv } m] \Rightarrow \diamond[A \text{ snd } m]$), which claims that the agent A eventually sends what it has received, is considered a liveness property, because it is representable as $(\diamond \neg[A \text{ rcv } m] \vee \square \diamond[A \text{ snd } m])$. Another liveness property is the formula $(\square \diamond[A \text{ rcv }])$, claiming that the agent A accepts an infinite number of messages.

In our specifications, we will distinguish between safety and liveness properties, and consider a protocol specification as a combination of properties of both classes.

3.3 Indexing Temporal Operators

Temporal logic in its elementary form does not support the development of systems in a *modular* style. The reason is that temporal logic describes systems in their entirety [BKP84]; in a specification, temporal formulas refer unrestrictedly to all processes in the system. Thus, whenever a new process is added to the system all existing formulas should apply to this process, and vice versa, all formulas specifying the new process should be satisfied by the existing system as well. Consider, for example, the following two agents A and B with input channels S and U and output channels R and V , respectively.



We specify A and B as two buffers, each of capacity one. That is, each agent contains at most one message at a time. Hence, each agent should deliver an already received message before receiving the next one. Obviously, the send and receive actions of each agent alternate. This leads to the following specifications:

Properties of A:

$$[A \text{ rcv } m] \Rightarrow \circ [A \text{ snd } m]$$

$$[A \text{ snd }] \Rightarrow \circ [A \text{ rcv }]$$

Properties of B:

$$[B \text{ rcv } m] \Rightarrow \circ [B \text{ snd } m]$$

$$[B \text{ snd }] \Rightarrow \circ [B \text{ rcv }]$$

Clearly, one can find implementations that satisfy the properties of the agents A and B , respectively, provided that they work separately. However, if we link the input of B to the output of A , we may have some problems, because the composite system should satisfy the following formula:

$$[A \text{ rcv } m] \Rightarrow \circ \circ ([A \text{ rcv }] \wedge [B \text{ snd } m])$$

In fact, this formula can never be satisfied, because only one action can be performed at one point in time (see Section 3.3.1).

Several attempts have been made to extend temporal logic in order to achieve modularity, such as in [BKP84], [NGO86], and [AL89]. These extensions depend, in general, on the models on which the temporal logic is based. For example, in the approach presented in [BKP84], Barringer et al. introduced auxiliary variables in order to be able to distinguish between actions performed by the system and those performed by the environment.

In our approach, we make the temporal logic modular by indexing the temporal operators with (finite) sets of channel names, where such a set of channels corresponds to a specific part of a system. In other words, temporal operators are constrained to specific parts of a system's behavior. An indexed temporal formula is interpreted as referring only to the part defined by its index. Thus, a system specification may contain formulas which need only be satisfied by selected components, rather than by all components during a system run. Consequently, the composition of system components is achieved trivially by conjunction: this will be proved later in Section 5.2.

Let M be a subsystem (a set of channels) and P be a temporal formula. We say M is *active* at a point in time if a transmission action is taking place within the system M . Informally, our indexed temporal operators have the following meaning:

$\bigcirc_M P$ P holds at the next point in time at which M will be active.

$\square_M P$ P holds at every future point in time at which M is active (incl. the present).

$\diamond_M P$ P holds at some future point in time at which M is active (incl. the present).

$P \text{ }_M\text{ until}_N Q$ P holds at all following points in time at which M is active up to a point in time at which Q holds on N .

The (indexed) past operators have analogous meanings to the ones given above for the (indexed) future operators.

For reason of greater uniformity, we introduce an operator Δ_M (*possibly now on M*), which refers explicitly to the present. Informally, the formula $\Delta_M P$ means that P holds now if the subsystem M is active. Obviously, this is a weak operator, because it does not entail that P holds now. Thus, we add to our temporal language a strong version of this operator: \mathbb{A}_M (*now on M*). The formula $\mathbb{A}_M P$ means that M is now active and P holds. These two operators offer the possibility of formulating the propositions “the subsystem M is inactive” and “the subsystem M is currently active” in a simple and nice way. They can be formulated by $\Delta_M \text{ false}$ and $\mathbb{A}_M \text{ true}$, respectively.

We omit the index if it includes all channel names in the system. In this case, temporal operators refer implicitly to the whole system behavior and are thus equivalent to the classical operators presented in Section 3.2. Moreover, if the index is a single item we omit the brackets, thus, we write $\diamond_R P$ instead of $\diamond_{\{R\}} P$, where R is a channel name.

To enhance readability, we prefer to write $\square_R [R \text{ xmt } m]$ instead of $\square_R [\text{ xmt } m]$, although the former contains redundancy, because the formula $\square_R [R \text{ xmt } m]$ can only be satisfied if R is transmitting; the formula $\square_R [S \text{ xmt } m]$ can never be satisfied.

It is important not to confuse the formula $\square [S \text{ xmt } m]$ with $\square_S [S \text{ xmt } m]$. The former means that the activity “ S transmits m ” takes place at every future point in time, whereas the formula $\square_S [S \text{ xmt } m]$ says that whenever S is transmitting, it transmits the message m .

3.3.1 Semantics

As mentioned in Section 2.4, a *model* $\mathcal{M} = (\text{Alg}, (\mathcal{E}, \leq))$ for our temporal formulas consists of an algebra Alg , and a linear-ordered set (\mathcal{E}, \leq) of events, where each event represents the occurrence of a send, receive, or a transmission action during a system run. A linear ordering (\mathcal{E}, \leq) represents a time frame in which each event is considered as an atomic point in time. Every point in time represents a world (in the sense of Kripke) in which atomic formulas can be interpreted.

Before giving the semantics of the temporal language, we will introduce some abbreviations.

Let (\mathcal{E}, \leq) be a linear ordering, S a channel name and M be a set of channel names.

$$\begin{aligned}\mathcal{E}_S &\stackrel{\text{def}}{=} \{e \in \mathcal{E} \mid \text{there exists a message } m \text{ s.t. } e = (S, m)\} \\ \mathcal{E}_M &\stackrel{\text{def}}{=} \bigcup_{S \in M} \mathcal{E}_S\end{aligned}$$

That is, \mathcal{E}_S denotes the subset of all those events that concern the channel S , and \mathcal{E}_M is the corresponding generalization to sets of channels.

The interpretation of temporal formulas containing free variables depends on an *assignment* α which assigns a value to each message variable. We denote by $\mathcal{M}, e \models_\alpha P$ the fact that the temporal formula P is valid for the model \mathcal{M} at the point e w.r.t. the assignment α . If P is a closed formula we omit α , because the validity of P does not depend on an assignment. This leads to the following semantic definition of the temporal operators:

- **Present Operators**

$$\begin{aligned}\mathcal{M}, e \models \Delta_M P &\quad \underline{\text{iff}} \quad e \in \mathcal{E}_M \text{ and } \mathcal{M}, e \models P \\ \mathcal{M}, e \models \triangle_M P &\quad \underline{\text{iff}} \quad e \in \mathcal{E}_M \text{ implies } \mathcal{M}, e \models P\end{aligned}$$

Obviously, the operators Δ and \triangle are dual, that is for any set of channels M and for any formula P , $\Delta_M P \equiv \neg \triangle_M \neg P$ holds, and vice versa.

- **Future Operators**

$$\begin{aligned}\mathcal{M}, e \models \circ_M P &\quad \underline{\text{iff}} \quad \mathcal{M}, e' \models P \text{ for the least } e' \in \mathcal{E}_M \text{ with } e < e' \\ &\quad \text{if such } e' \text{ exists, otherwise the formula is valid} \\ \mathcal{M}, e \models \square_M P &\quad \underline{\text{iff}} \quad \text{for every } e' \in \mathcal{E}_M \text{ with } e \leq e', \mathcal{M}, e' \models P \\ \mathcal{M}, e \models \diamond_M P &\quad \underline{\text{iff}} \quad \text{there exists } e' \in \mathcal{E}_M \text{ with } e \leq e' \text{ and } \mathcal{M}, e' \models P\end{aligned}$$

$$\begin{aligned}\mathcal{M}, e \models P \text{ }_M \text{until}_N Q &\quad \underline{\text{iff}} \quad \text{for some } e' \in \mathcal{E}_N \text{ with } e \leq e', \mathcal{M}, e' \models Q \text{ and} \\ &\quad \text{for every } e'' \in \mathcal{E}_M \text{ with } e \leq e'' < e', \mathcal{M}, e'' \models P\end{aligned}$$

- **Past Operators**

$$\begin{aligned}\mathcal{M}, e \models \bullet_M P &\quad \underline{\text{iff}} \quad \mathcal{M}, e' \models P \text{ for the greatest } e' \in \mathcal{E}_M \text{ with } e' < e \\ &\quad \text{if such } e' \text{ exists, otherwise the formula is valid} \\ \mathcal{M}, e \models \blacksquare_M P &\quad \underline{\text{iff}} \quad \text{for every } e' \in \mathcal{E}_M \text{ with } e' \leq e, \mathcal{M}, e' \models P \\ \mathcal{M}, e \models \blacklozenge_M P &\quad \underline{\text{iff}} \quad \text{there exists } e' \in \mathcal{E}_M \text{ with } e' \leq e \text{ and } \mathcal{M}, e' \models P\end{aligned}$$

$$\begin{aligned}\mathcal{M}, e \models P \text{ }_M \text{since}_N Q &\quad \underline{\text{iff}} \quad \text{for some } e' \in \mathcal{E}_N \text{ with } e' \leq e, \mathcal{M}, e' \models Q \text{ and} \\ &\quad \text{for every } e'' \in \mathcal{E}_M \text{ with } e' < e'' \leq e, \mathcal{M}, e'' \models P\end{aligned}$$

- **Atomic formulas** are predicates that refer to the activities of transmitting messages. These predicates are defined by

$$\begin{aligned} \mathcal{M}, e \models_{\alpha} [S \mathbf{xmt} m] & \quad \text{iff} \quad e = (S, \alpha(m)) \\ \mathcal{M}, e \models [S \mathbf{xmt}] & \quad \text{iff} \quad \text{there is } v \in Alg \text{ such that } e = (S, v) \end{aligned}$$

Actually, the indexed next-time operator defined above is a weak operator, because $\circ_M P$ does not entail that P eventually happens; $\circ_M P$ is satisfied in a sequence of events in which M will never be active. However, it is useful to also have a next-time operator which guarantees that the desired action will eventually occur. The same holds for the previous operator. For this reason we introduce the operators $\tilde{\circ}$ (*strong next-time*) and $\tilde{\bullet}$ (*strong previous*) defined as:

$$\begin{aligned} \tilde{\circ}_M P & \stackrel{\text{def}}{\iff} \neg \circ_M \neg P \\ \tilde{\bullet}_M P & \stackrel{\text{def}}{\iff} \neg \bullet_M \neg P \end{aligned}$$

From this definition it follows immediately:

$$\begin{aligned} \tilde{\circ}_M P & \Rightarrow \circ_M P \wedge \diamond_M P \\ \tilde{\bullet}_M P & \Rightarrow \bullet_M P \wedge \blacklozenge_M P \end{aligned}$$

Apart from these operators, we add the unary operators \blacklozenge (*later* or *strict eventually*), and the binary operators **before** and **unless** (*weak until*), which will be frequently used in our specifications; they are defined as:

$$\begin{aligned} \blacklozenge_M P & \stackrel{\text{def}}{\iff} \circ \diamond_M P \\ P_M \mathbf{unless}_N Q & \stackrel{\text{def}}{\iff} \square_M P \vee P_M \mathbf{until}_N Q \\ P_M \mathbf{before}_N Q & \stackrel{\text{def}}{\iff} (\neg Q)_N \mathbf{unless}_M P \end{aligned}$$

A temporal formula is valid in the model \mathcal{M} , denoted by $\mathcal{M} \models P$, if $\mathcal{M}, e \models P$ for every $e \in \mathcal{E}$. P is valid, denoted by $\models P$, if $\mathcal{M} \models P$ for every model \mathcal{M} . The validity of a formula P is thus defined by requiring that P holds at all points in time in all models. This kind of interpretation yields the so-called *floating* version of temporal logic. There is another version called *anchored* (see [MP89]), in which the validity of a formula depends only on its interpretation at the initial point in time.

It should be stressed that indexing temporal operators does not increase the expressive power of the original temporal language; indexing is no more than syntactic sugar to achieve more structured specifications, yet it is also a simple route to modularity. We have, for example,

$$\begin{aligned} \square_M P & \Leftrightarrow \square(\mathit{active}(M) \Rightarrow P) \\ \diamond_M P & \Leftrightarrow \diamond(\mathit{active}(M) \wedge P) \\ \tilde{\circ}_M P & \Leftrightarrow \circ(\mathit{inactive}(M) \mathbf{until} (\mathit{active}(M) \wedge P)) \end{aligned}$$

where $\mathit{active}(M)$ is an abbreviation for the formula $(\bigvee_{S \in M} [S \mathbf{xmt}])$, stating that at least one of the streams of the subsystem M is transmitting. As shown above, $\mathit{active}(M)$ can be formulated by the temporal formula $\Delta_M \mathit{true}$. The predicate $\mathit{inactive}(M)$ stands for $\neg \mathit{active}(M)$, which can be formulated by $\Delta_M \mathit{false}$.

3.4 The Proof System

In this section we present a sound and complete proof system for the propositional part of the temporal logic. The proof system consists of three parts, concerning present, future, and past formulas. Restricted to the future and past parts, our rules and axioms are indexed versions of the general proof system presented in [LPZ85]. In our proof system, we consider (implicitly) axioms and theorems taken from the theory of sets. These axioms and theorems are needed in order to be able to derive formulas such as $\Box_M P \Rightarrow \Box_N P$, if N is a subset of M .

- **Present Part**

- *Rule*

$$(\text{mp}) : \frac{P, P \Rightarrow Q}{Q}$$

- *Axioms*

Ax1: All tautologies of the propositional logic.

Ax2: $\Delta_M P \Leftrightarrow \neg \Delta_M \neg P$

Ax3: $\Delta_M P \Rightarrow \Delta_M P$

Ax4: $\Delta_M P \wedge \Delta_N P \Leftrightarrow \Delta_{M \cup N} P$

Ax5: $*_M \Delta_M P \Leftrightarrow *_M P$ for $*$ $\in \{ \tilde{\square}, \circ, \bullet, \blacklozenge, \Delta, \triangle, \square, \diamond \}$

As mentioned in Section 3.3, the operators Δ and \triangle are introduced for technical reasons. Axiom (**Ax5**) states that they can be eliminated from most formulas. Although there are (complete) proof systems for classical propositional logic, in axiom (**Ax1**) we consider all tautologies of this logic, because we are only interested in proving temporal theorems.

- **Future Part**

- *Rule*

$$(\square\text{-Gen}) : \frac{P}{\Box_M P}$$

This rule states that if P is derivable, then so is $\Box_M P$ for an arbitrary set M . Consequently, the *deduction theorem* from classical propositional logic, claiming that $(P \vdash Q \text{ iff } \vdash P \Rightarrow Q)$, is not valid, because $P \Rightarrow \Box_M P$ is not a valid formula.

– *Axioms*

- FAx1:** $\tilde{\circ}_M P \Leftrightarrow \neg \circ_M \neg P$
FAx2: $\Delta_M P \Rightarrow \circ_M \bullet_M P$
FAx3: $\tilde{\circ}_M P \Rightarrow \circ_M P$
FAx4: $\circ_M (P \Rightarrow Q) \Rightarrow (\circ_M P \Rightarrow \circ_M Q)$
FAx5: $\circ_M P \Leftrightarrow \circ (\Delta_M P \wedge (\Delta_M \text{false} \Rightarrow \circ_M P))$
FAx6: $\diamond_M P \Leftrightarrow \neg \square_M \neg P$
FAx7: $\square_M P \Rightarrow (\Delta_M P \wedge \circ_M \square_M P)$
FAx8: $\square_M (P \Rightarrow Q) \Rightarrow (\square_M P \Rightarrow \square_M Q)$
FAx9: $\square (P \Rightarrow \circ_M P) \Rightarrow (P \Rightarrow \square_M P)$
FAx10: $P_M \text{until}_N Q \Leftrightarrow \Delta_N Q \vee (\Delta_M P \wedge \circ (P_M \text{until}_N Q))$
FAx11: $P_M \text{until}_N Q \Rightarrow \diamond_N Q$

• **Past Part**

– *Rule*

$$(\blacksquare\text{-Gen}) : \frac{P}{\blacksquare_M P}$$

– *Axioms*

- PAx1:** $\bullet_M P \Leftrightarrow \neg \blacklozenge_M \neg P$
PAx2: $\Delta_M P \Rightarrow \bullet_M \tilde{\circ}_M P$
PAx3: $\bullet_M P \Rightarrow \bullet_M P$
PAx4: $\bullet_M (P \Rightarrow Q) \Rightarrow (\bullet_M P \Rightarrow \bullet_M Q)$
PAx5: $\bullet_M P \Leftrightarrow \bullet (\Delta_M P \wedge (\Delta_M P \Rightarrow \bullet_M P))$
PAx6: $\blacklozenge_M P \Leftrightarrow \neg \blacksquare_M \neg P$
PAx7: $\blacksquare_M P \Rightarrow (\Delta_M P \wedge \bullet_M \blacksquare_M P)$
PAx8: $\blacksquare_M (P \Rightarrow Q) \Rightarrow (\blacksquare_M P \Rightarrow \blacksquare_M Q)$
PAx9: $\blacksquare (P \Rightarrow \bullet_M P) \Rightarrow (P \Rightarrow \blacksquare_M P)$
PAx10: $P_M \text{since}_N Q \Leftrightarrow \Delta_N Q \vee (\Delta_M P \wedge \bullet (P_M \text{since}_N Q))$
PAx11: $\blacklozenge \bullet \text{false}$

The finiteness of the past can be shown by axiom (**PAx11**), which states that the initial point is always reachable.

It should be noted that an axiom, such as $(\diamond_M P \Leftrightarrow \neg \square_M \neg P)$, represents an axiom scheme that holds for any formula P and any index M ; so P can be substituted by any temporal formula and M can be replaced by an arbitrary set of channels. In particular, M can be considered as the set of all channels, and then the formula $(\diamond P \Leftrightarrow \neg \square \neg P)$ becomes an axiom. This entails that the proof system presented in [LPZ85] is included in our proof system, and the rules and axioms there are special cases of our rules and axioms.

The proof system presented above is sound and complete with respect to the semantics given in the last section. The validity of the axioms and rules can be trivially checked from the semantics of the temporal operators. In general, the proof of completeness is more complicated. Fortunately, the completeness of our proof system can be derived from the completeness of the proof system presented in [LPZ85].

If we consider a temporal logic without indexing – in actual fact, a special case of our logic –, we realize that it is isomorphic to the logic presented in [LPZ85]. A complete proof system for this logic is also given in [LPZ85] and, as shown above, this proof system is included in our proof system. Moreover, since indexing temporal operators does not increase the expressive power of the language, each indexed temporal formula can be transformed into an equivalent non-indexed one, e.g.

$$\diamond_M P \equiv \diamond(\text{active}(M) \wedge P)$$

Let P be an indexed temporal formula and \tilde{P} be its non-indexed form. Then we deduce:

$$\begin{array}{ll} \models P & \text{implies } \models_{lpz} \tilde{P} \\ \models_{lpz} \tilde{P} & \text{implies } \vdash_{lpz} \tilde{P} \quad (\text{because of the completeness of the proof system in [LPZ85]}) \\ \vdash_{lpz} \tilde{P} & \text{implies } \vdash P \quad (\text{because our proof system includes the system in [LPZ85]}) \end{array}$$

(\models_{lpz} and \vdash_{lpz} respectively denote validity and derivability in the non-indexed temporal logic). $\vdash \tilde{P}$ entails $\vdash P$. Hence, we establish the completeness of our proof system.

3.5 Proving Temporal Rules and Theorems

In this section, we apply the proof system presented in Section 3.4 to derive some theorems and rules. A wide variety of properties are satisfied by the temporal operators: properties of *duality*, *idempotency*, *distributivity*, etc. Many of these laws have been investigated and (partially) proved for classical temporal logic (see, e.g. [Kr87] and [MP91]). In the following, we investigate some of these laws for the indexed version of temporal logic.

A proof of a theorem is presented as a sequence of lines, where each line consists of either an axiom (or proved theorem) or an application of a rule (or proved rule). Further, each line includes a justification indicating which axiom or which rule has been applied. A proof of a rule may additionally include lines which list one of its premises. Furthermore, in our proofs we will use the following rule taken from classical temporal logic.

$$\text{(prop): } \frac{P_1 \wedge P_2 \wedge \dots \wedge P_n \Rightarrow Q}{P_1, P_2, \dots, P_n \vdash Q}$$

This rule states that if $P_1 \wedge P_2 \wedge \dots \wedge P_n \Rightarrow Q$ is a tautology in classical propositional logic, then we take $P_1, P_2, \dots, P_n \vdash Q$ as a derivation rule. For example, the formula $(P \Rightarrow Q \wedge P \Rightarrow R) \Rightarrow (P \Rightarrow Q \wedge R)$ is a tautology in classical propositional logic, so we can apply the derivation rule $P \Rightarrow Q, P \Rightarrow R \vdash P \Rightarrow Q \wedge R$ in our proofs.

Monotonicity Rules

All our indexed unary and binary temporal operators are monotonic.

- \square_M is monotonic,

$$\text{(\square}_M\text{-Mon): } \frac{P \Rightarrow Q}{\square_M P \Rightarrow \square_M Q} \qquad \text{(\square}_M\text{-Mon'): } \frac{P \Leftrightarrow Q}{\square_M P \Leftrightarrow \square_M Q}$$

In the following we derive $(\Box_M\text{-Mon})$. $(\Box_M\text{-Mon}')$ follows immediately from $(\Box_M\text{-Mon})$.

Proof :

1. $P \Rightarrow Q$ [premise]
2. $\Box_M (P \Rightarrow Q)$ [\Box -Gen]
3. $\Box_M (P \Rightarrow Q) \Rightarrow (\Box_M P \Rightarrow \Box_M Q)$ [FAx8]
4. $\Box_M P \Rightarrow \Box_M Q$ [mp]

■

- \Diamond_M is monotonic,

$$(\Diamond_M\text{-Mon}): \frac{P \Rightarrow Q}{\Diamond_M P \Rightarrow \Diamond_M Q} \quad (\Diamond_M\text{-Mon}'): \frac{P \Leftrightarrow Q}{\Diamond_M P \Leftrightarrow \Diamond_M Q}$$

Proof of $(\Diamond_M\text{-Mon})$:

1. $P \Rightarrow Q$ [premise]
2. $\neg Q \Rightarrow \neg P$ [prop]
3. $\Box_M \neg Q \Rightarrow \Box_M \neg P$ [\Box -Mon]
4. $\neg \Box_M \neg P \Rightarrow \neg \Box_M \neg Q$ [prop]
5. $\Diamond_M P \Rightarrow \Diamond_M Q$ [FAx6]

■

- In the same way we can prove that all other temporal operators are monotonic.

Duality Theorems

Each of the unary operators has a dual.

- \triangleleft_M and \triangle_M are dual.

$$(\text{Dual1}): \neg \triangleleft_M P \Leftrightarrow \triangle_M \neg P \quad (\text{Dual1}'): \neg \triangle_M P \Leftrightarrow \triangleleft_M \neg P$$

Proof of (Dual1):

1. $P \Leftrightarrow \neg(\neg P)$ [prop]
2. $\triangleleft_M P \Leftrightarrow \triangleleft_M \neg(\neg P)$ [\triangleleft_M -Mon]
3. $\neg \triangleleft_M P \Leftrightarrow \neg \triangleleft_M \neg(\neg P)$ [prop]
4. $\neg \triangleleft_M P \Leftrightarrow \triangle_M \neg P$ [Ax2]

■

- Further, $\tilde{\bigcirc}_M$ is dual to \bigcirc_M , $\tilde{\diamond}_M$ is dual to \square_M , $\tilde{\bullet}_M$ is dual to \bullet_M , and $\tilde{\blacklozenge}_M$ is dual to \blacksquare_M , and vice versa. We have:

$$\begin{array}{ll}
(\text{Dual2}) : \neg \tilde{\bigcirc}_M P \Leftrightarrow \bigcirc_M \neg P & (\text{Dual2}') : \neg \bigcirc_M P \Leftrightarrow \tilde{\bigcirc}_M \neg P \\
(\text{Dual3}) : \neg \tilde{\diamond}_M P \Leftrightarrow \square_M \neg P & (\text{Dual3}') : \neg \square_M P \Leftrightarrow \tilde{\diamond}_M \neg P \\
(\text{Dual4}) : \neg \tilde{\bullet}_M P \Leftrightarrow \bullet_M \neg P & (\text{Dual4}') : \neg \bullet_M P \Leftrightarrow \tilde{\bullet}_M \neg P \\
(\text{Dual5}) : \neg \tilde{\blacklozenge}_M P \Leftrightarrow \blacksquare_M \neg P & (\text{Dual5}') : \neg \blacksquare_M P \Leftrightarrow \tilde{\blacklozenge}_M \neg P
\end{array}$$

Introduction Rules

The introduction rules enable us to apply the principle of induction for proving temporal properties. For example, the rule (\square_M -Int) states that in order to prove $Q \Rightarrow \square_M P$, it suffices to derive $Q \Rightarrow \triangle_M P$ and $Q \Rightarrow \bigcirc_M Q$.

- Introduction Rule for \square_M :

$$(\square_M\text{-Int}): \frac{Q \Rightarrow \triangle_M P, \quad Q \Rightarrow \bigcirc_M Q}{Q \Rightarrow \square_M P}$$

Proof :

$$\begin{array}{ll}
1. Q \Rightarrow \bigcirc_M Q & [\text{premise}] \\
2. \square(Q \Rightarrow \bigcirc_M Q) & [\square\text{-Gen}] \\
3. Q \Rightarrow \square_M Q & [\text{mp, FAx9}] \\
4. \square_M Q \Rightarrow \square_M \triangle_M P & [(\square_M\text{-Mon}), \text{premise}] \\
5. \square_M \triangle_M P \Rightarrow \square_M P & [\text{Ax5}] \\
6. Q \Rightarrow \square_M P & [(\text{mp})]
\end{array}$$

■

Note that $Q \Rightarrow \square_M P$ can also be proved if we have in the premises $Q \Rightarrow \bigcirc Q$ instead of $Q \Rightarrow \bigcirc_M Q$.

- Introduction Rule for \diamond_M :

$$(\diamond_M\text{-Int}): \frac{(\triangle_M P \vee \tilde{\bigcirc}_M Q) \Rightarrow Q}{\diamond_M P \Rightarrow Q}$$

This rule can be proved by duality to (\square_M -Int):

- Introduction Rule for \bigcirc_M :

$$(\bigcirc_M\text{-Int}): \frac{Q \Rightarrow \bigcirc(\triangle_M P \vee (\triangle_M \text{false} \Rightarrow Q))}{Q \Rightarrow \bigcirc_M P}$$

- Introduction Rule for ${}_M \mathbf{unless}_N$:

$$({}_M \mathbf{unless}_N\text{-Int}): \frac{R \Rightarrow (\triangle_N Q \vee (\triangle_M P \wedge \bigcirc R))}{R \Rightarrow P {}_M \mathbf{unless}_N Q}$$

- Similar introduction rules can be given for \blacksquare_M , \blacklozenge_M , and \bullet_M .

Idempotency Theorems

Apart from the “next-time” and “previous” operators, all other unary temporal operators are idempotent. That is, applying an operator twice yields the same result as would a single application.

- Idempotence of \Box_M :

$$(\Box_M\text{-Idem}): \quad \Box_M \Box_M P \Leftrightarrow \Box_M P$$

Proof :

1. $\Box_M P \Rightarrow \Delta_M P$ [FAx7]
2. $\Box_M \Box_M P \Rightarrow \Box_M (\Delta_M P)$ [\Box_M -Mon]
3. $\Box_M \Box_M P \Rightarrow \Box_M P$ [Ax5]
5. $\Box_M P \Rightarrow \bigcirc_M \Box_M P$ [FAx7]
6. $\Box (\Box_M P \Rightarrow \bigcirc_M \Box_M P)$ [\Box -Gen]
7. $\Box (\Box_M P \Rightarrow \bigcirc_M \Box_M P) \Rightarrow (\Box_M P \Rightarrow \Box_M \Box_M P)$ [FAx9]
8. $\Box_M P \Rightarrow \Box_M \Box_M P$ [mp]
9. $\Box_M P \Leftrightarrow \Box_M \Box_M P$ [prop]

■

- Idempotence of \Diamond_M :

$$(\Diamond_M\text{-Idem}): \quad \Diamond_M \Diamond_M P \Leftrightarrow \Diamond_M P$$

Proof :

1. $\Box_M \neg P \Leftrightarrow \Box_M \Box_M \neg P$ [\Box_M -Idem]
2. $\neg \Box_M \neg P \Leftrightarrow \neg \Box_M \Box_M \neg P$ [prop]
3. $\Diamond_M P \Leftrightarrow \neg \Box_M (\neg \neg \Box_M \neg P)$ [Fax6]
4. $\Diamond_M P \Leftrightarrow \neg \Box_M \neg \Diamond_M P$
5. $\Diamond_M P \Leftrightarrow \Diamond_M \Diamond_M P$

■

- In the same way we can prove that Δ_M , \triangleleft_M , \blacksquare_M , and \blacklozenge_M are idempotent.
- Moreover, we can prove similar properties for the ${}_M\mathbf{until}_N$ operator, i.e. if we apply it twice we end up with the original formula.

$$({}_M\mathbf{until}_N\text{-Idem}): \quad (P {}_M\mathbf{until}_N Q) {}_M\mathbf{until}_N Q \Leftrightarrow P {}_M\mathbf{until}_N Q$$

$$({}_M\mathbf{until}_N\text{-Idem}') : \quad P {}_M\mathbf{until}_N (P {}_M\mathbf{until}_N Q) \Leftrightarrow P {}_M\mathbf{until}_N Q$$

- Analogous theorems can be proven for ${}_M\mathbf{since}_N$.

Reflexibility Theorems

Most temporal logics consider the present as part of both the future and the past; the reflexivity law ($\Box P \Rightarrow P$) is often taken as an axiom. In our approach, we have the following theorems:

- \Box_M , \Diamond_M , \blacksquare_M and \blacklozenge_M are reflexive.

$$\begin{aligned} (\Box_M\text{-Ref}): & \quad \Box_M P \Rightarrow \Delta_M P \\ (\Diamond_M\text{-Ref}): & \quad \Delta_M P \Rightarrow \Diamond_M P \\ (\blacksquare_M\text{-Ref}): & \quad \blacksquare_M P \Rightarrow \Delta_M P \\ (\blacklozenge_M\text{-Ref}): & \quad \Delta_M P \Rightarrow \blacklozenge_M P \end{aligned}$$

(\Box_M -Ref) and (\blacksquare_M -Ref) follow immediately from axioms (**FAx**) and (**PAx**), respectively. (\Diamond_M -Ref) and (\blacklozenge_M -Ref) follow by duality.

Expressibility Theorems

Considering linear-time logic, the temporal operators can be expressed in terms of two basic operators (*until* and *next-time*). This is also the case for indexed temporal operators.

- For the future operators we have the following theorems:

$$\begin{aligned} (\text{Ex1}): & \quad \Box_M P \Leftrightarrow P_M \mathbf{unless} \text{ false} \\ (\text{Ex2}): & \quad \Diamond_N P \Leftrightarrow \text{true} \mathbf{until}_N P \\ (\text{Ex3}): & \quad \widetilde{\circ}_M P \Leftrightarrow \circ(\text{false}_M \mathbf{until}_M P) \\ (\text{Ex4}): & \quad \circ_M P \Leftrightarrow \circ(\text{false}_M \mathbf{unless}_M P) \end{aligned}$$

We prove (Ex1):

Proof :

$$\begin{aligned} 1. & \quad P_M \mathbf{unless} \text{ false} \Rightarrow \Box_M P && \text{[Def of } \mathbf{unless}] \\ 2. & \quad \Box_M P \Rightarrow \Delta_M P \wedge \circ_M \Box_M P && \text{[FAx7]} \\ 3. & \quad \Box_M P \Rightarrow \Delta \text{false} \vee (\Delta_M P \wedge \circ_M \Box_M P) && \text{[prop]} \\ 4. & \quad \Box_M P \Rightarrow P_M \mathbf{unless} \text{ false} && \text{[} \mathbf{unless}\text{-Int]} \\ 5. & \quad \Box_M P \Leftrightarrow P_M \mathbf{unless} \text{ false} \end{aligned}$$

■

- Similar theorems hold for the past operators.

Recursion Theorems

The following theorems give a “kind” of a recursive definition for the temporal operators. For example, (Rec1) is a formulation of the following informal representation:

$$\Box_M P \Leftrightarrow \Delta_M P \wedge \circ_M P \wedge \circ_M \circ_M P \wedge \circ_M \circ_M \circ_M P \wedge \dots$$

- For the future operators we have the following theorems:

$$\begin{aligned}
(\text{Rec1}): \quad & \Box_M P \Leftrightarrow \Delta_M P \wedge \circ_M \Box_M P \\
(\text{Rec2}): \quad & \Diamond_M P \Leftrightarrow \Delta_M P \vee \tilde{\circ}_M \Diamond_M P \\
(\text{Rec3}): \quad & \tilde{\circ}_M P \Leftrightarrow \circ (\Delta_M P \vee (\Delta_M \text{false} \Rightarrow \tilde{\circ}_M P)) \\
(\text{Rec4}): \quad & P_M \mathbf{until}_N Q \Leftrightarrow \Delta_M Q \vee (\Delta_M P \wedge \circ (P_M \mathbf{until}_N Q))
\end{aligned}$$

We prove (Rec1).

Proof :

$$\text{Set } Q = \Delta_M P \wedge \circ_M \Box_M P$$

$$\begin{array}{ll}
1. \Box_M P \Rightarrow Q & [\text{FAx7}] \\
2. Q \Rightarrow \Delta_M P & [\text{prop}] \\
3. \circ_M \Box_M P \Rightarrow \circ_M Q & [\tilde{\circ}_M\text{-Mon}] \\
4. \Delta_M P \wedge \circ_M \Box_M P \Rightarrow \circ_M Q & [\text{prop}] \\
5. Q \Rightarrow \circ_M Q & [\text{def}] \\
6. Q \Rightarrow \Delta_M P \wedge \circ_M Q & [\text{prop}] \\
7. Q \Rightarrow \Box_M P & [\Box_M\text{-Int}] \\
8. Q \Leftrightarrow \Box_M P &
\end{array}$$

■

(Rec2) can be proven by duality. (Rec3) and (Rec4) are axioms.

- Analogous recursion theorems hold for the past operators \blacksquare_M , \blacklozenge_M , $\tilde{\bullet}_M$, and $M\mathbf{since}_N$.

Distributivity Theorems

The *always* operator distributes over conjunction and implication, but not over disjunction. The *eventually* operator, on the other hand, distributes over disjunction, but not over conjunction. The *next-time* operators, in contrast, distribute over all boolean connections.

- For future operators we have the following distributivity laws:

$$\begin{aligned}
(\text{Dist1}): \quad & \Delta_M (P \wedge Q) \Leftrightarrow \Delta_M P \wedge \Delta_M Q \\
(\text{Dist1}): \quad & \Delta_M (P \vee Q) \Leftrightarrow \Delta_M P \vee \Delta_M Q \\
(\text{Dist2}): \quad & \Delta_M (P \wedge Q) \Leftrightarrow \Delta_M P \wedge \Delta_M Q \\
(\text{Dist3}): \quad & \Delta_M (P \vee Q) \Leftrightarrow \Delta_M P \vee \Delta_M Q \\
(\text{Dist4}): \quad & \Box_M (P \wedge Q) \Leftrightarrow \Box_M P \wedge \Box_M Q \\
(\text{Dist5}): \quad & \Diamond_M (P \vee Q) \Leftrightarrow \Diamond_M P \vee \Diamond_M Q \\
(\text{Dist6}): \quad & \tilde{\circ}_M (P \wedge Q) \Leftrightarrow \tilde{\circ}_M P \wedge \tilde{\circ}_M Q \\
(\text{Dist7}): \quad & \tilde{\circ}_M (P \vee Q) \Leftrightarrow \tilde{\circ}_M P \vee \tilde{\circ}_M Q \\
(\text{Dist8}): \quad & \circ_M (P \wedge Q) \Leftrightarrow \circ_M P \wedge \circ_M Q \\
(\text{Dist9}): \quad & \circ_M (P \vee Q) \Leftrightarrow \circ_M P \vee \circ_M Q
\end{aligned}$$

$$(\text{Dist10}): \quad (P \wedge Q)_M \mathbf{until}_N R \Leftrightarrow P_M \mathbf{until}_N R \wedge Q_M \mathbf{until}_N R$$

We prove (Dist1):

Proof :

1. $P \wedge Q \Rightarrow P$ [Ax1]
2. $\Box_M (P \wedge Q) \Rightarrow \Box_M P$ [\Box_M -Mon]
3. $P \wedge Q \Rightarrow Q$ [Ax1]
4. $\Box_M (P \wedge Q) \Rightarrow \Box_M Q$ [\Box_M -Mon]
6. $\Box_M (P \wedge Q) \Rightarrow \Box_M P \wedge \Box_M Q$ [prop]
7. $P \Rightarrow (Q \Rightarrow (P \wedge Q))$ [Ax1]
8. $\Box_M P \Rightarrow \Box_M (Q \Rightarrow (P \wedge Q))$ [\Box_M -Mon]
9. $\Box_M P \Rightarrow \Box_M Q \Rightarrow \Box_M (P \wedge Q)$ [\Box_M -Mon]
10. $\Box_M P \wedge \Box_M Q \Rightarrow \Box_M (P \wedge Q)$ [prop]
12. $\Box_M P \wedge \Box_M Q \Leftrightarrow \Box_M (P \wedge Q)$ [prop]

■

- However, only the following weak laws hold:

- (WDist1): $\Box_M P \vee \Box_M P \Rightarrow \Box_M (P \vee Q)$
(WDist2): $\Diamond_M (P \wedge Q) \Rightarrow \Diamond_M P \wedge \Diamond_M Q$
(WDist3): $P_M \mathbf{until}_N R \vee Q_M \mathbf{until}_N R \Rightarrow (P \vee Q)_M \mathbf{until}_N R$

- The same holds for past operators.

Commutativity Theorems

Examples of commutativity theorems:

- (Com1): $\Box_M \circ_M P \Leftrightarrow \circ_M \Box_M P$
(Com2): $\Diamond_M \tilde{\circ}_M P \Leftrightarrow \tilde{\circ}_M \Diamond_M P$
(Com3): $\blacklozenge_M \Diamond_M P \Leftrightarrow \Diamond_M \blacklozenge_M P$
(Com4): $\blacksquare_M \Box_M P \Leftrightarrow \Box_M \blacksquare_M P$

We prove (Com1):

Proof :

Set $Q = \circ_M \Box_M P$

1. $Q \Rightarrow \Delta_M \circ_M P$ [FAx7]
2. $Q \Rightarrow \circ_M Q$ [FAx7]
3. $Q \Rightarrow \Box_M \circ_M P$ [\Box_M -Int]
4. $\Box_M \circ_M P \Rightarrow (\Delta_M \circ_M P \wedge \circ_M \circ_M \Box_M P)$ [FAx7]
5. $\Box_M \circ_M P \Rightarrow \circ_M (\Delta_M P \wedge \circ_M \Box_M P)$ [Dist3]
7. $\Box_M \circ_M P \Rightarrow \circ_M \Box_M P$ [FAx7]
8. $\Box_M \circ_M P \Leftrightarrow \circ_M \Box_M P$ [prop]

■

Inclusion Rules

- The following rules define a relationship between inclusion of indices and the unary temporal operators.

$$\begin{array}{llll}
(\text{Inc1}): & N \subseteq M \vdash \Delta_M P & \Rightarrow & \Delta_N P & \text{especially: } P \Rightarrow \Delta_N P \\
(\text{Inc2}): & N \subseteq M \vdash \Delta_N P & \Rightarrow & \Delta_M P & \text{especially: } \Delta_N P \Rightarrow P \\
(\text{Inc3}): & N \subseteq M \vdash \Box_M P & \Rightarrow & \Box_N P & \text{especially: } \Box P \Rightarrow \Box_M P \\
(\text{Inc4}): & N \subseteq M \vdash \Diamond_N P & \Rightarrow & \Diamond_M P & \text{especially: } \Diamond_M P \Rightarrow \Diamond P
\end{array}$$

We prove (Inc1):

Proof :

$$\begin{array}{ll}
1. N \subseteq M & [\text{premise}] \\
2. M \cup N = M & [\text{Axiom}] \\
3. \Delta_M P \wedge \Delta_N P \Leftrightarrow \Delta_{M \cup N} P & [\text{Ax4}] \\
4. \Delta_M P \wedge \Delta_N P \Leftrightarrow \Delta_M P & \\
5. \Delta_M P \Rightarrow \Delta_N P & [\text{prop}]
\end{array}$$

■

We prove (Inc3):

Proof :

$$\begin{array}{ll}
1. N \subseteq M & [\text{premise}] \\
2. \Delta_M P \Rightarrow \Delta_N P & [\text{Inc1}] \\
3. \Box(\Delta_M)P \Rightarrow \Box(\Delta_N)P & [\Box\text{-Mon}] \\
4. \Box_M P \Rightarrow \Box_N P & [(\text{IndE3}) \text{ below}]
\end{array}$$

■

The proofs of (Inc2) and (Inc4) ensue from duality theorems.

- However, neither $\tilde{\text{O}}_N P \Rightarrow \tilde{\text{O}}_M P$ nor $\tilde{\text{O}}_M P \Rightarrow \tilde{\text{O}}_N P$ hold for the *next-time* operator whenever N is a subset of M . The same applies to $\tilde{\text{O}}, \bullet, \text{ and } \blacklozenge$.
- Analogous theorems can be proven for **until**, **since**, and past operators.

Union Theorems

- Considering the *always* operator, the union of the indices corresponds to a conjunction at the level of temporal formulas, whereas the union of the indices of the *eventually* operator yields a disjunction.

$$\begin{array}{ll}
(\text{Un1}): & \Box_{M \cup N} P \Leftrightarrow \Box_M P \wedge \Box_N P \\
(\text{Un2}): & \Diamond_{M \cup N} P \Leftrightarrow \Diamond_M P \vee \Diamond_N P
\end{array}$$

Proof of (Un1):

$$\text{Set } Q = \Box_M P \wedge \Box_N P$$

1. $N \subseteq N \cup M$ [Axiom]
2. $M \subseteq N \cup M$ [Axiom]
3. $\Box_{M \cup N} P \Rightarrow \Box_M P$ [Inc3]
4. $\Box_{M \cup N} P \Rightarrow \Box_N P$ [Inc3]
5. $\Box_{M \cup N} P \Rightarrow \Box_M P \wedge \Box_N P$ [prop]
6. $\Box_M P \Rightarrow \circ_M \Box_M P$ [FAx7]
7. $\Box_M P \Rightarrow \circ \Box_M P$ [(IndE3) below]
8. $\Box_N P \Rightarrow \circ \Box_N P$
9. $\Box_M P \wedge \Box_N P \Rightarrow \circ \Box_M P \wedge \circ \Box_N P$ [prop]
10. $\Box_M P \wedge \Box_N P \Rightarrow \circ (\Box_M P \wedge \Box_N P)$ [\circ -Dist]
11. $Q \Rightarrow \circ Q$
12. $Q \Rightarrow \Delta_M P \wedge \Delta_N P$ [Ax, prop]
13. $Q \Rightarrow \Delta_{M \cup N} P$ [Ax4]
14. $Q \Rightarrow \Box_{M \cup N} P$ [\Box -Int]
15. $\Box_{M \cup N} P \Leftrightarrow \Box_M P \wedge \Box_N P$ [prop]

■

Elimination of Indices

As mentioned in Section 3.3, each indexed temporal formula can be transformed into its non-indexed equivalent. Note that the formula $(\Delta_M P)$ can always be replaced by the formula $(\text{active}(M) \Rightarrow P)$.

- For future operators we have:

- (IndE1): $\Box_M P \Leftrightarrow \Box(\Delta_M P)$
 (IndE2): $\Diamond_M P \Leftrightarrow \Diamond(\Delta_M P)$
 (IndE3): $\circ_M \Box_M P \Leftrightarrow \circ \Box_M P$

Proof (IndE1):

1. $\Box_M P \Rightarrow \Delta_M P$ [FAx7]
2. $\Box_M P \Rightarrow \Delta(\Delta_M P)$ [(Inc1)]
3. $\Box_M P \Rightarrow \circ_M \Box_M P$ [FAx7]
4. $\Box_M P \Rightarrow \circ \Box_M P$ [(IndE3) below]
5. $\Box_M P \Rightarrow \Box(\Delta_M P)$ [\Box -Int]
6. $\Box(\Delta_M P) \Rightarrow \Delta_M P$ [FAx7]
7. $\Box(\Delta_M P) \Rightarrow \circ \Box(\Delta_M P)$ [FAx7]
8. $\Box_M P \Rightarrow \Box_M \Rightarrow P$ [\Box -Int]
9. $\Box_M P \Leftrightarrow \Box(\Delta_M P)$ [prop]

■

Proof (IndE3):

1. $\circ_M \square_M P \Leftrightarrow \circ (\Delta_M \square_M P \wedge (\Delta_M \text{false} \Rightarrow \circ_M \square_M P))$ [Ex3]
2. $\circ_M \square_M P \Leftrightarrow \circ (\square_M P \wedge (\Delta_M \text{false} \Rightarrow \circ_M \square_M P))$ [prop]
3. $\circ_M \square_M P \Leftrightarrow \circ (\Delta_M P \wedge \circ_M \square_M P \wedge (\Delta_M \text{false} \Rightarrow \circ_M \square_M P))$ [Rec1]
4. $\circ_M \square_M P \Leftrightarrow \circ (\Delta_M P \wedge \circ_M \square_M P)$ [prop]
5. $\circ_M \square_M P \Leftrightarrow \circ \square_M P$ [Rec1]

■

- Analogous theorems hold for the past operators.

Most of the theorems and rules presented in this section stem from classical temporal logic. The class of properties of temporal operators is very large, and the laws presented above constitute only a small part of this class. Most of these properties also apply to the indexed version of temporal logic. However, there are properties which are not preserved when temporal operators are indexed such as $(\square P \Rightarrow \diamond P)$ and $(\square P \Rightarrow \circ P)$, because $\square_M P$ does not guarantee the occurrence of P in the future.

Note that there is an interesting class of temporal formulas whose meaning is not changed by indexing, such as:

$$\diamond_S [S \mathbf{xmt} m] \Leftrightarrow \diamond [S \mathbf{xmt} m]$$

$$\blacklozenge_S [S \mathbf{xmt} m] \Leftrightarrow \blacklozenge [S \mathbf{xmt} m]$$

$$\blacklozenge_S [S \mathbf{xmt} m] \Leftrightarrow \blacklozenge [S \mathbf{xmt} m]$$

Chapter 4

Extending the Temporal Logic

In this chapter, we investigate the usefulness of the temporal logic defined in the preceding chapter for the specification of communication protocols. We will show that a large class of properties cannot be expressed within this logic. The source of the inadequacy is the inability of temporal logic to uniquely identify messages on a stream. Consequently, one cannot couple sent messages with received ones. Thus, an extension of temporal logic is required in order to overcome this lack of expressiveness. In our approach, we introduce a mechanism for uniquely identifying messages: each message is assigned a color. We will see that a hierarchy of equivalence classes of colors is needed in order to express desired properties of communication protocols.

This chapter is organized as follows. The first section presents some inexpressiveness results. In Section 4.2, we introduce colors and show which role they play in the specification of protocols. Then, in Sections 4.3, 4.4, and 4.5 the logic is extended step by step. In Section 4.8, we illustrate the application of the extended model by the specification of some services provided by the Internet Protocol.

4.1 The Inexpressiveness of Temporal Logic

The specification of communication protocols is based on the description of the behavior of the agents which comprise the distributed system. As mentioned in Section 2.1, an agent is viewed as a black box which receives and sends messages on its input and output channels, without considering the internal structure; such agents are often called message-passing systems.

Many studies have established that there are some inexpressiveness problems when so-called message-passing systems are specified using temporal logic. Sistla et al. ([SCFG82] and [SCFM84]) proved that is not possible to describe unbounded buffers (these are particular message-passing systems) with linear temporal logic. In a study of message-passing systems by Koymans ([Koy92]), it has been found that the class of transmission mediums that satisfy the properties

- (a) that the medium does not create messages, neither by generating “new” messages, nor by duplicating existing ones, and

(b) that any received message should be eventually sent,

cannot be specified by a finite set of temporal formulas.

Most analyses, such as [Koy92] and [Lam83b], show that the cause of this inexpressiveness is that it is impossible to couple each message sent by an agent to a unique message received by that agent. This is because messages cannot be uniquely identified on infinite streams. To make this clear, consider a medium A that should satisfy the property (a). It is desirable to formulate this property using the following simple formula which claims that A only sends what it has received.

$$\forall m : ([A \text{ snd } m] \Rightarrow \blacklozenge [A \text{ rcv } m])$$

Unfortunately, this formula does not correspond to the property (a), because it allows the agent A to send many copies of a received message.

Hence, it becomes evident that mechanisms for the unique identification of messages on a stream are needed in order to overcome this deficiency. In this context, we can distinguish two approaches. In the first approach, messages are made implicitly unique through the addition of special auxiliary variables and operations on these variables; Hailpern [Hai82] introduced *history variables* to this end. In this approach, one reasons about agents using input and output histories instead of detached messages. In the second approach, unique identification of messages is explicitly assumed a priori by means of *conceptual time-stamps*. That is, a message is assumed to be composed of a data element, which represents its content, and a time-stamp, which makes it unique on a stream.

In some works, such as [Pnu92], natural numbers are used for time-stamping; each message on a stream is indexed by a natural number, which serves as the unique identifier for this message. For some applications, this may be sufficient for describing the properties of the desired systems. However, for many applications, taking natural numbers as unique identifiers may lead to a number of difficulties, especially when one considers the composition of subsystems. For instance, one could never link the output channel of a transmission medium which may duplicate messages to the input channel of an agent which expects to receive distinct messages.

Our approach is based on the concept of time-stamping, the difference being that we use “*color*” to uniquely identify messages. So our messages are assumed to be colored data elements. Further, we characterize the set of colors in a way that allows us, on the one hand, to describe potential failures of agents, such as the possibility of messages being lost, duplicated, or permuted, and, on the other hand, to compose subsystems. We will show that we need a hierarchy of equivalence classes of colors.

4.2 Introducing Colors

As mentioned above, the unique identification of messages is explicitly incorporated in the specification of an agent. That is, in a specification we assume that received and sent messages are distinct. Note that this assumption is not restrictive, because one can impose the distinction of messages by means of abstract time-stamps. In doing so, we ensure that messages are distinct even when they have the same data content. This is why we introduce *colors* as conceptual time-stamps. From now on a message is viewed as a colored data element.

Further, we assume the existence of enough colors to guarantee that each data element can be assigned an infinite number of colors.

It should be stressed that colors are introduced only at the conceptual level; we will never refer explicitly to colors in a specification. This entails that they are entirely invisible for agents, i.e. the functional behavior of an agent does not depend on how the incoming messages are colored. Indeed, the introduction of colors at the syntactical level depends on whether the protocol modules (in the implementation) need to test time-stamps or not.

In order to reason about “colored” messages, we introduce a relation \approx between messages which is defined as: $m \approx m'$ iff m and m' have the same color. The fact that a channel C transmits only distinctly colored messages can be expressed by the temporal formula:

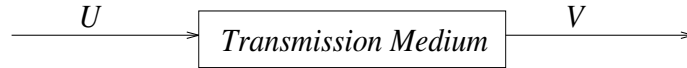
$$\text{(Unq): } [C \mathbf{xmt} m] \wedge \diamond [C \mathbf{xmt} m'] \Rightarrow m \not\approx m'$$

This formula says that if C transmits a message m and later another message m' , then m and m' must be distinctly colored. Obviously, the fact that m and m' have distinct colors ensures that they are distinct, even if they have the same data content.

In order to formulate protocol properties by a (finite) set of temporal formulas, the formula (Unq) must be satisfied by every channel included in the system we intend to specify. As a result, we take it as a general axiom in our specifications. The necessity of this assumption can be shown in the following example.

Example 4.2.1 [Reliable Medium]

A transmission medium is modeled by an agent that receives and sends messages on its input and output channels, respectively.



A transmission medium is said to be reliable if it satisfies the following properties:

- Messages cannot be created, neither “new” messages nor duplicates of received messages.
- No message is lost.

The first property is a safety property and can be specified by

$$[V \mathbf{xmt} m] \Rightarrow \blacklozenge [U \mathbf{xmt} m]$$

This formula claims that the medium sends only what it has received. However, without the property (Unq) on V , this formula does not exclude the possibility of duplicating messages.

The second property of the transmission medium is a liveness property, claiming that any received message will eventually be sent. This can be expressed by

$$[U \mathbf{xmt} m] \Rightarrow \diamond [V \mathbf{xmt} m]$$

Similarly, if we do not consider the property that U transmits only distinct colored messages, then the medium may send one message for many received identical ones. Obviously, this will violate the property of losing no messages.

Another property that strongly depends on the assumption (Unq) is the property of preserving the order of messages. That is, messages should be sent in the same order as they were received. We would suggest formulating this property in the following way, which also seems to be the most natural way:

$$(\blacklozenge[V \mathbf{xmt} m] \wedge [V \mathbf{xmt} n]) \Rightarrow \blacklozenge(\blacklozenge[U \mathbf{xmt} m] \wedge [U \mathbf{xmt} n])$$

However, without the assumption of unique identification, the following sequence of events, where the order of messages is not preserved, would be an acceptable computation:

$$(U, v_1)(U, v_2)(U, v_1)(V, v_2)(V, v_1) \dots$$

As shown in this example, the specification of the transmission medium can ensure its proper functioning only if the uniqueness of messages on the streams U and V is assumed.

4.3 Specification of Failures

The properties of the transmission medium presented in the last section are desirable, but they are not always feasible. In the world of distributed systems, we also have to deal with unreliable systems due to physical failures. Generally, a transmission medium may create, duplicate, lose, or permute messages. Hence, the specification language should allow the description of unreliable systems as well.

In Section 4.2, it has been shown how the assumption (Unq) is necessary in order to express agent properties. However, if we want to specify unreliable systems, we need a stronger assumption concerning the color of messages. In the following example, we discuss what should be added in order to be able to formulate properties of unreliable systems.

Example 4.3.1 [Unreliable Medium]

We will specify a transmission medium that may duplicate messages, but neither loses messages nor creates “new” ones. Duplication of messages means that the transmission medium may send a received message twice. This medium is modeled by the following agent *Duplicate*.



If we assume that all messages received on S are distinctly colored, it is possible to formulate the property of duplicating messages by allowing the channel R to transmit messages of the same color, provided that each color occurs at most twice. This can be formulated by the following temporal formula:

$$[R \mathbf{xmt} m] \wedge \blacklozenge([R \mathbf{xmt} m'] \wedge \blacklozenge[R \mathbf{xmt} m'']) \Rightarrow (m \not\approx m' \vee m \not\approx m'')$$

Obviously, this formula is not consistent with the assumption (Unq). Consequently, there may be some complications when we combine the agent *Duplicate* with the transmission medium specified in Section 4.2, which expects to receive only distinctly colored messages. The problem to be solved is how to formulate the property of duplicating messages without violating the uniqueness property (Unq) on the channels S and R .

The main idea is to use equivalence classes of colors. One can imagine that each color builds a class. For example, the class *blue* includes the colors *dark blue*, *light blue*, and so on. Further, one can talk, for example, about the class *dark blue* as including all colors that can be derived from dark blue. In this way, we obtain a hierarchy of equivalence classes of colors. We will see later that such a hierarchy is needed when we describe protocols.

For the moment, we assume only one equivalence relation on colors (apart from equality), because this suffices to formulate the properties of the medium we intend to specify. Further, we assume that there is an infinite number of classes such that a channel may transmit an infinite number of nonequivalently colored messages. This leads to the following equivalence relations on the set of messages:

$$\begin{aligned} m \sim m' &\stackrel{\text{def}}{=} m \text{ and } m' \text{ have equivalent colors} \\ m \cong m' &\stackrel{\text{def}}{=} m \text{ and } m' \text{ have equivalent colors and the same data element} \end{aligned}$$

Now, in order to formulate the properties of *Duplicate*, we first describe how messages on the streams S and R are colored. We assume that the messages on the stream S are pairwise non-equivalently colored. This is formulated by

$$[S \text{ xmt } m] \wedge \diamond [S \text{ xmt } m'] \Rightarrow m \not\sim m'$$

Obviously, this entails that messages on S are distinctly colored.

In order to allow duplication of messages, we assume that R may transmit messages with equivalent (but different by Unq) colors. However, at most two colors from each class are allowed to be transmitted on R . This is formulated by:

$$[R \text{ xmt } m] \wedge \diamond ([R \text{ xmt } m'] \wedge \diamond [R \text{ xmt } m'']) \Rightarrow (m \not\sim m' \vee m \not\sim m'')$$

Following these two assumptions, the transmission medium *Duplicate* is specified by

$$\begin{aligned} [R \text{ xmt } m] &\Rightarrow \blacklozenge [S \text{ xmt } \tilde{m}] \\ [S \text{ xmt } m] &\Rightarrow \blacklozenge [R \text{ xmt } \tilde{m}] \end{aligned}$$

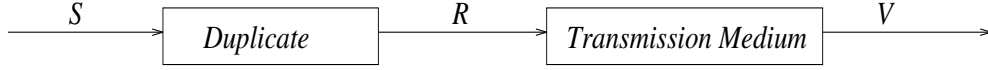
where \tilde{m} denotes an arbitrary message that is \cong -equivalent to m .

The second formula claims that R transmits one equivalent for every received message. Because R may transmit messages with equivalent colors, *Duplicate* may send two \cong -equivalent messages for each message received. These are considered as two copies. If we now compare two streams of messages received and sent on S and R , respectively, without considering colors, we will observe that the stream R may contain duplicates of data elements which occurred on the stream S .

4.4 Composition of Agents

Distributed systems are often composed of more than one component. Due to the complexity of these systems, it is useful to have a specification language that supports modular decomposition. That is, a system component can be independently specified, and the specification of the whole system can be derived from the specifications of its components. The temporal logic presented in Section 3.3 supports such modular decomposition, as was proved in [Jma95b].

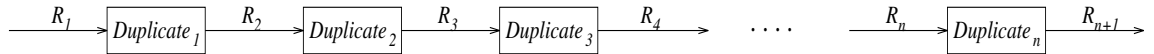
In our approach, the composition of two networks is achieved by connecting some input channels of one network to some output channels of the other in a one-to-one manner. This is done by identifying the names of the linked input and output channels in the resulting network. For example, in composing the two agents *Duplicate* and *Transmission Medium*, we identify the two channels R and U to a channel R .



Thanks to the compositionality of our temporal logic, the specification of this network is obtained by the conjunction of the specifications of both agents, where in the specification of the transmission medium we substitute each occurrence of U by R . This leads to the following specification:

$$\begin{aligned}
 [S \mathbf{xmt} m] \wedge \diamond [S \mathbf{xmt} m'] &\Rightarrow m \not\sim m' \\
 [R \mathbf{xmt} m] \wedge \diamond ([R \mathbf{xmt} m'] \wedge \diamond [R \mathbf{xmt} m'']) &\Rightarrow (m \not\sim m' \vee m \not\sim m'') \\
 [R \mathbf{xmt} m] \wedge \diamond [R \mathbf{xmt} m'] &\Rightarrow m \not\sim m' \\
 [V \mathbf{xmt} m] \wedge \diamond [V \mathbf{xmt} m'] &\Rightarrow m \not\sim m' \\
 [R \mathbf{xmt} m] &\Rightarrow \blacklozenge [S \mathbf{xmt} \tilde{m}] \\
 [S \mathbf{xmt} m] &\Rightarrow \blacklozenge [R \mathbf{xmt} \tilde{m}] \\
 [V \mathbf{xmt} m] &\Rightarrow \blacklozenge [R \mathbf{xmt} m] \\
 [R \mathbf{xmt} m] &\Rightarrow \blacklozenge [V \mathbf{xmt} m]
 \end{aligned}$$

As shown in this example, the composition of agents entails a partitioning of the colors set into equivalence classes. The set of colors is partitioned in such a way that we obtain an infinite number of equivalence classes, where each class itself is infinite. However, there are some situations in which we have to apply such partitioning several times recursively. For example, let us consider the following composition.



In this network, each agent $Duplicate_i$ has the same behavior properties as the agent $Duplicate$ already specified in Section 4.3. We take as a specification for $Duplicate_n$ the specification of $Duplicate$. This entails that $Duplicate_n$ receives on R_n only messages that are colored with nonequivalent colors. In specifying $Duplicate_{n-1}$, this restriction should be respected; $Duplicate_{n-1}$ can send only nonequivalently colored messages on R_n . On the other hand, we have to ensure the proper functioning of $Duplicate_{n-1}$ by allowing it to send messages twice. Obviously, we have the same situation as when we composed $Duplicate$ with the transmission

medium. So we partition the set of colors again, adding a second equivalence relation in a way that each equivalence class w.r.t. to the second relation includes an infinite number of equivalence classes w.r.t. the first. If we translate these two relations to the set of messages, we obtain in addition to \sim and \cong the following relations:

$$\begin{aligned} m \sim_2 m' &\stackrel{\text{def}}{=} m \text{ and } m' \text{ have equivalent colors w.r.t. the second relation} \\ m \cong_2 m' &\stackrel{\text{def}}{=} m \text{ and } m' \text{ are equivalent w.r.t. } \sim_2 \text{ and have the same data element} \end{aligned}$$

The specification of $Duplicate_{n-1}$ is obtained from the specification of $Duplicate$ (see Section 4.3) by substituting each occurrence of S and R with R_{n-1} and R_n , respectively, and replacing the relations \sim and \cong by \sim_2 and \cong_2 . This leads to the following specification:

$$\begin{aligned} [R_{n-1} \mathbf{xmt} m] \wedge \diamond [R_{n-1} \mathbf{xmt} m'] &\Rightarrow m \not\sim_2 m' \\ [R_n \mathbf{xmt} m] \wedge \diamond ([R_n \mathbf{xmt} m'] \wedge \diamond [R_n \mathbf{xmt} m'']) &\Rightarrow (m \not\sim_2 m' \vee m \not\sim_2 m'') \\ [R_n \mathbf{xmt} m] &\Rightarrow \blacklozenge [R_{n-1} \mathbf{xmt} \hat{m}] \\ [R_{n-1} \mathbf{xmt} m] &\Rightarrow \diamond [R_n \mathbf{xmt} \hat{m}] \end{aligned}$$

where \hat{m} denotes any message that is equivalent to m w.r.t. \cong_2 .

Now we have no problems if we assume that the messages on R_n have to be nonequivalent w.r.t. \sim . Accordingly, the agents $Duplicate_n$ and $Duplicate_{n-1}$ can be combined without difficulty.

In order to ensure the proper functioning of each agent in the network, we have to apply the process of partitioning to the set of colors successively until the whole system is specified. Hence, we introduce n equivalence relations on the set of colors with the property that these relations form an inclusion chain, i.e. the i -th relation is included in the $(i+1)$ -th one, where the smallest relation is the equality on colors. Moreover, we assume that each equivalence class w.r.t. the $(i+1)$ -th relation contains infinitely many classes w.r.t. the i -th equivalence relation. If we translate these relations to the set of messages, as done in Section 4.3, we obtain a hierarchy of equivalence classes on the set of messages. Using this hierarchy, we are able to specify any combination of message-passing systems.

4.5 Extending Operations w.r.t. Colors

Agents are active system components which, in addition to sending and receiving data, can perform internal operations on data. In the context of communication protocols, an operation may be the composition of messages from individual pieces, splitting packets into frames, selecting fragments from messages, etc. Actually, these operations are defined on data and not on colored messages. Since in our extended model we deal with agents that send and receive colored messages, we have to extend these operations to the set of colored messages. In our approach, it is not necessary to give a complete definition of the extended operations, rather it suffices to give the properties that they have to fulfill. Such a characterization enables us to specify protocols in a purely temporal formalism.

In the following example, we discuss the properties that should be satisfied by the extended operations. For this purpose, we specify an agent *Split* that receives packets, splits each of

them into two frames, and then sends the frames stemming from the same packet consecutively.

Example 4.5.1 [Split]

The agent *Split* is a transmission medium that performs a split operation (say sp) on received packets before it sends them. This agent has an input channel S and an output channel R .



The unique requirement we place on the behavior of *Split* is that the frames of a received packet must eventually be sent consecutively.

Following the assumption (Unq) introduced in Section 4.2, the packets on the stream S must be colored with pairwise different colors. The problem that now arises is how to couple sent frames to the packets from which they stem. Obviously, this will not be possible if we assume that frames on the stream R are distinctly colored. On the other hand, we cannot assume that frames from the same packet have the same color. A solution might be to use an equivalence relation on colors, as in the specification presented in Section 4.4. Accordingly, we assume that S transmits only nonequivalently colored packets, whereas R may transmit equivalently (but distinctly) colored frames. These assumptions are formulated by:

$$[S \mathbf{xmt} p] \wedge \diamond [S \mathbf{xmt} p'] \Rightarrow p \not\sim p'$$

$$[R \mathbf{xmt} fr] \wedge \diamond ([R \mathbf{xmt} fr'] \wedge \diamond [R \mathbf{xmt} fr'']) \Rightarrow (fr \not\sim fr' \vee fr \not\sim fr'')$$

Under these assumptions, and if for any packet p $sp(p) = (sp_1(p), sp_2(p))$, we formulate the requirement claiming that the frames of a received packet must eventually be sent consecutively by:

$$[S \mathbf{xmt} p] \Rightarrow \diamond ([R \mathbf{xmt} sp_1(p)] \wedge \tilde{O}_R [R \mathbf{xmt} sp_2(p)])$$

The above formula does not ensure unique correspondence between a packet and its frames, because it says nothing about the colors of $sp_1(p)$ and $sp_2(p)$. In order to match a packet to its frames, we have to assume that frames have equivalent colors to the color of the packet they stem from.

$$(sp_1(p) \sim p) \wedge (sp_2(p) \sim p)$$

This formula states that an (extended) operation should preserve the equivalence class of its arguments. In other words, the relation \sim should be a congruence relation.

Moreover, in order to avoid inconsistency in our specifications, we have to assume that the frames stemming from the same packet have different colors. This leads to the addition of the following property.

$$sp_1(p) \not\sim sp_2(p)$$

The most important requirement on the extension is that the image (by the original operation) of the content of a colored message is the same as the content of its image by the extended operation. These three properties are generalized in the next section.

4.6 The Extended Model

Because we are aiming for specifications that are purely temporal, and because colors should be hidden at the syntactical level, the three properties elaborated above should be satisfied by the operations in the extended semantical model. In the following, we generalize them for an arbitrary extended operation. First, we assume that for the specification of a system we have a set of n equivalence relations on the set of colored messages $\sim_1, \sim_2, \dots, \sim_n$, as defined in Section 4.4, which form the inclusion chain $\sim_1 \subset \sim_2 \subset \dots \subset \sim_n$.

Let g be an operation defined on the sets of data with the arity (l, k) . For each of the equivalence relations \sim_i , we assign to the function g a function \widehat{g}^i defined on the corresponding sets of colored data.

Let $a_1, \dots, a_l, b_1, \dots, b_k$ be colored data with $\widehat{g}^i(a_1, \dots, a_l) = (b_1, \dots, b_k)$. Generally, the arguments a_1, \dots, a_l are messages received on the input channels (there may be only one channel) of an agent, and the image components b_1, \dots, b_k are messages to be sent on the output channels of that agent. The extension \widehat{g}^i should have the following properties:

- In order to match sent messages to received messages, we expect that the components of an image are equivalent w.r.t. \sim_i with at least one argument, for example the first.

$$\bigwedge_{j=1}^k a_1 \sim_i b_j$$

Note that this property entails that \sim_i is a congruence relation on the set of messages. Hence we have

$$(x_1, \dots, x_l) \sim_i (y_1, \dots, y_l) \Rightarrow \widehat{g}^i((x_1, \dots, x_l)) \sim_i \widehat{g}^i((y_1, \dots, y_l))$$

for all colored messages, x_1, \dots, x_l and y_1, \dots, y_l

- The components of an image must be pairwise nonequivalent w.r.t. \sim_{i-1} . Hence, we avoid inconsistency in our specifications; usually if we allow equivalence of messages w.r.t. the relation \sim_i , this is because they must be pairwise nonequivalent w.r.t. \sim_{i-1} .

$$\forall m, j, 1 \leq m, j \leq k : (b_j \sim_{i-1} b_m) \Rightarrow j = m$$

- Restricted on the corresponding data sets \widehat{g}^i is equal to g .

$$g(\text{Content}(a_1), \dots, \text{Content}(a_l)) = (\text{Content}(b_1), \dots, \text{Content}(b_k))$$

where *Content* is the function that extracts the data part from a colored message.

Assuming the properties introduced above, we can specify our systems using the extension by colors without any complications, preserving the consistency of specifications, and applying a purely temporal formalism.

Let $\mathcal{M} = (\text{Alg}, (\mathcal{E}, \leq))$ be a semantic model. An extended model \mathcal{M}_{ext} obtained from \mathcal{M} consists of a colored algebra and a sequence of events, where each event is viewed as a pair

containing a channel name and a colored message. The colored algebra arises from the algebra Alg when the data elements are colored and the operations are extended to the corresponding sets of colored data.

Remark 4.6.1

Coloring data elements with an infinite number of colors would force us to deal with infinite sets of data, even if the original sets of data are finite. This entails that quantification over finite sets of data will be replaced by quantification over infinite sets of data, and that first-order temporal logic must be used instead of propositional temporal logic. However, as long as colors do not influence the functional behavior of agents, one can use propositional temporal formulas to prove properties stated over infinite sets of colored messages, provided that the original sets of data are finite. This can be justified by the results proved by Wolper in [Wol86]. He showed that under the assumption that the functional behavior of agents does not depend on data, a large class of properties stated over an infinite number of data-values are equivalent to properties stated over a finite set of data-values.

4.7 The Predicate Calculus

On the basis of our extended semantics we now present the predicate part of the proof system given in Section 3.4. Assuming we have n congruence relations \sim_1, \dots, \sim_n on the set of colored messages, we then have the following rule and axioms:

- *Quantifier Rule*

$$P \Rightarrow Q \vdash P \Rightarrow \forall x.Q \quad \text{if there are no free occurrences of } x \text{ in } P$$

- *Quantifier Axioms*

$$\mathbf{QAx1:} \quad \neg \exists x.P \Leftrightarrow \forall x.\neg P$$

$$\mathbf{QAx2:} \quad \forall x.P(x) \Rightarrow P(t) \quad \text{if } t \text{ is substitutable for } x \text{ in } P$$

$$\mathbf{QAx3:} \quad \forall x.\circ P \Rightarrow \circ \forall x.P$$

$$\mathbf{QAx4:} \quad \forall x.\bullet P \Rightarrow \bullet \forall x.P$$

We say t is substitutable for x in P if its substitution for all free occurrences of x in P does not introduce new bound occurrences of variables.

- *Equivalence Axioms*

$$\mathbf{EAx1:} \quad t \sim t \quad \text{for any term } t \text{ and for all } \sim \in \{\sim_1, \dots, \sim_n\}$$

$$\mathbf{EAx2:} \quad t_1 \sim t_2 \wedge t_2 \sim t_3 \Rightarrow t_1 \sim t_3 \quad \text{for all } \sim \in \{\sim_1, \dots, \sim_n\}$$

$$\mathbf{EAx3:} \quad t_1 \sim t_2 \Rightarrow t_2 \sim t_1 \quad \text{for all } \sim \in \{\sim_1, \dots, \sim_n\}$$

$$\mathbf{EAx4:} \quad t_1 \sim_i t_2 \Rightarrow t_1 \sim_{i+1} t_2$$

The axioms EAx1–EAx3 state that each of the \sim_i is an equivalence relation. The axiom EAx4 states that the equivalence relations form an inclusion chain.

• *Congruence Axioms*

Let \widehat{f}^i be an extension of an operation f with the arity (m, l) , and $\widehat{f}_j^i(t_1, \dots, t_m)$ be the j -th component of $\widehat{f}^i(t_1, \dots, t_m)$; we then have the following axioms:

$$\mathbf{CAx1:} \quad \bigwedge_{k=1}^l (\widehat{f}_k^i(t_1, \dots, t_m) \sim_i t_1)$$

This axiom states that \widehat{f}^i preserves the equivalence class of its arguments.

$$\mathbf{CAx2:} \quad \widehat{f}_k^i(t_1, \dots, t_m) \sim_{i-1} \widehat{f}_j^i(t_1, \dots, t_m) \Rightarrow j = k$$

The components of an image of \widehat{f}^i are not equivalent w.r.t. \sim_{i-1} .

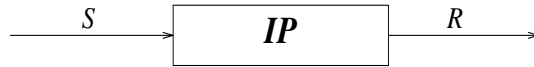
$$\mathbf{CAx3:} \quad f(\text{Content}(t)) = \text{Content}(\widehat{f}^i(t))$$

It can be shown that this proof system is sound.

4.8 A Specification Example: The Internet Protocol

In this section we illustrate the application of our extended temporal logic for communication protocols by specifying some properties of the service provided by the Internet Protocol (in short IP). We will show that the model, based on the hierarchy of colors, is suitable for the description of different system properties, such as loss of messages, duplication of messages, or preservation of ordering.

The Internet Protocol is viewed as an unreliable transmission medium with a unique input channel S and output channel R .



The Internet Protocol has the following properties:

- Messages may be lost, duplicated, or permuted, but they cannot be created.
- The IP must not filter out some specific messages forever.

The specification of the IP behavior requires the use of an equivalence relation on the set of colors, because the property of duplicating messages (see Section 4.3) as well as the possibility of receiving a message many times cannot be specified if the channels S and R are supposed to transmit only differently colored messages. Thus, we assume the existence of an equivalence relation on the set of colors. We denote by \tilde{m} any message that has an equivalent color to the

color of m and the same content. Following the assumption (Unq), the requirement claiming that IP can only send what it has received (no creation) is formulated by:

$$[R \mathbf{xmt} m] \Rightarrow \blacklozenge [S \mathbf{xmt} \tilde{m}]$$

Note that we have written $[S \mathbf{xmt} \tilde{m}]$ and not $[S \mathbf{xmt} m]$, otherwise IP could send at most one message for each received message, and hence the property that IP may duplicate messages would no longer be satisfied.

The next property we expect from IP is that it cannot permanently filter a specific message; this means that if a message has been “infinitely” often received, it must eventually be sent. A message is considered to have been infinitely often received if an infinite number of messages from its equivalence class have been received. This property is formulated by:

$$\square \blacklozenge [S \mathbf{xmt} \tilde{m}] \Rightarrow \blacklozenge [R \mathbf{xmt} m]$$

These two formulas ensure that messages can neither be created nor permanently filtered, but they may be lost, duplicated, or permuted. If we want to prevent messages being permuted, we should add the following formula to the specification of IP:

$$(\blacklozenge [R \mathbf{xmt} m] \wedge [R \mathbf{xmt} n]) \Rightarrow \blacklozenge (\blacklozenge [S \mathbf{xmt} m] \wedge [S \mathbf{xmt} n])$$

Note that this specification cannot be formulated without adding an equivalence relation on colors as done before. We must, on the one hand, preserve the property of transmitting only distinct messages on S and R to express the property of preserving the order of received messages. On the other hand, we have to allow the transmission of identical messages on S and R .

Chapter 5

Development Methodology

The task of software development is, given a requirements specification, to construct an executable program which is a correct realization of the given specification, i.e. it meets all requirements of the specification. In general, the development process of a software system consists of several phases, including e.g. requirements engineering, development of design specifications, and program implementations. Actually, there is no universal strategy that would guarantee a correct implementation of a given specification. In general, one provides methodologies and perhaps heuristics which are oriented towards a specific problem area. This chapter presents a methodology for constructing design specifications from requirements specifications of communication protocols. It should be noted that requirements and design specifications are represented in the same language.

The development of design specifications proceeds systematically in a step-by-step fashion. In each development step the original requirements specification is enriched with more details and design decisions. One of the overall aims of using formal languages is to provide rigorous development rules that guarantee the correctness of the final product w.r.t. the original requirements specification. In this context, it is very important to have a formalism that supports hierarchical development in a modular style. This reduces the complexity of verifying development steps, because one can decompose the system into several parts.

We will show that the temporal logic presented here is suitable for hierarchical development of protocols in a modular style. It supports composition and refinement of specifications. We give a rule that justifies the composition of specifications, proving that the logic is modular, and a rule for verifying refinements. Moreover, we introduce a rule that demonstrates the compatibility between compositions and refinements. This enables us to develop system components independently.

This chapter is organized as follows. The first section defines the notion of processes and operations on them. Further, we define a satisfaction relation between processes and specifications. In Section 5.2, we deal with the composition of specifications and present a composition rule. Section 5.3 handles refinements of specifications. Section 5.4 describes how we proceed when we develop a specific protocol.

5.1 Processes and the Satisfaction Relation

In general, when dealing with the concepts of compositionality, modularity, and refinements in the development of systems, three levels of description are considered: a specification language for describing properties of systems (e.g. temporal logic), a language for representing processes (such as CCS or CSP), and a semantic model which is often a computational model, e.g. transition systems or event structures. Often, the semantic model defines the semantics of processes and serves for the interpretation of specifications as well. Based on these domains, a satisfaction relation between processes and specifications is defined in the following way. A process \mathcal{P} satisfies a specification \mathcal{S} if every computation of \mathcal{P} is a model for \mathcal{S} . Further, the concepts of compositionality and modularity can be defined as follows. A formalism is said to be *compositional* if to each decomposition of the system (process) there is a corresponding decomposition of the specification. A formalism is *modular* if the specification of a system can be derived from the specifications of its components. Compositionality and modularity are necessary for top-down and bottom-up development styles [Zwi89].

According to the definitions given above, compositionality and modularity mean that operations that are available at the level of processes correspond to building operations on the set of specifications. In the context of concurrent and distributed systems, the most important operation on processes is parallel composition. In our approach, we do not specify any concrete implementation language for agents, i.e. we do not give any syntax for processes. However, we assume that the language of the processes in question includes the concepts of data types and communications, and allows the parallel composition of processes. A language like LOTOS or Occam can be taken as an implementation language for agents, because on the one hand, it supports the concepts listed above and, on the other hand, event-based semantics for (LOTOS or Occam) processes can be given in the semantic model presented in Section 2.4. The processes discussed below are considered to be programs written in such a language.

Given a process \mathcal{P} , the semantics of \mathcal{P} , denoted as $\llbracket \mathcal{P} \rrbracket$, consists of a pair (Alg, \mathcal{V}) , where Alg is an algebra defining the data types of \mathcal{P} , and \mathcal{V} is the set of all infinite sequences of events

$$\sigma = e_1 e_2 e_3 \dots$$

corresponding to the executions that can be performed by \mathcal{P} . An event is a pair (C, v) , where C is a channel name and v is a data element. An event (C, v) in a sequence σ corresponds to the occurrence of a transmission of the value v on the channel C . To each process \mathcal{P} we associate a set $\alpha\mathcal{P}$ of events in which \mathcal{P} can engage, and call it the alphabet of \mathcal{P} .

Definition 5.1.1 [Parallel Composition]

Let \mathcal{P}_1 and \mathcal{P}_2 be processes, with semantics (Alg_1, \mathcal{V}_1) and (Alg_2, \mathcal{V}_2) , respectively. The parallel composition of \mathcal{P}_1 and \mathcal{P}_2 , denoted as $\mathcal{P}_1 \parallel \mathcal{P}_2$, yields a process with the semantics (Alg, \mathcal{V}) , which is defined as follows:

$$\begin{aligned} Alg & \stackrel{\text{def}}{=} Alg_1 \cup Alg_2 \\ \mathcal{V} & \stackrel{\text{def}}{=} \{ \sigma \in \alpha(\mathcal{P}_1 \parallel \mathcal{P}_2)^\omega \mid \sigma \downarrow \alpha\mathcal{P}_1 \in \mathcal{V}_1 \wedge \sigma \downarrow \alpha\mathcal{P}_2 \in \mathcal{V}_2 \} \\ \alpha(\mathcal{P}_1 \parallel \mathcal{P}_2) & \stackrel{\text{def}}{=} \alpha\mathcal{P}_1 \cup \alpha\mathcal{P}_2 \end{aligned}$$

Usually, the algebra Alg is obtained by the amalgamation (or *sum*, see [EM85]) of Alg_1 and Alg_2 w.r.t. the algebra representing the data types used in both processes, \mathcal{P}_1 and \mathcal{P}_2 . It is required that the data types used in both \mathcal{P}_1 and \mathcal{P}_2 have same representation in Alg_1 and Alg_2 . The amalgamation of Alg_1 and Alg_2 corresponds to their union, that is, the union of their families of data sets and of their families of operations.

An execution of $\mathcal{P}_1 \parallel \mathcal{P}_2$ is an interleaving of an execution of \mathcal{P}_1 and an execution of \mathcal{P}_2 , in which any action in the intersection ($\alpha\mathcal{P}_1 \cap \alpha\mathcal{P}_2$) is a common action of both processes. This definition corresponds to the trace semantics definition of the CSP parallel construct in [Hoa85].

Definition 5.1.2 [Satisfaction Relation]

We say that a process \mathcal{P} , with $\llbracket \mathcal{P} \rrbracket = (Alg, \mathcal{V})$, implements a specification $\mathcal{S} = \langle Dspec, Net, Pr \rangle$, denoted as $\mathcal{P} \mathbf{sat} \mathcal{S}$, if Alg is a *Dspec*-Algebra and for all $\sigma \in \mathcal{V}$ we have $(Alg, \sigma) \models Pr$. That is, the temporal formulas in Pr are valid in every execution of \mathcal{P} .

Further, we define the reduct of a process on a signature. This is needed for defining refinements.

Definition 5.1.3

Let \mathcal{P} be a process which implements the specification $\mathcal{S} = \langle (\Sigma, E), Net, Pr \rangle$, and let Σ' be a signature with $\Sigma' \subseteq \Sigma$. The reduct of the process \mathcal{P} on Σ' , written as $\mathcal{P}|_{\Sigma'}$, is the same process as \mathcal{P} , except that only communications of sorts in Σ' are observable. Formally,

$$\llbracket \mathcal{P}|_{\Sigma'} \rrbracket \stackrel{\text{def}}{=} (Alg|_{\Sigma'}, \{\sigma|_{\Sigma'} \mid \sigma \in \mathcal{V}\})$$

where $Alg|_{\Sigma'}$ is exactly the same algebra as Alg , except that it only includes carrier sets and functions for sorts and operation symbols in Σ' . For a sequence σ of events, $\sigma|_{\Sigma'}$ is obtained from σ by deleting all events whose values are not included in $Alg|_{\Sigma'}$.

5.2 Composition

In this section, we show that the indexed temporal logic defined in Section 3.3 is modular. Moreover, we give a justification rule for composing specifications. In our approach, composition is achieved by union, so the parallel composition of two processes \mathcal{P}_1 and \mathcal{P}_2 with specifications $\mathcal{S}_1 = \langle Dspec_1, Net_1, Pr_1 \rangle$ and $\mathcal{S}_2 = \langle Dspec_2, Net_2, Pr_2 \rangle$, respectively, yields a process that satisfies the specification $\mathcal{S}_1 \oplus \mathcal{S}_2$, which we define as follows:

$$\mathcal{S}_1 \oplus \mathcal{S}_2 \stackrel{\text{def}}{=} \langle Dspec_1 \cup Dspec_2, Net_1 \cup Net_2, Pr_1 \cup Pr_2 \rangle$$

The composition of data-specification parts is achieved by the union of both specifications, which corresponds to the union of the sorts, the operation symbols, and the equations. We

assume that the sorts and operation symbols to be identified in the composite specification have the same names in both specifications. Accordingly, the union of the specifications corresponds to a pushout construction w.r.t. their intersection (see [EM85]).

The composition of two networks is achieved by connecting some (external) input ports of one network to some (external) output ports of the other in a one-to-one manner. This is done by identifying the names of the linked input and output ports in the resulting network. For example, in composing the two agents in Figure 5.1 to generate the network in Figure 5.2, we identify the channels R and U to a channel R .



Figure 5.1: Two Agents

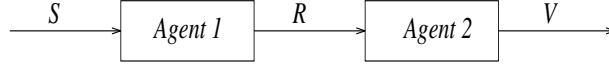


Figure 5.2: The Composition of the Two Agents

In combining two networks, one has to define the set of channels that will be identified in the composite system. This set of channels can be defined implicitly by assuming that the channels to be identified have the same names in both networks. The temporal properties of the composite system are then obtained through the conjunction of the properties of the components. The identification of two distinctly named channels can be achieved trivially by renaming one of them. In this case, we have to substitute the new name for the old one in the corresponding specification before combining the specifications. For instance, if Pr_1 and Pr_2 are the sets of temporal properties of $Agent_1$ and $Agent_2$, respectively, then the set of properties corresponding to the network in Figure 5.2 is $(Pr_1 \cup Pr_2[U \setminus R])$, where $Pr_2[U \setminus R]$ is obtained by substituting each occurrence of U by R in Pr_2 .

However, it is not always possible to combine two networks. Obviously, two channels can only be identified if one of them is an external input and the other an external output, and if they transmit data of the same type. Furthermore, we require that the networks to be composed have disjoint sets of agents. Formally, two networks $Net_1 = (Ag_1, St_1, Eq_1)$ and $Net_2 = (Ag_2, St_2, Eq_2)$ are said to be *composable* if the following hold:

- The networks have disjoint sets of agents, $Ag_1 \cap Ag_2 = \emptyset$.
- For each $S \in St_1 \cap St_2$, the type of S in Net_1 is the same as in Net_2 .
- For each $S \in St_1 \cap St_2$, S is an external input port in Net_1 (or Net_2) and an external output port in Net_2 (or Net_1). That is, S occurs in Eq_1 (or Eq_2) only on the left-hand side, and in Eq_2 (or Eq_1) only on the right-hand side.

If the above requirements hold, then the combination of Net_1 and Net_2 corresponds to their union:

$$Net_1 \cup Net_2 \stackrel{\text{def}}{=} (Ag_1 \cup Ag_2, St_1 \cup St_2, Eq_1 \cup Eq_2)$$

The composition rule states that the specification of a composed system can be constructed from the specifications of its components by combining the networks and joining the sets of temporal properties.

Composition Rule

$$\frac{\mathcal{P}_1 \mathbf{sat} \mathcal{S}_1 \quad ; \quad \mathcal{P}_2 \mathbf{sat} \mathcal{S}_2}{\mathcal{P}_1 \parallel \mathcal{P}_2 \mathbf{sat} \mathcal{S}_1 \oplus \mathcal{S}_2} \quad \text{if } Net_1 \text{ and } Net_2 \text{ are composable.}$$

Such a rule is often applied in a bottom-up development. In this development style, if the components \mathcal{P}_1 and \mathcal{P}_2 are already constructed, one aims to deduce properties of the composition of both processes. The rule given above says (implicitly) that every property that can be derived from the conjunction of the properties of the components should be a property of the composite system. In other words, the conjunction of the component properties represents an *upper bound* (w.r.t. strength) of the properties that can be satisfied by $\mathcal{P}_1 \parallel \mathcal{P}_2$. However, such a rule cannot be provided for most temporal logics, and especially not for those which include the “next-time” operator. Moreover, compared with composition rules for (extended) temporal logics [AL89, BK83, BKP84], the above composition rule is simpler and easier to apply. Because our specification technique is designed from the beginning for modular development, our composition rule is almost trivial, in contrast to some extended temporal logics in which composition rules are so complex that they are nearly incomprehensible.

In the following, we sketch the soundness of this rule. Let $\mathcal{S}_1 = \langle Dspec_1, Net_1, Pr_1 \rangle$ and $\mathcal{S}_2 = \langle Dspec_2, Net_2, Pr_2 \rangle$ be two specifications, and let \mathcal{P}_1 and \mathcal{P}_2 be processes with $\mathcal{P}_1 \mathbf{sat} \mathcal{S}_1$ and $\mathcal{P}_2 \mathbf{sat} \mathcal{S}_2$.

Now let (Alg_1, \mathcal{V}_1) , (Alg_2, \mathcal{V}_2) , and (Alg, \mathcal{V}) be the semantics of \mathcal{P}_1 , \mathcal{P}_2 , and $\mathcal{P}_1 \parallel \mathcal{P}_2$, respectively. In order to prove that $(\mathcal{P}_1 \parallel \mathcal{P}_2 \mathbf{sat} \mathcal{S}_1 \oplus \mathcal{S}_2)$, we have to prove that Alg is an algebra which satisfies $Dspec_1 \cup Dspec_2$, and that every formula in $Pr_1 \cup Pr_2$ is valid in every execution of $\mathcal{P}_1 \parallel \mathcal{P}_2$. That the algebra $(Alg = Alg_1 \cup Alg_2)$ is an algebra of $Dspec_1 \cup Dspec_2$ follows directly from the amalgamation lemma (see [EM85]).

Let $\sigma \in \mathcal{V}$ be an execution of $\mathcal{P}_1 \parallel \mathcal{P}_2$. According to the definition of the parallel composition given in Section 5.1, any execution of $\mathcal{P}_1 \parallel \mathcal{P}_2$ restricted to the alphabet of \mathcal{P}_1 is an execution of \mathcal{P}_1 ; thus we have: $\sigma \downarrow \alpha \mathcal{P}_1 \in \mathcal{V}_1$

Let Q now be a temporal formula included in the specification of the process \mathcal{P}_1 . The validity of the temporal formula Q in any execution sequence of $\mathcal{P}_1 \parallel \mathcal{P}_2$ depends only on the order of the events included in $\alpha \mathcal{P}_1$, because all temporal operators in Q are indexed with channels of \mathcal{P}_1 . It can be proved by induction on the structure of temporal formulas that the validity of a temporal formula in any sequence of events depends only on the order of events which occurred on the indices of this formula. That is, for any sequence of events σ and any indexed temporal formula F :

$$(Alg, \sigma) \models F \quad \text{iff} \quad (Alg, \sigma \downarrow Index(F)) \models F$$

where $Index(F)$ is the set of events performed on the channels that occur in F as indices. For example, the validity of the formula $(\square_S [A \mathbf{rcv} m])$ in a sequence σ depends only on the

sequence of events performed on S : $(\sigma \downarrow \{(S, v) \mid \text{for an arbitrary value } v\})$.

So we deduce that if Q is valid in $\sigma \downarrow \alpha\mathcal{P}_1$, then Q is also valid in σ . Hence, the parallel composition $\mathcal{P}_1 \parallel \mathcal{P}_2$ satisfies Q for every Q in Pr_1 . Thus, we establish that $\mathcal{P}_1 \parallel \mathcal{P}_2$ satisfies the specification Pr_1 . Analogously, we can show that $\mathcal{P}_1 \parallel \mathcal{P}_2$ satisfies the specification Pr_2 as well. Hence, the parallel composition $\mathcal{P}_1 \parallel \mathcal{P}_2$ satisfies $Pr_1 \cup Pr_2$. Therefore, the composition rule is sound.

5.3 Refinement

In this section, we show how the temporal logic defined in Section 3.3 also supports stepwise refinement of distributed systems. It is widely believed that stepwise refinement simplifies the development of systems and guarantees a safe way to achieve the desired (correct) system. During a development process which starts from an abstract specification, a sequence of more and more refined specifications is constructed. A specification in this sequence is considered as a refinement of the earlier specifications and, conversely, also as an abstraction of subsequent specifications. At each stage of the development process we are obliged to prove that the specification proposed is indeed a refinement of its predecessor. A refinement step is said to be *correct* if every implementation of the proposed specification is also an implementation of the preceding one. In this context, it is important that the formalism used offers the possibility of proving the correctness of refinement steps. We will introduce a rule for proving that one specification refines another.

Roughly speaking, a refinement is a development step in which one adds some details and design decisions to the original specification. In our approach, we follow principles successfully applied in the **KORSO**-methodology [PW94]: we replace a specification with a more concrete one that performs the same external behavior. In one step, a system refinement is achieved by refining one of its agents. An agent is refined to a network which should exhibit the same external behavior. As an example for a system refinement, we consider a transmission medium that receives and sends messages. At a high level of abstraction, this system is viewed as a single agent with an input and an output port.

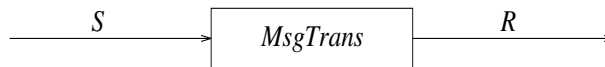


Figure 5.3: An Abstract Transmission Medium

At a lower level of abstraction, this transmission medium is described as a network made up of three agents, as depicted in Figure 5.4. In general, development steps, and especially refinements, are influenced by some specific implementation requirements. In our example, in refining *MsgTrans* we should take into account that the actual transmission medium can transmit only a specific class of fixed length (small) messages, called *packets*. In order to realize a complete message transmission over a packet transmission medium, one has to introduce an agent *MsgtoPck*, which splits messages into packets, and an agent *PcktoMsg*, which combines packets to make up messages.

In a refinement step, an agent is replaced by a network whose external input and output channels are the respective input and output channels of that agent. The replacement of an

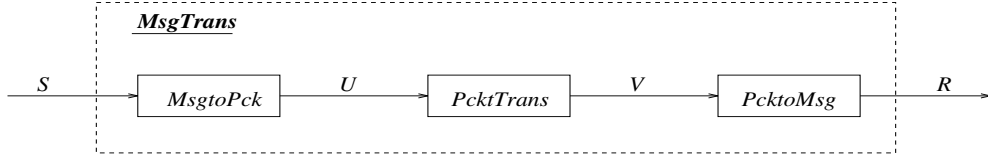


Figure 5.4: A Refinement of The Transmission Medium

agent by a network is formally described as follows.

Let $Net_1 = (Ag_1, St_1, Eq_1)$ be a network and A be an agent in Ag_1 with $A(S_1, \dots, S_n) = (R_1, \dots, R_m) \in Eq_1$. The agent A can be replaced by $Net = (Ag, St, Eq)$ in Net_1 , if the following requirements hold:

- Net_1 and Net have disjoint sets of agents, $Ag_1 \cap Ag = \emptyset$.
- The channels included in both Net_1 and Net are those of the agent A , $St_1 \cap St = \{S_1, \dots, S_n\} \cup \{R_1, \dots, R_m\}$
- The external input and output ports in Net are S_1, \dots, S_n , and R_1, \dots, R_m , respectively.

Following these requirements, the replacement of the agent A by Net in Net_1 , denoted as $Repl(Net, A, Net')$, yields a network Net_2 , defined by

$$\begin{aligned} Ag_2 &\stackrel{\text{def}}{=} (Ag_1 \setminus \{A\}) \cup Ag \\ St_2 &\stackrel{\text{def}}{=} St_1 \cup St \\ Eq_2 &\stackrel{\text{def}}{=} (Eq_1 \setminus \{A(S_1, \dots, S_n) = (R_1, \dots, R_m)\}) \cup Eq \end{aligned}$$

In refining a specification, we do not reify data types, in the sense that we define concrete data types for abstract ones, rather the originals are enriched by new data types. This enrichment is brought about by the addition of new channels which may transmit data of new sorts. In the example above, we need to enrich the specification of messages by a specification for packets. Moreover, we have to add an operation for splitting messages into packets and an operation for recombining packets.

In each refinement step the property part of the original specification is enriched by the property part of the network introduced. In order to accomplish the development step, we have to prove that the proposed specification is indeed a refinement of the original one. So we have to prove that every system which implements the resulting specification also implements the original. Formally:

Definition 5.3.1 [Refinement]

A specification $\mathcal{S}_2 = \langle (\Sigma_2, E_2), Net_2, Pr_2 \rangle$ is a refinement¹ of $\mathcal{S}_1 = \langle (\Sigma_1, E_1), Net_1, Pr_1 \rangle$, denoted as \mathcal{S}_2 **refines** \mathcal{S}_1 , if for every process \mathcal{P} , $\mathcal{P} \text{ sat } \mathcal{S}_2$ implies $\mathcal{P}|_{\Sigma_1} \text{ sat } \mathcal{S}_1$, where $\mathcal{P}|_{\Sigma_1}$ denotes the reduct of \mathcal{P} on the signature Σ_1 .

¹In the **KORSO**-methodology this is called “realization”.

In general, it is quite difficult to perform a refinement proof by reasoning about system behavior. One should be able to perform a refinement proof at the syntactical level. In our approach, the proof of refinement is based mainly on the proof system of temporal logic; refinement is expressed by logical consequence. The proof that a specification \mathcal{S}_2 is a refinement of \mathcal{S}_1 is reduced to the proof that $Dspec_2$ is an enrichment of $Dspec_1$, that Net_2 is obtained by replacing an agent A in Net_1 by a network Net , and that Pr_1 can be derived from Pr_2 . This leads to the following rule:

<p style="text-align: center;"><u>Refinement Rule</u></p> $\frac{Dspec_1 \subseteq Dspec_2 \quad ; \quad Net_2 = Repl(Net_1, A, Net) \quad ; \quad Pr_2 \vdash Pr_1}{\langle Dspec_2, Net_2, Pr_2 \rangle \text{ refines } \langle Dspec_1, Net_1, Pr_1 \rangle}$

According to the rule given above, the addition of a temporal property to the original specification is considered as a refinement. This provides a reasonable way of developing systems, because it is sometimes necessary to enrich the specification by design decisions (temporal properties) without modifying the structure of the system. Unfortunately, classical temporal logics do not support hierarchical development. In general, a temporal property with a next-time operator may lose its validity at a lower level: an action that is performed in one step at the highest level may be performed in several steps at a lower level.

In the following, we sketch the soundness of the refinement rule.

Let $\mathcal{S}_1 = \langle (\Sigma_1, E_1), Net_1, Pr_1 \rangle$ and $\mathcal{S}_2 = \langle (\Sigma_2, E_2), Net_2, Pr_2 \rangle$ be specifications, with $(\Sigma_1, E_1) \subseteq (\Sigma_2, E_2)$, $Net_2 = Repl(Net_1, A, Net)$, and $Pr_2 \vdash Pr_1$. Further, let \mathcal{P} be a process with $\llbracket \mathcal{P} \rrbracket = (Alg, \mathcal{V})$. Assume now that $\mathcal{P} \text{ sat } \mathcal{S}_2$: this implies that Alg is a (Σ_2, E_2) -algebra. Because $(\Sigma_1, E_1) \subseteq (\Sigma_2, E_2)$, we have $Alg|_{\Sigma_1}$ is a (Σ_1, E_1) -algebra. From $Pr_2 \vdash Pr_1$ it follows that for all $\sigma \in \mathcal{V}$

$$(Alg, \sigma) \models Pr_2 \quad \text{implies} \quad (Alg|_{\Sigma_1}, \sigma|_{\Sigma'}) \models Pr_1$$

This entails that $\mathcal{P}|_{\Sigma_1} \text{ sat } \mathcal{S}_1$. We conclude that $\mathcal{S}_2 \text{ refines } \mathcal{S}_1$.

In a stepwise development methodology, the development process can be viewed as a chain of specifications

$$\mathcal{S}_0 \text{ refines } \mathcal{S}_1 \text{ refines } \mathcal{S}_2 \dots \text{ refines } \mathcal{S}_n$$

where \mathcal{S}_0 is the initial requirements specification and \mathcal{S}_n the design specification, including the details and the design decisions needed to implement the desired system. At this stage of development the proof obligation is that \mathcal{S}_n is a refinement of \mathcal{S}_0 . An indirect way to prove this is to show that the refinement relation is transitive, that is, if a specification \mathcal{S} can be refined to a specification \mathcal{S}' , which can itself be refined to \mathcal{S}'' , then \mathcal{S} can be refined immediately to \mathcal{S}'' . This is the content of the following rule:

Transitivity Rule

$$\frac{\mathcal{S}_2 \text{ refines } \mathcal{S}_1 \quad ; \quad \mathcal{S}_3 \text{ refines } \mathcal{S}_2}{\mathcal{S}_3 \text{ refines } \mathcal{S}_1}$$

A refinement relation that is transitive is termed *vertically composable* [ZCdR92], because transitivity allows the composition of consecutive refinements. Another important property that should be satisfied by refinements is horizontal composability. A refinement relation is said to be *horizontally composable* if it is compatible with the building operations of specifications. That is, if a system can be composed into two subsystems, then refining the subsystems yields a refinement of the whole system.

Compatibility Rule

$$\frac{\mathcal{S}_1 \text{ refines } \mathcal{S}'_1 \quad ; \quad \mathcal{S}_2 \text{ refines } \mathcal{S}'_2}{\mathcal{S}_1 \oplus \mathcal{S}_2 \text{ refines } \mathcal{S}'_1 \oplus \mathcal{S}'_2}$$

This rule is very useful for the development of distributed systems, because it allows system components to be developed independently.

5.4 How to Develop Protocols

Generally, communication protocols within distributed systems are organized in a hierarchy of several layers. For example, the ISO-model for Open System Interconnection (OSI) consists of seven layers. In such a hierarchy, each layer provides services to the next higher layer by using the services of the next lower layer. At each layer one distinguishes between the services provided by this layer and the protocol (communication rules) used to ensure these services. The specification of the layer corresponds to the description of the services which are provided to the next higher layer, such as bidirectional transmission of messages, establishment of connections, etc. The specification of the protocol describes requirements on the behavior of the protocol entities such that they support the desired services by using the services offered by the next lower layer.

In our methodology, we develop the specification of a layer protocol in the following way: we start with a specification of the services provided by this layer. This specification represents the requirements specification in the development process. At this stage the layer is considered as an agent whose services are expressed by a set of temporal formulas. Such a specification has the following form:

SPECIFICATION Layer-i-Specification

IMPORT \ll *specifications describing the data transmitted by layer- i* \gg

SIGNATURE

FUN Layer-i : stream \times stream \rightarrow stream \times stream

NETWORK



PROPERTIES \ll *temporal formulas describing the behavior of the agent* \gg

Note that we evolve this kind of specification from an informal language, i.e. we translate informal requirements to the temporal language. Because there are no formal techniques for proving that informal and formal requirements specifications match, we follow general principles of requirements engineering (see, e.g. [Par91]). In developing formal requirements specifications from informal ones, we take two aspects of specifications into account: *completeness* and *consistency*. Completeness means that the formal specification reflects all informal requirements. Obviously, it is difficult to verify such a property formally. All we can do is check that informal safety and liveness properties have been translated to safety and liveness temporal formulas. Consistency means that there is no contradiction within the formal specification. In contrast to completeness, consistency can be formally verified by showing that at least one model satisfies the specification.

Thereafter, we specify the services provided by the next lower layer in the same way, i.e. based on an informal description. The main task to be accomplished now is to specify the protocol, i.e. how the services provided can be realized based on the services of the lower layer. This task corresponds to a refinement step, as described in Section 5.3. Usually, new agents which correspond to the protocol entities are added to the lower layer in such a way that the properties of $Layer_i$ can be deduced from the properties of the newly added agents and the properties of the lower layer. The obtained protocol specification is, thus, a refinement (realization) of the initial specification. The following specification can be viewed as a refinement of the $layer_i$, in which only two agents are added to the lower layer. This is a simple model; in general, a refinement will be more complicated.

SPECIFICATION Layer-i-Implementation

IMPORT \ll *specifications describing the data transmitted by layer- i* \gg

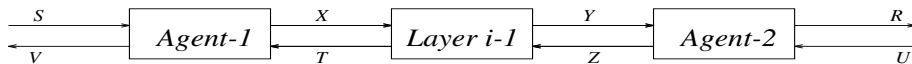
IMPORT Layer-(i-1)-Specification

SIGNATURE

FUN Agent-1 : stream \times stream \rightarrow stream \times stream

FUN Agent-2 : stream \times stream \rightarrow stream \times stream

NETWORK



PROPERTIES \ll *will be developed in performing the correctness proof* \gg

In contrast to traditional methods, in our approach the specifications of the added entities

are not given as gigantic pieces of text, rather are developed step by step. We start with a specification that does not include any requirement on the behavior of the newly introduced agents. Then, we try to perform the the correctness proof. We follow the usual procedure for proving theorems: in performing the proofs we determine which properties (about the behavior of the new agents) are needed to complete the proof. The specification of the protocol is then enriched by these properties. The protocol specification is constructed when all the properties of the original specification have been proved. The obtained specification is, therefore, correct.

Once a protocol specification has been developed, we proceed by replacing each formula in which more than one agent occurs with (equivalent) formulas, each including at most one agent. For example, assume that we have in our specification the following formula, which states that the *Agent*₁ (referring to the specification above) sends a message *m* until the *Agent*₂ receives this message.

$$(\text{Non} - \text{Imp1}) : \quad [\text{Agent}_1 \text{ snd } m]_X \text{until}_Y [\text{Agent}_2 \text{ rcv } m]$$

This is a temporal formula in which the behavior of *Agent*₁ depends on some action that may occur on *Agent*₂'s side. In order to be able to develop (implement) each of the agents independently, we should have a specification that describes each of the agents individually. Accordingly, we have to refine the properties referring to more than one agent to properties each of which refers at most to one agent. In our example, it is necessary, in order to be able to refine (Non - Imp1), to provide a mechanism that enables *Agent*₁ to know whether *Agent*₂ has received the transmitted message or not. This can be achieved by an acknowledgment mechanism: the agent *Agent*₂ acknowledges each received message. This can be formulated by:

$$(\text{Imp1}_1) : \quad \Delta_Y [\text{Agent}_2 \text{ rcv } m] \Rightarrow \Diamond_Z [\text{Agent}_2 \text{ snd Ack}(m)]$$

In so doing, the desired behavior can be realized if the agent *Agent*₁ retransmits the message *m* until it receives an acknowledgment of this message from *Agent*₂.

$$(\text{Imp1}_2) : \quad [\text{Agent}_1 \text{ snd } m]_X \text{until}_T [\text{Agent}_1 \text{ rcv Ack}(m)]$$

In contrast to the initial formula, each of the above two formulas refers to a single agent. Once these formulas have been established, we include them in our specification in place of the original. In this way, we obtain the desired protocol specification in which each agent is described individually, i.e. an agent occurs only in formulas that concern its individual behavior. This is a very important property, given that the agents will be implemented in separated locations.

Finally, we have to check the consistency of the developed protocol specification. This can be done by verifying that the specifications of the individual agents are consistent, and that there are no contradictions in the properties of the commonly used channels. Nevertheless, the implementability of the protocol should be ensured, in the sense that there are “practically computable” realizations of the agents. For example, we have to show that the behavior of agents can be realized with bounded memory.

Chapter 6

Applications

In this chapter, we demonstrate the application of the theory presented here by specifying three communication protocols. First, we specify a widely known classical protocol, namely the Alternating Bit protocol. Then, we specify a modern protocol used in local area networks, the CSMA/CD protocol. We conclude this chapter with the specification of the Three-Way Handshake, a protocol for establishing connections between transmission entities¹.

6.1 The Alternating Bit Protocol

The Alternating Bit (AB) protocol is widely known among researchers. Because of its simplicity, it has been used often in order to illustrate new methods in the area of protocol specification [Hai82, SMSV83, Lam83a]. The AB protocol guarantees a reliable bidirectional data transmission between two nodes in a distributed system. The services provided are based on an unreliable transmission medium which may corrupt messages or duplicate them, but can neither create nor permute them. Because the messages come in the correct order and because an appropriate mechanism for detecting corrupted messages is used, the reliability of data transmission is ensured by repeating transmission of a message until it is acknowledged. By adding an alternating control bit to each message sent, the receiver will be able to recover duplicated transmissions. This solution (alternating bit) has been proposed by Bartlett et al. in [BSW69].

In this section we develop an algebraic-temporal specification of the AB protocol. First, we give a specification of the services provided by the protocol. Then, we specify the transmission medium. Based on these two specifications, we develop temporal properties of the AB protocol step by step.

6.1.1 Specification of the AB Protocol

In this section, we describe the services provided by the AB protocol. Although this protocol actually provides bidirectional data transmission, for simplicity's sake we consider only uni-

¹In this chapter we usually try to work with the most general form of temporal axioms and theorems. This entails in particular that we try to minimize the number of indexed operators.

directional transmission. This simplification does not significantly change the problem, but it makes the proofs easier to understand. So we consider the system as an agent (AB) that provides reliable transmission of messages. Reliable transmission is characterized by the following three properties:

- messages are not created, i.e. every message sent must have been previously received,
- messages are not lost, i.e. every message received will eventually be sent,
- messages are sent in the same order as they were received.

The first and the third properties are safety properties, whereas the second describes a liveness property. These properties are translated into the temporal formulas AB_1 , AB_2 , and AB_3 in the following specification. The specification mainly describes the behavioral aspects of the system, whereas the concepts concerning the message layout are treated in another specification (`Message`). Such an algebraic specification describes the type of messages transmitted over the channels `S` and `R`. From this specification we need only import the sort `msg`.

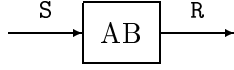
SPECIFICATION `AB-Specification`

IMPORT `Messages` ONLY `msg`

SIGNATURE

FUN `AB` : `stream[msg]` \rightarrow `stream[msg]`

NETWORK



PROPERTIES $\forall x, y : \text{msg}$

AXM (AB_1) : $[R \text{ xmt } x] \Rightarrow \blacklozenge [S \text{ xmt } x]$

AXM (AB_2) : $[S \text{ xmt } x] \Rightarrow \blacklozenge [R \text{ xmt } x]$

AXM (AB_3) : $(\blacklozenge [R \text{ xmt } x] \wedge [R \text{ xmt } y]) \Rightarrow \blacklozenge (\blacklozenge [S \text{ xmt } x] \wedge [S \text{ xmt } y])$

According to the discussion in Chapter 4, this specification corresponds to the properties listed above only under the assumption that the messages on the streams are distinct (see Section 4.2). Accordingly, we assume that the messages on the streams `S` and `R` are colored with distinct colors.

6.1.2 Specification of the Transmission Medium

The transmission medium is viewed as an agent that provides unreliable transmission of messages. Data may be corrupted or duplicated, but they are neither created nor permuted. We assume that detecting corrupted messages is a task that should be accomplished by the lower layer; that is, a corrupted message will be discarded and considered lost. However, we assume that a message cannot get lost infinitely many times. Hence, if a message is entered into the medium an infinite number of times, then it will eventually come through the medium.

The following specification describes the behavior of the transmission medium:

- Axiom (Med_1) is a safety property which states that legal messages cannot be created.
- Axiom (Med_2) is a liveness property. It states that a message cannot be lost forever.
- Axiom (Med_3)² states that messages are not permuted, i.e. the order of delivery is the order in which they are read.

SPECIFICATION `Medium`

IMPORT `Message` ONLY `msg`

SIGNATURE

FUN `Med`: `stream[msg]` \rightarrow `stream[msg]`

NETWORK



PROPERTIES $\forall x, y : \text{msg}$

AXM (Med_1): $[\text{Med } \text{snd } x] \Rightarrow \blacklozenge [\text{Med } \text{rcv } x]$

AXM (Med_2): $\square \blacklozenge [\text{Med } \text{rcv } x] \Rightarrow \blacklozenge [\text{Med } \text{snd } x]$

AXM (Med_3): $(\blacklozenge [\text{Med } \text{snd } x] \wedge [\text{Med } \text{snd } y]) \Rightarrow \blacklozenge (\blacklozenge [\text{Med } \text{rcv } x] \wedge [\text{Med } \text{rcv } y])$

As we will see in the next section, these conditions are sufficient to ensure a proper functioning of the AB protocol.

6.1.3 Formal Derivation of the AB Protocol

In Section 6.1.1, we presented a specification of the services to be provided by the AB protocol, without giving any details describing the internal structure of the system. In this section, we develop a more detailed specification, which describes how the AB protocol provides its services based on the much weaker services of the transmission medium. This development step corresponds to a refinement step, as described in Section 5.3. Hence, we add to the original system, which consists of the unreliable medium `Med`, two agents A and B, corresponding to the protocol entities of the AB protocol. The main task to accomplish is to design the behavior of the agents A and B such that externally we achieve reliable transmission of messages. We proceed as follows: we try to perform the correctness proof, and while doing so we determine which properties (about the behavior of the new agents) are needed in order to perform the proof.

The proof obligations are the above properties (AB_1), (AB_2), and (AB_3). We start with the following specification, which does not include any requirements on the behavior of the new agents A and B, but includes all requirements made on the behavior of the medium:

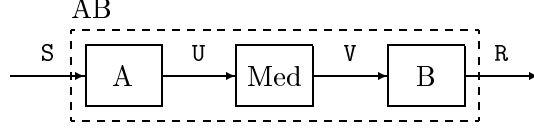
SPECIFICATION `AB-Implementation`

IMPORT `Medium` ONLY `Med msg`

SIGNATURE

²In order to illustrate different styles, we use here the other variant based on `snd` and `rcv` actions instead of `xmt`.

FUN A : $\text{stream}[\text{msg}] \rightarrow \text{stream}[\text{msg}]$
 FUN B : $\text{stream}[\text{msg}] \rightarrow \text{stream}[\text{msg}]$
 NETWORK



PROPERTIES \ll *will be developed later* \gg

This specification will be enriched by properties about the behavior of the agents A and B until the correctness proof is performed. The resulting final specification is then an implementation of the specification presented in Section 6.1.1.

Step 1: Establishing the Safety Property AB_1

In the following, we prove the safety property AB_1 . In order to perform this proof, we need two axioms (A_1) and (B_1) for the agents A and B. Note that the choice of these properties is completely up to us.

Proof of (AB_1):

$$\begin{aligned} & [R \text{ xmt } x] \\ \Rightarrow & \bullet_V [V \text{ xmt } x] && \text{[by axiom } B_1 \text{ below]} \\ \Rightarrow & \bullet_V \blacklozenge [U \text{ xmt } x] && \text{[by } Med_1] \\ \Rightarrow & \bullet_V \blacklozenge \bullet_S [S \text{ xmt } x] && \text{[by axiom } A_1 \text{ below]} \\ \Rightarrow & \blacklozenge [S \text{ xmt } x] && \text{[by temporal logic]} \\ & q.e.d. \end{aligned}$$

So it is necessary to extend the property part of the specification **AB-Implementation** by two axioms which state that A and B will only send what they have received.

ENRICH **AB-Implementation** BY

PROPERTIES $\forall x : \text{msg}$

$$\text{AXM } (A_1) : [A \text{ snd } x] \Rightarrow \bullet_S [A \text{ rcv } x]$$

$$\text{AXM } (B_1) : [B \text{ snd } x] \Rightarrow \bullet_V [B \text{ rcv } x]$$

Note that the axiom A_1 requires that the agent A can send only the last message received on the channel S. However, a weak version of this axiom, such as $[A \text{ snd } x] \Rightarrow \blacklozenge_S [A \text{ rcv } x]$, would be sufficient to perform the proof of the safety property. We have chosen the strong version for two reasons. On the one hand, the property A_1 guarantees that the order of messages is preserved by A. On the other hand, in connection with axiom A_3 (see Section 6.1.3), we ensure that the agent A can only start the transmission of a new message if the arrival of the previous message has been acknowledged.

Moreover, taking $[A \text{ snd } x] \Rightarrow \blacklozenge_S [A \text{ rcv } x]$ instead of the axiom A_1 would allow the agent A to be always ready to accept messages, which would be stored until they have been delivered.

Because it is not foreseeable how long it might take the agent A to get a frame through the medium, it would be necessary to have an unbounded amount of memory. Axiom A_1 , in contrast, ensures that the agent A only needs a memory of size one.

Step 2: Establishing the Liveness Property AB_2

The proof of the liveness property is more intricate than the proof of the safety property AB_1 . It turns out that we have to use strong design decisions in order to establish this property. To make the proof more structured, we start by proving the following *liveness lemma*, stating that everything sent by A reaches B.

THM ($Live_1$): $[A \text{ snd } x] \Rightarrow \Diamond \blacklozenge [B \text{ rcv } x]$

Proof of ($Live_1$):

We perform this proof by contradiction. We start from the two assumptions:

- (*) $[A \text{ snd } x]$
- (**) $\neg(\Diamond \blacklozenge [B \text{ rcv } x])$

Then we can perform the following deduction

$$\begin{aligned}
 & \neg(\Diamond \blacklozenge [B \text{ rcv } x]) && \text{[by (**)]} \\
 \Rightarrow & \Box \blacksquare \neg[B \text{ rcv } x] && \text{[by temporal logic]} \\
 \Rightarrow & \Box \Diamond [A \text{ snd } x] && \text{[by (*) and FeedBack below]} \\
 \Rightarrow & \Box \Diamond [Med \text{ rcv } x] && \text{[by temporal logic]} \\
 \Rightarrow & \Diamond [Med \text{ snd } x] && \text{[by Med}_2\text{]} \\
 \Rightarrow & \Diamond [B \text{ rcv } x] && \text{[by temporal logic]}
 \end{aligned}$$

contradiction!

q.e.d.

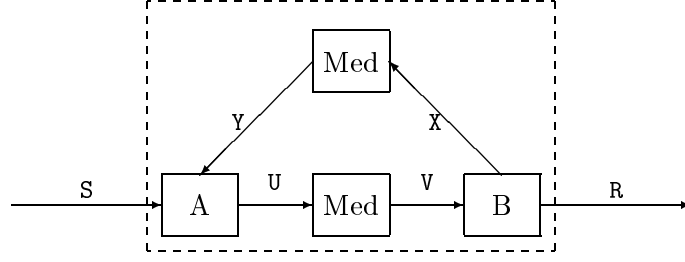
The proof of the property ($Live_1$) depends on the following lemma, which states that the agent A should infinitely often retransmit a message this message does not reach B. For methodological reasons, we characterize this lemma as a theorem, because we intend to base it on more fundamental (implementable) properties.

ENRICH AB-Implementation BY

PROPERTIES $\forall x : \text{msg}$

THM ($FeedBack$): $[A \text{ snd } x] \wedge \Box \blacksquare \neg[B \text{ rcv } x] \Rightarrow \Box \Diamond_U [A \text{ snd } x]$

Clearly, this property presents a problem, because it makes the behavior of the agent A dependent on something happening to B. This requires the introduction of some feedback which enables the agent B to send acknowledgements to A. This necessitates replacing the net of AB – Implementation with the following net



None of the theorems we have had so far is violated by this extension (thanks to the compositionality of our logic). Based on this design we can now replace the property `FeedBack` by implementable properties which describes the individual behavior of agents separately. We determine these properties in performing the following proof:

Proof of (FeedBack):

$$\begin{aligned}
 & [A \text{ snd } x] \wedge \square \blacksquare \neg[B \text{ rcv } x] \\
 \Rightarrow & [A \text{ snd } x] \wedge \square \blacksquare \blacksquare \neg[B \text{ snd } x] && \text{[by temporal logic]} \\
 \Rightarrow & [A \text{ snd } x] \wedge \square \blacksquare \neg\blacklozenge[B \text{ snd } x] && \text{[by temporal logic]} \\
 \Rightarrow & [A \text{ snd } x] \wedge \square \blacksquare \neg\blacktriangle_X[B \text{ snd } x] && \text{[by } B_2 \text{ below]} \\
 \Rightarrow & [A \text{ snd } x] \wedge \square \neg\blacklozenge_X[B \text{ snd } x] && \text{[by temporal logic]} \\
 \Rightarrow & [A \text{ snd } x] \wedge \square_Y \neg[Med \text{ snd } x] && \text{[by } Med_1 \text{]} \\
 \Rightarrow & [A \text{ snd } x] \wedge \square_Y \neg[A \text{ rcv } x] && \text{[by temporal logic]} \\
 \Rightarrow & \square \blacklozenge[A \text{ snd } x] && \text{[by } A_3 \text{ below]}
 \end{aligned}$$

q.e.d.

Together with two further properties of A and B, we thus obtain a proof for our theorem `AB2`.

$$\begin{aligned}
 & [S \text{ xmt } x] \\
 \Rightarrow & \blacklozenge[A \text{ snd } x] && \text{[by } A_2 \text{ below]} \\
 \Rightarrow & \blacklozenge\blacklozenge[B \text{ rcv } x] && \text{[by } Live_1 \text{]} \\
 \Rightarrow & \blacklozenge\blacklozenge[R \text{ xmt } x] && \text{[by } B_3 \text{ below]} \\
 \Rightarrow & \blacklozenge[R \text{ xmt } x] && \text{[by } Unq \text{]}
 \end{aligned}$$

q.e.d.

The last two proofs are based on the following properties for A and B. The first one states that A sends everything it receives on the input stream S. The second one states that A repeats a message forever, if it is not acknowledged. The third one states that B acknowledges only what it has received. And, finally, the fourth one says that B passes all messages through at least once.

ENRICH AB-Implementation BY

PROPERTIES $\forall x : \text{msg}$

$$\text{AXM } (A_2) : \blacktriangle_S[A \text{ rcv } x] \Rightarrow \blacklozenge_U[A \text{ snd } x]$$

$$\text{AXM } (A_3) : [A \text{ snd } x] \wedge \square_Y \neg[A \text{ rcv } x] \Rightarrow \square \blacklozenge[A \text{ snd } x]$$

$$\text{AXM } (B_2) : \blacktriangle_X[B \text{ snd } x] \Rightarrow \blacklozenge[B \text{ rcv } x]$$

$$\text{AXM } (B_3) : [B \text{ rcv } x] \Rightarrow \blacklozenge_{\blacklozenge_R}[B \text{ snd } x]$$

Ensuring the liveness of the whole system: The properties established above ensure that every message sent by A reaches B, but they do not guarantee that A eventually receives an

acknowledgment from B, because none of them claims that B acknowledges received messages. According to property (A₃), the agent A will retransmit an unacknowledged message forever. This entails that A would never be ready to accept a new message on S, because following property (A₁), A cannot receive a new message before the old one has been acknowledged. Therefore, in order to guarantee the liveness of the whole system, we have to add the following property saying that every message sent will eventually be acknowledged.

THM (Live₂): $[A \text{ snd } x] \Rightarrow \Diamond_Y [A \text{ rcv } x]$

Proof of Live₂:

We also perform this proof by contradiction, that is, we start from the two assumptions:

- (*) $[A \text{ snd } x]$
- (**) $\neg(\Diamond_Y [A \text{ rcv } x])$

Then we can perform the following deduction:

$$\begin{array}{ll}
& \neg(\Diamond_Y [A \text{ rcv } x]) & \text{[by (**)]} \\
\Rightarrow & \Box_Y \neg[A \text{ rcv } x] & \text{[by temporal logic]} \\
\Rightarrow & \Box \Diamond [A \text{ snd } x] & \text{[by (*) and A}_3\text{]} \\
\Rightarrow & \Box \Box \Diamond [A \text{ snd } x] & \text{[by temporal logic]} \\
\Rightarrow & \Box \Diamond [B \text{ rcv } x] & \text{[by Med}_2\text{]} \\
\Rightarrow & \Box \Diamond_X [B \text{ snd } x] & \text{[by B}_4\text{ below]} \\
\Rightarrow & \Diamond [A \text{ rcv } x] & \text{[by Med}_2\text{]}
\end{array}$$

contradiction!

q.e.d.

So it is necessary to augment the specification AB – Implementation with the following axiom, which states that B will eventually acknowledge the receipt of a message by sending it back.

ENRICH AB-Implementation BY

PROPERTIES $\forall x : \text{msg}$

AXM (B₄): $[B \text{ rcv } x] \Rightarrow \Diamond_X [B \text{ snd } x]$

Actually, the property (live₂) does not belong to the proof obligations. However, we think that this is a design decision that should be added in order to ensure the liveness of the whole system.

Step 3: Establishing the Safety Property AB₃

Each of the agents A, B, and Med preserves the order of messages. Therefore, we deduce immediately that the whole system satisfies the property of preserving the order of messages:

Proof of (AB₃):

$$\begin{array}{ll}
& \blacklozenge [R \text{ xmt } x] \wedge [R \text{ xmt } y] \\
\Rightarrow & \blacklozenge_R [B \text{ snd } x] \wedge \blacklozenge_R [B \text{ snd } y] & \text{[by temporal logic]} \\
\Rightarrow & \blacklozenge (\blacklozenge_V [B \text{ rcv } x] \wedge \blacklozenge_V [B \text{ rcv } y]) & \text{[by OrderB below]}
\end{array}$$

$$\begin{aligned}
&\Rightarrow \quad \blacklozenge (\blacklozenge_U [Med \mathbf{rcv} x] \wedge \blacktriangle_U [Med \mathbf{rcv} y]) && \text{[by Med}_3\text{]} \\
&\Rightarrow \quad \blacklozenge (\blacklozenge_S [A \mathbf{rcv} x] \wedge \blacktriangle_S [A \mathbf{rcv} y]) && \text{[by OrderA below]} \\
&\Rightarrow \quad \blacklozenge (\blacklozenge [S \mathbf{xmt} x] \wedge [S \mathbf{xmt} y]) \\
&q.e.d.
\end{aligned}$$

In the following, we prove the property that the agent B preserves the order of the received messages:

$$\begin{aligned}
\text{THM (OrderB):} \quad &(\blacktriangle_R [B \mathbf{snd} x] \wedge \blacklozenge_R [B \mathbf{snd} y]) \Rightarrow \blacklozenge (\blacktriangle_V [B \mathbf{rcv} x] \wedge \blacklozenge_V [B \mathbf{rcv} y]) \\
&\quad \blacklozenge_R [B \mathbf{snd} x] \wedge \blacktriangle_R [B \mathbf{snd} y] \\
\Rightarrow &\quad \blacklozenge \bullet_V [B \mathbf{rcv} x] \wedge \bullet_V [B \mathbf{rcv} y] && \text{[by B}_1\text{]} \\
\Rightarrow &\quad \bullet_V (peventiV[B \mathbf{rcv} x] \wedge [B \mathbf{rcv} y]) && \text{[by temporal logic]} \\
\Rightarrow &\quad \blacklozenge (\blacklozenge_V [B \mathbf{rcv} x] \wedge \blacktriangle_V [B \mathbf{rcv} y]) && \text{[by temporal logic]} \\
&q.e.d.
\end{aligned}$$

The proof of (Order_A) proceeds in the same way.

Step 4: Ensuring Implementability

A necessary condition for the implementability of the agents A and B is the *consistency* of their specifications. The properties (A₁)–(A₃) do not present any difficulties: one can trivially develop an implementation which meets all these properties. However, the specification of the agent B is a slightly more complicated situation. Due to the fact that the transmission medium may duplicate messages, and because the agent B cannot distinguish between duplicated messages and new ones, the agent B may send a message on R several times. This means that B may send messages with the same color on R, which contradicts the assumption that messages on R are distinctly colored.

Therefore, it is necessary to introduce a mechanism that enables the agent B to detect duplicate messages. We adopt the idea of alternating bits suggested in [BSW69], that is we index the messages on S alternately with 0 and 1.³ Thus we enrich the specification AB–Implementation with the following property claiming that two messages which are consecutively transmitted on S have inverted indices.

```

ENRICH AB-Implementation BY
IMPORT Bit
PROPERTIES  $\forall x, y : \text{msg}, \forall c, d : \text{bit}$ 
  AXM (S1):  $[S \mathbf{xmt} x_c] \wedge \tilde{O}_S [S \mathbf{xmt} y_d] \Rightarrow d = \bar{c}$ 

```

where Bit is the specification of indices:

```

SPECIFICATION Bit
SIGNATURE
  SORT bit
  FUN 0, 1 : bit

```

³This can be done by adding an agent which indexes incoming messages alternately with 0 and 1 and sends them on S.

$\text{FUN } \bar{} : \text{bit} \rightarrow \text{bit}$
 PROPERTIES
 AXM : $\bar{\bar{1}} = 0$
 AXM : $\bar{\bar{0}} = 1$

Following the properties (A₁) and (A₃), if A initiates the transmission of a certain message, it retransmits that message until it is acknowledged. Moreover, A can only initiate the transmission of a new message if the old one has been acknowledged. Hence, the index of messages sent by A changes iff the transmission of a new message is initiated. Because the order of messages is preserved by the transmission medium, duplicate messages come to B consecutively and with the same index, i.e. a change of the index means the arrival of a new message. Accordingly, in order not to duplicate messages, the agent B has to remember the index of the last received message and should behave as follows. If a received message has the same index as the last one received, then this is a duplicate and should be discarded. If the received message has a distinct index, this means that this is a new message and should immediately be sent on R. All we need to prove is that messages sent on R are distinctly colored.

THM (Cons) : $\forall c, d : \text{bit} \quad \Delta_R [B \text{ snd } x_c] \wedge \Diamond_R [B \text{ snd } y_d] \Rightarrow x \not\sim y$

We distinguish two cases:

1. **Case:** $c \neq d$

$$\begin{aligned}
 & \Delta_R [B \text{ snd } x_c] \wedge \Diamond_R [B \text{ snd } y_d] \\
 \Rightarrow & \bullet_V [B \text{ rcv } x_c] \wedge \Diamond_R \bullet_V [B \text{ rcv } y_d] && \text{[by axiom B}_1\text{]} \\
 \Rightarrow & \bullet_V ([B \text{ rcv } x_c] \wedge \Diamond [B \text{ rcv } y_d]) && \text{[by temporal logic, } (c \neq d)\text{]} \\
 \Rightarrow & \bullet_V (x \not\sim y) && \text{[by Med}_4\text{ below]}
 \end{aligned}$$

2. **Case:** $c = d$

$$\begin{aligned}
 & \Delta_R [B \text{ snd } x_c] \wedge \Diamond_R [B \text{ snd } y_c] \\
 \Rightarrow & \exists z : \Delta_R [B \text{ snd } x_c] \wedge \Diamond_R ([B \text{ snd } z_{\bar{c}}] \wedge \Diamond_R [B \text{ snd } y_c]) && \text{[by B}_5\text{ below]} \\
 \Rightarrow & \exists z : \bullet_V [B \text{ rcv } x_c] \wedge \Diamond (\bullet_V ([B \text{ rcv } z_{\bar{c}}] \wedge \Diamond [B \text{ rcv } y_c])) && \text{[by axiom B}_1\text{]} \\
 \Rightarrow & \exists z : \bullet_V ([B \text{ rcv } x_c] \wedge \Diamond ([B \text{ rcv } z_{\bar{c}}] \wedge \Diamond [B \text{ rcv } y_c])) && \text{[by temporal logic]} \\
 \Rightarrow & (x \not\sim z) && \text{[by Med}_4\text{ below]} \\
 \Rightarrow & (x \not\sim y) && \text{[by Med}_5\text{ below]} \\
 & q.e.d.
 \end{aligned}$$

In performing the proof of (Cons), we assumed the lemmas (B₅), (Med₄), and (Med₅). The property (B₅) states that between two messages with same index on R there is always a third one which has an inverted index. The second lemma requires that two messages transmitted on V with distinct indices must also be distinctly colored. Finally, (Med₅) states that the medium never transmits a distinctly colored message between two equivalently colored messages.

THM (B₅) : $\Delta_R [B \text{ snd } x_c] \wedge \Diamond_R [B \text{ snd } y_c] \Rightarrow$
 $\quad \exists z : \Delta_R [B \text{ snd } x_c] \wedge \Diamond_R ([B \text{ snd } z_{\bar{c}}] \wedge \Diamond_R [B \text{ snd } y_c])$
 THM (Med₄) : $[Med \text{ snd } x_c] \wedge \Diamond [Med \text{ snd } y_d] \wedge c \neq d \Rightarrow x \not\sim y$
 THM (Med₅) : $[Med \text{ snd } x] \wedge \Diamond ([Med \text{ snd } y] \wedge \Diamond [Med \text{ snd } z]) \wedge x \not\sim y \Rightarrow x \not\sim z$

As we will show, the proof of B_5 requires the enrichment of the specification, whereas the properties (Med_4) and (Med_5) can be directly deduced from the property of preserving the order of messages (Med_3) .

Proof of (B_5) :

$$\begin{aligned}
& \triangle_R [B \mathbf{snd} x_c] \wedge \diamond_R [B \mathbf{snd} y_c] \\
\Rightarrow & \triangle_R [B \mathbf{snd} x_c] \wedge \tilde{\circ}_R \diamond [B \mathbf{snd} y_c] && \text{[by Def. of } \diamond \text{]} \\
\Rightarrow & \exists z : \triangle_R [B \mathbf{snd} x_c] \wedge \tilde{\circ}_R [B \mathbf{snd} z_{\bar{c}}] \wedge \tilde{\circ}_R \diamond [B \mathbf{snd} y_c] && \text{[by axiom } B_6 \text{ below]} \\
\Rightarrow & \exists z : \triangle_R [B \mathbf{snd} x_c] \wedge \tilde{\circ}_R [B \mathbf{snd} z_{\bar{c}}] \wedge \tilde{\circ}_R \diamond [B \mathbf{snd} y_c] && \text{[by temporal logic]} \\
\Rightarrow & \exists z : \triangle_R [B \mathbf{snd} x_c] \wedge \tilde{\circ}_R ([B \mathbf{snd} z_{\bar{c}}] \wedge \diamond [B \mathbf{snd} y_c]) && \text{[by temporal logic]} \\
\Rightarrow & \exists z : \triangle_R [B \mathbf{snd} x_c] \wedge \diamond_R ([B \mathbf{snd} z_{\bar{c}}] \wedge \diamond_R [B \mathbf{snd} y_c]) && \text{[by temporal logic]} \\
& \textit{q.e.d.}
\end{aligned}$$

The proof of (B_5) requires that the agent B sends messages with alternating bits. So it is necessary to augment the specification $AB - \text{implementation}$ with the following axiom.

ENRICH AB -Implementation BY
 PROPERTIES $\forall x, y : \text{msg}, \forall c, d : \text{bit}$
 AXM (B_6) : $\triangle_R [B \mathbf{snd} x_c] \wedge \tilde{\circ}_R [B \mathbf{snd} y_d] \Rightarrow d = \bar{c}$

This completes the refinement of the AB protocol.

Taking Reality into Account

In our specification, we assumed that the transmission medium cannot filter out a message forever:

$$\text{AXM } (Med_2) : \quad \square \diamond [\text{Med } \mathbf{rcv} x] \Rightarrow \diamond [\text{Med } \mathbf{snd} x]$$

Accordingly, if necessary, the agent A would send a message infinitely many times in order to get it through the medium. Obviously, this is not a realistic assumption, because no agent can base its functioning on an infinite number of attempts. In practice, one uses a *time-out* strategy: after a certain amount of time or a certain number of attempts the transmission is abandoned and the next higher layer is notified that the transmission has failed.

Theoretically speaking, we merely achieve a “*weak implementation*”, that is, an implementation which either meets the requirements or at least returns an error indication.

6.2 The CSMA/CD Protocol

The CSMA/CD (Carrier Sense, Multiple Access with Collision Detection) protocol provides a medium access method intended for use in local area networks. Generally speaking, local area networks may present an access conflict when transmitting frames on the physical medium, because all connected stations share a common transmission line. Before initiating a transmission, a station asks whether the medium is free or not; if the medium is free, a frame transmission is initiated, otherwise the transmission is delayed. However, two stations may test the medium at the same time, and consequently they may transmit frames simul-

taneously. If two stations attempt to transmit frames at the same time, then these frames collide on the transmission line and, as a result, are (partially) lost. The main task of the CSMA/CD protocol is to enable reliable transmission of frames between the stations of a local area network.

The specification presented here is based on the standard definition [Ame85]. An overview of the protocol structure and its relationship to the OSI reference model is given in Figure 6.1. Following the standard definition, the function intended for the data-link layer is accomplished by two sublayers, the Logical Link Control (LLC) and the Media Access Control (MAC). In considering the CSMA/CD protocol, the MAC sublayer plays the most important role in providing the facilities for sending and receiving frames. Nevertheless, the services provided by the physical layer are the basis for the MAC sublayer to accomplish its function. Hence, in our development, we concentrate on the behavior of these two layers. First, we give separate specifications to each layer. Then, we develop a specification which describes how the MAC sublayer realizes its services with regard to the services provided by the physical layer.

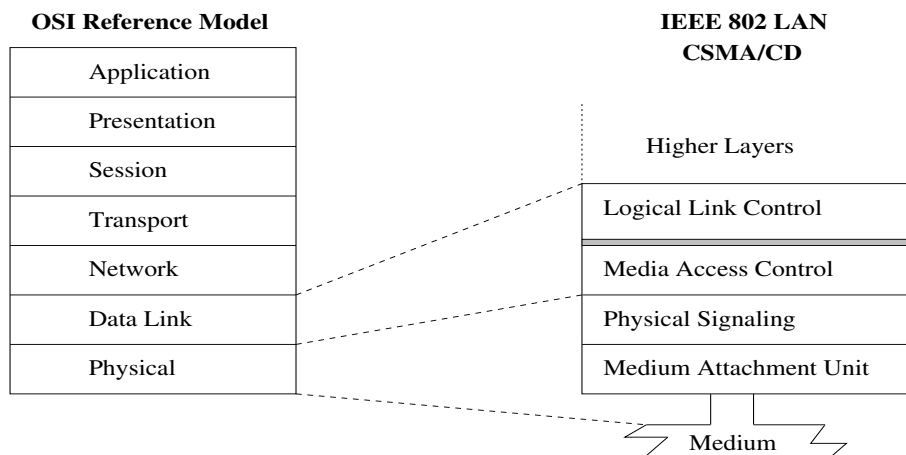


Figure 6.1: LAN Standard Relationship to the OSI Model

6.2.1 Specification of the CSMA/CD Protocol

In this section, we describe the services provided by the CSMA/CD MAC sublayer, that is, the services provided at the interfaces between the LLC and MAC sublayers. For simplicity, we consider a local area network with two stations. Following the standard definition, the MAC layers together with the lower layers and the physical medium enable bidirectional transmission of messages⁴ to the LLC entities. In practice, the transmission of frames is not always successful; after a transmission request the LLC sublayer is informed as to whether its data have been successfully transmitted or not. Moreover, the MAC sublayer may also pass illegal data to the LLC, which may consist of corrupted or incomplete messages. However, we do not consider failures in our specifications, rather investigate an idealized case in which the transmission is error-free. Hence, we view the system as an agent (CSMA/CD) that provides reliable bidirectional transmission of messages between the LLC entities. Reliable

⁴Messages correspond to the data passed from the LLC sublayer to the MAC sublayer.

transmission is characterized by the following three properties (just like in the requirements specification of the AB protocol):

- messages are not created, i.e. every message sent must have been previously received,
- messages are not lost, i.e. every message received will eventually be sent,
- messages are sent in the same order as they were received.

These properties are formulated in the following specification by the formulas Prop_1 , Prop_2 , and Prop_3 . The specification mainly describes the behavioral aspects of the system, whereas the concepts concerning the message layout are treated in another specification, `Message`. This algebraic specification describes the type of messages exchanged at the interfaces between the LLC and the MAC layers; it includes the description of legal as well as incomplete and corrupted messages. From this specification we need only import the sort of legal messages, `msg`.

SPECIFICATION CSMA/CD

IMPORT Message ONLY msg

SIGNATURE

FUN CSMA/CD : stream[msg] × stream[msg] → stream[msg] × stream[msg]

NETWORK



PROPERTIES $\forall m, n : \text{msg}$

AXM (Prop_1) : $[R \text{ xmt } m] \Rightarrow \blacklozenge [S \text{ xmt } m]$

AXM (Prop_2) : $[S \text{ xmt } m] \Rightarrow \blacklozenge [R \text{ xmt } m]$

AXM (Prop_3) : $\blacklozenge [R \text{ xmt } m] \wedge [R \text{ xmt } n] \Rightarrow \blacklozenge (\blacklozenge [S \text{ xmt } m] \wedge [S \text{ xmt } n])$

analogously for U and V

We should stress here again that this specification corresponds to the properties listed above only under the assumption that the messages on the streams are distinct (see Section 4.2). Accordingly, we assume that the messages on each of the streams `S`, `R`, `U`, and `V` are distinctly colored.

6.2.2 Specification of the Transmission Medium

The transmission medium consists of the physical layer together with the actual physical medium. It is seen as an agent (`M`) that provides unreliable bidirectional transmission of frames between the MAC entities. The unreliability is caused by the possibility of messages being destroyed whenever they collide. A collision arises if two frames are introduced to the medium simultaneously. In our linear-time logic, simultaneous occurrence of two actions is modeled by a nondeterministic interleaving. Thus, a collision arises if a frame is received by the medium before another frame previously received has been sent. Accordingly, we define

a predicate `Collision` as an abbreviation for the following temporal formula, which express that a collision has arisen.

$$\text{Collision} \stackrel{\text{def}}{=} [M \text{ rcv}] \wedge \tilde{\bullet}_M [M \text{ rcv}]$$

The messages passed at the interface between the physical layer and the MAC sublayer are called *frames*. Apart from a data field, a frame contains fields concerning e.g. destination and source addresses, frame length, and frame check sequence, which is needed to verify the validity of a frame. All these can, of course, be specified algebraically. We assume the existence of a specification, `Frame`, describing the layout of frames. For the description of the behavioral aspects of the medium, we import the sorts of legal frames (`frame`). Additionally, we import the constant symbol `coll`, which is used to indicate the occurrence of a collision, and the symbol `ackn`, needed to confirm successful transmission of a frame. In practice, transmission of a frame is automatically acknowledged if the collision-detect signal remains unactivated during the transmission of the frame.

The following specification describes the service provided by the transmission medium to the MAC entities. Based on the standard definition, the medium should fulfill the following properties:

- A characteristic property of the underlying network (Bus-System) is that if a frame enters the medium, then either it comes through undamaged or it collides with another frame. This property is formulated by the temporal formula M_1 .
- One of the main tasks of the medium is to detect a collision and to signal it immediately by sending the `coll` signal to the participating users. Hence, the property M_2 states that if a collision has occurred, then the constant `coll` is sent on the channels `b` and `d`.
- Legal frames may be lost but they should not be created. Hence, the property M_3 states that the medium can only send what it has received.
- The property M_4 ensures that the medium cannot confirm a frame before it has been successfully delivered, whereas property M_5 states that a successful transmission is acknowledged immediately after a frame has been delivered.

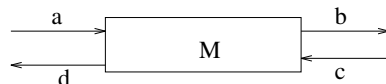
SPECIFICATION `Medium`

IMPORT `Frame ONLY frame coll ackn`

SIGNATURE

FUN `M`: `stream[frame] × stream[frame] → stream[frame] × stream[frame]`

NETWORK



PROPERTIES $\forall p, q : \text{frame} \setminus \{\text{coll}, \text{ackn}\}$

AXM (M_1): $\Delta_a [M \text{ rcv } p] \Rightarrow \text{Collision} \vee \tilde{\text{O}}_b [M \text{ snd } p]$

AXM (M_2): $\text{Collision} \Rightarrow \tilde{\text{O}}_b [M \text{ snd } \text{coll}] \wedge \tilde{\text{O}}_d [M \text{ snd } \text{coll}]$

$$\begin{aligned}
\text{AXM } (M_3) : & \quad \Delta_b [\mathbf{M\ snd\ p}] \Rightarrow \tilde{\bullet}_a [\mathbf{M\ rcv\ p}] \\
\text{AXM } (M_4) : & \quad \Delta_d [\mathbf{M\ snd\ ackn}] \Rightarrow [\mathbf{M\ rcv\ p}]_a \mathbf{before}_b [\mathbf{M\ snd\ p}] \\
\text{AXM } (M_5) : & \quad \Delta_b [\mathbf{M\ snd\ p}] \Rightarrow [\mathbf{M\ snd\ ackn}]_d \mathbf{before}_b [\mathbf{M\ snd\ q}]
\end{aligned}$$

analogously for direction $c \rightarrow d$

This specification describes the transmission of frames from the channel a to the channel b . The transmission from c to d works in the same way. Therefore, we should add to the above specification similar axioms to M_1 , M_3 , M_4 , and M_5 referring to the channels c and d . All we need to do is replace in these axioms the occurrences of a , b , and d by c , d , and b , respectively.

6.2.3 Formal Derivation of the Protocol Specification

In Section 6.2.1, we presented a specification that describes the services provided by the CSMA/CD protocol, without giving any internal details of the system. In this section we will take some design decisions and show how these services are provided, based on the services of the transmission medium.

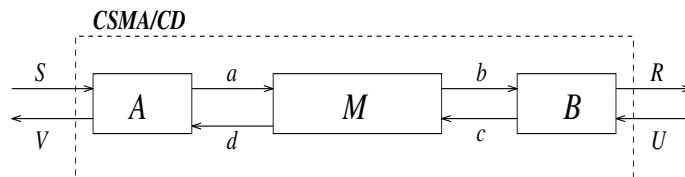
Once we decide to implement the CSMA/CD protocol, we have to turn our attention to the unreliability of the transmission medium. Thus, the main task we have to accomplish is to augment the original system consisting of the unreliable medium such that externally we get reliable transmission of messages. Therefore, we add to the medium two agents A and B which correspond to the protocol entities of the MAC sublayers. The next, even more important, step is to design the behavior of the newly introduced agents. We proceed as follows: we try to perform the correctness proof, and while doing so we determine which properties (about the behavior of the new agents) are needed in order to perform the proof.

The proof obligations are (Prop_1) , (Prop_2) , and (Prop_3) from the specification presented in Section 6.2.1. We start with the following specification, which does not include any requirements on the behavior of the new agents A and B , but includes all requirements made on the behavior of the medium:

```

SPECIFICATION CSMA/CD-Implementation
IMPORT Medium ONLY frame coll ackn M
IMPORT Message ONLY msg
SIGNATURE
  FUN A : stream[msg] × stream[frame] → stream[msg] × stream[frame]
  FUN B : stream[msg] × stream[frame] → stream[msg] × stream[frame]
NETWORK

```



PROPERTIES \ll *will be developed later* \gg

This specification will be enriched by properties about the behavior of the agents A and B until the correctness proof is performed. The resulting final specification is then an implementation of the specification presented in Section 6.2.1.

In communicating with the transmission medium, the agents A and B have to deal with frames, whereas in communicating with the external world they exchange messages. Thus, messages must be transformed into frames, and vice versa. The transformations are internal routines that should be performed by both agents A and B. For simplicity we assume that a message is encapsulated within one frame, and that a frame is decapsulated to one message. So, we add to the system specification the operation f , for transforming messages into frames, and the operation g , for transforming frames into messages. The main property we are interested in is that encapsulating a message and then decapsulating it yields the original message. Hence, we extend the system specification in the following way:

```
ENRICH CSMA/CD-Implementation BY
SIGNATURE
  FUN f : msg → frame
  FUN g : frame → msg
PROPERTIES ∀ m : msg
  AXM :   g(f(m)) = m
```

In order to ensure the consistency of our specifications, we have to assume that the functions f and g preserve the color of their arguments (see Section 4.5).

Step 1: Proof of the Safety Property: Prop₁

We want to establish the safety property saying that frames cannot be created. It turns out that we need to add requirements to the agents A and B which guarantee that they only send what they have received.

Proof of : $[R \mathbf{xmt} m] \Rightarrow \blacklozenge [S \mathbf{xmt} m]$

$$\begin{aligned}
& [R \mathbf{xmt} m] \\
\Rightarrow & \bullet_b [B \mathbf{rcv} f(m)] && \text{[by axiom B}_1 \text{ below]} \\
\Rightarrow & \blacklozenge_b [M \mathbf{snd} f(m)] && \text{[by temporal logic]} \\
\Rightarrow & \blacklozenge_b \bullet_a [M \mathbf{rcv} f(m)] && \text{[by axiom M}_3 \text{]} \\
\Rightarrow & \blacklozenge_b \bullet_a [A \mathbf{snd} f(m)] && \text{[by temporal logic]} \\
\Rightarrow & \blacklozenge_b \bullet_a \bullet_s [A \mathbf{rcv} g(f(m))] && \text{[by axiom A}_1 \text{ below]} \\
\Rightarrow & \blacklozenge [S \mathbf{xmt} m] && \text{[by temporal logic]} \\
& \textit{q.e.d.}
\end{aligned}$$

We have to extend the property part of CSMA/CD-Implementation by the following two axioms:

```
ENRICH CSMA/CD-Implementation BY
PROPERTIES ∀ p : frame , m : msg
  AXM : (A1)   Δa [A snd p] ⇒ ●s [A rcv g(p)]
  AXM : (B1)   ΔR [B snd m] ⇒ ●b [B rcv f(m)]
```

Step 2: Proof of the Liveness Property: Prop₂

In this section we deal with the proof of the property **Prop₂**, which is a liveness property stating that every frame on **S** will eventually be delivered on the stream **R**. In considering the transmission from **S** to **R**, the realization of this behavior is entirely the responsibility of the agent **A**, which should repeat the transmission of a frame until it has been delivered to **B**.

Proof of: $[S \text{ xmt } m] \Rightarrow \diamond [R \text{ xmt } m]$

$$\begin{aligned}
& [S \text{ xmt } m] \\
& \diamond_a [A \text{ snd } f(m)] && \text{[by axiom A}_2 \text{ below]} \\
\Rightarrow & \diamond_a ([A \text{ snd } (f(m)) \wedge \neg \text{Collision})] && \text{[by theorem nocoll below]} \\
\Rightarrow & \diamond_a \tilde{O}_b [M \text{ snd } f(m)] && \text{[by axiom M}_1 \text{]} \\
\Rightarrow & \diamond_b [M \text{ snd } f(m)] && \text{[by temporal logic]} \\
\Rightarrow & \diamond_b \diamond [R \text{ xmt } g(f(m))] && \text{[by axiom B}_2 \text{ below]} \\
\Rightarrow & \diamond [R \text{ xmt } m] && \text{[by temporal logic]} \\
& \text{q.e.d.}
\end{aligned}$$

This proof requires the extension of the specification **CSMA/CD-Implementation** by the liveness properties **A₂** and **B₂**; these state that the agents **A** and **B** should eventually send what they have received.

```

ENRICH CSMA/CD-Implementation BY
PROPERTIES  $\forall p : \text{frame}, m : \text{msg}$ 
  AXM : A2    $[S \text{ xmt } m] \Rightarrow \diamond_a [A \text{ snd } f(m)]$ 
  AXM : B2    $\triangleleft_b [B \text{ rcv } p] \Rightarrow \diamond [R \text{ xmt } g(p)]$ 

```

Further, we have to prove the property **nocoll**, stating that if the agent **A** starts the transmission of a frame, then eventually it reaches a state where no collision occurs. This entails that the frame will be delivered intact.

THM (nocoll): $\triangleleft_a [A \text{ snd } p] \Rightarrow \diamond_a ([A \text{ snd } p] \wedge \neg \text{Collision})$

Proof of: nocoll

We perform this proof by contradiction, beginning with the following assumption:

(*) : $[A \text{ snd } m] \wedge \square_a ([A \text{ snd } m] \Rightarrow \text{Collision})$

Then we can perform the following deduction:

$$\begin{aligned}
& [A \text{ snd } p] \wedge \square_a ([A \text{ snd } p] \Rightarrow \text{Collision}) && \text{[by (*)]} \\
\Rightarrow & [A \text{ snd } p] \wedge \text{Collision} \wedge \square_a ([A \text{ snd } p] \Rightarrow \text{Collision}) && \text{[by temporal logic]} \\
\Rightarrow & [A \text{ snd } p] \wedge \tilde{O}_d [M \text{ snd } \text{coll}] \wedge \square_a ([A \text{ snd } p] \Rightarrow \text{Collision}) && \text{[by axiom M}_2 \text{]} \\
\Rightarrow & [A \text{ snd } p] \wedge \tilde{O}_a [A \text{ snd } p] \wedge \square_a ([A \text{ snd } p] \Rightarrow \text{Collision}) && \text{[by axiom A}_3 \text{ below]} \\
\Rightarrow & \square_a [A \text{ snd } p] \wedge \square_a ([A \text{ snd } p] \Rightarrow \text{Collision}) && \text{[by temporal logic]} \\
\Rightarrow & \square_a \text{Collision} && \text{[by temporal logic]}
\end{aligned}$$

This contradicts axiom A₄ below!

q.e.d.

The proof of this property requires that the agent **A** retransmits a frame whenever it receives the constant **coll** indicating the occurrence of a collision. Furthermore, it is required that a

collision cannot occur continuously (as will be discussed in a moment). So it is necessary to augment the specification CSMA/CD – Implementation with the following axioms:

ENRICH CSMA/CD-Implementation BY

PROPERTIES $\forall p : \text{frame}$

$$\text{AXM (A}_3\text{)} : \quad \Delta_a [\mathbf{A\ snd\ } p] \wedge \tilde{\text{O}}_d [\mathbf{A\ rcv\ coll}] \Rightarrow \tilde{\text{O}}_a [\mathbf{A\ snd\ } p]$$

$$\text{AXM (A}_4\text{)} : \quad \neg \square_a \text{ Collision}$$

Axiom (A₄) needs an assessment w.r.t. its feasibility: Theoretically, it is possible for a collision to occur infinitely often, such that one cannot get frames through the medium. In order to establish the liveness property, it is therefore necessary to first assume that a collision cannot occur continuously. In practice, after a maximal number of attempts the transmission is abandoned and the LLC layer is informed that the transmission of its data has failed.

Step 3: Proof of the Safety Property: Prop₃

The property M₂ ensures that the transmission medium delivers frames in the same order as they have been received. On the other hand, the property of preserving the order of frames is also satisfied by both agents A and B. Therefore, we deduce immediately that the whole system satisfies this property.

Proof of: $\blacklozenge [R\ \mathbf{xmt}\ m] \wedge [R\ \mathbf{xmt}\ n] \Rightarrow \blacklozenge (\blacklozenge [S\ \mathbf{xmt}\ m] \wedge [S\ \mathbf{xmt}\ n])$

$$\begin{aligned} & \blacklozenge [R\ \mathbf{xmt}\ m] \wedge [R\ \mathbf{xmt}\ n] \\ \Rightarrow & \blacklozenge_R [B\ \mathbf{snd}\ m] \wedge \Delta_R [B\ \mathbf{snd}\ n] && \text{[by temporal logic]} \\ \Rightarrow & \blacklozenge (\blacklozenge_b [M\ \mathbf{snd}\ g(m)] \wedge \Delta_b [M\ \mathbf{snd}\ g(n)]) && \text{[by Order}_B \text{ below]} \\ \Rightarrow & \blacklozenge \blacklozenge (\blacklozenge_a [A\ \mathbf{snd}\ g(m)] \wedge \Delta_a [M\ \mathbf{snd}\ g(n)]) && \text{[by Order}_M \text{ below]} \\ \Rightarrow & \blacklozenge \blacklozenge \blacklozenge (\blacklozenge [S\ \mathbf{xmt}\ f(g(m))] \wedge [S\ \mathbf{xmt}\ f(g(n))]) && \text{[by Order}_A \text{ below]} \\ \Rightarrow & \blacklozenge (\blacklozenge [S\ \mathbf{xmt}\ m] \wedge [S\ \mathbf{xmt}\ n]) && \text{[by temporal logic]} \end{aligned}$$

q.e.d.

In the following, we prove the property stating that the agent B preserves the order of the received messages:

$$\begin{aligned} \text{THM (Order}_B\text{)} : \quad & (\Delta_R [B\ \mathbf{snd}\ m] \wedge \blacklozenge_R [B\ \mathbf{snd}\ n]) \Rightarrow \blacklozenge (\Delta_b [B\ \mathbf{rcv}\ g(m)] \wedge \blacklozenge_b [B\ \mathbf{rcv}\ g(n)]) \\ \Rightarrow & \blacklozenge_R [B\ \mathbf{snd}\ m] \wedge \Delta_R [B\ \mathbf{snd}\ n] \\ \Rightarrow & \blacklozenge \blacklozenge_b [B\ \mathbf{rcv}\ g(n)] \wedge \tilde{\blacklozenge}_b [B\ \mathbf{rcv}\ g(n)] && \text{[by axiom B}_1\text{]} \\ \Rightarrow & \tilde{\blacklozenge}_b (\blacklozenge_b [B\ \mathbf{rcv}\ g(m)] \wedge [B\ \mathbf{rcv}\ g(n)]) && \text{[by temporal logic]} \\ \Rightarrow & \blacklozenge (\blacklozenge_b [B\ \mathbf{rcv}\ g(m)] \wedge \Delta_b [B\ \mathbf{rcv}\ g(n)]) && \text{[by temporal logic]} \end{aligned}$$

q.e.d.

The properties Order_A and Order_M can be formulated and proved in a similar way.

Step 4: Ensuring Consistency

A necessary condition for the implementability of the agents A and B is the *consistency* of their specifications. This is no problem for A₁ – A₄; we can easily develop an implementation that meets all these requirements. The case is slightly different for the specification of B.

According to the assumption claiming that messages on the stream R are all distinct, the proper functioning of the agent B can only be guaranteed if the medium M does not duplicate frames on the stream b . Otherwise we may obtain the following situation, where M sends a duplicate of a frame which has already been transmitted on R . Then we deduce:

$$\begin{aligned}
& \exists p : \Delta_R [B \mathbf{snd} g(p)] \wedge \diamond_b [M \mathbf{snd} p] \\
\Rightarrow & \exists p : \Delta_R [B \mathbf{snd} g(p)] \wedge \diamond_R [B \mathbf{snd} g(p)] && \text{[by axiom B}_2\text{]} \\
\Rightarrow & \exists p : g(p) \not\sim g(p) && \text{[by Unq]} \\
\Rightarrow & \exists p : p \not\sim p \\
& \textit{Contradiction!} \\
& \textit{q.e.d.}
\end{aligned}$$

For this reason, we should ensure that the medium does not duplicate frames. Recalling the properties M_2 , M_4 , and M_5 , we can deduce that the medium cannot duplicate frames, as long as the agent A stops the retransmission of a frame once its transmission has been acknowledged. However, the properties $A_1 - A_4$ allow the agent A to continue retransmitting a frame, even if it has already been delivered to B . This may result in a frame being delivered several times. Hence, we have to establish the following property stating that the frames on the stream b are distinctly colored.

$$\text{THM (M}_6\text{)} : \Delta_b [M \mathbf{snd} p] \wedge \diamond_b [M \mathbf{snd} q] \Rightarrow p \not\sim q$$

Proof of unq:

$$\begin{aligned}
& \Delta_b [M \mathbf{snd} p] \wedge \diamond_b [M \mathbf{snd} q] \\
\Rightarrow & \Delta_b [M \mathbf{snd} p] \wedge \diamond_d ([M \mathbf{snd} \mathit{ackn}] \wedge \diamond_b [M \mathbf{snd} q]) && \text{[by axiom M}_5\text{]} \\
\Rightarrow & \Delta_b [M \mathbf{snd} p] \wedge \diamond_d ([M \mathbf{snd} \mathit{ackn}] \wedge \diamond_a [M \mathbf{rcv} q]) && \text{[by axiom M}_4\text{]} \\
\Rightarrow & \bullet_a [M \mathbf{rcv} p] \wedge \diamond_d ([M \mathbf{snd} \mathit{ackn}] \wedge \diamond_a [M \mathbf{rcv} q]) && \text{[by axiom M}_3\text{]} \\
\Rightarrow & \diamond (\diamond_a [A \mathbf{snd} p] \wedge [A \mathbf{rcv} \mathit{ackn}] \wedge \diamond_a [A \mathbf{snd} q]) && \text{[by temporal logic]} \\
\Rightarrow & \diamond (p \not\sim q) && \text{[by axiom A}_5\text{ below]} \\
\Rightarrow & p \not\sim q \\
& \textit{q.e.d.}
\end{aligned}$$

This leads to the extension of the properties part of the specification `CSMA/CD-Implementation` by the following axiom; it claims that the agent A never retransmits a certain frame once it has received an acknowledgment frame indicating that the frame has been delivered to B intact.

ENRICH `CSMA/CD-Implementation` BY

PROPERTIES $\forall p, q : \text{frame}$

$$\text{AXM (A}_5\text{)} : \blacklozenge_a [A \mathbf{snd} p] \wedge [A \mathbf{rcv} \mathit{ackn}] \wedge \diamond_a [A \mathbf{snd} q] \Rightarrow p \not\sim q$$

This completes the refinement of the `CSMA/CD` protocol.

6.2.4 Considering Failures

In order to present an easily comprehensible specification, we have made some simplifications. We have considered a system with only two stations and unidirectional communication, whereas the protocol is designed to be used with an arbitrary number of stations and bidirectional communication. Moreover, we have investigated an ideal situation where all agents

work perfectly. However, in the real world we have to deal with the possibility of the system failing, e.g. the network itself or the receiver breaking down. This would mean that a frame cannot always be successfully transmitted. In order to model the eventuality of failure in our specification, we have to modify the original CSMA/CD specification by replacing the liveness property Prop_2 by the following formula; it states that a frame is either eventually delivered or a failure is reported to the next higher layer:

$$[S \mathbf{xmt} m] \Rightarrow (\diamond [R \mathbf{xmt} m] \underline{\text{xor}} \diamond [V \mathbf{xmt} \text{fail}(m)])$$

where $\text{fail}(m)$ is a data element indicating the cause of failure in transmitting the message m . This can, of course, be specified algebraically.

6.3 A Transport Protocol

Transport protocols are designed to provide reliable communication between processes which must communicate over a less reliable medium, such as a packet-switching network, which may corrupt, lose, or duplicate packets, or deliver them in the wrong order. Apart from a data transfer protocol, a transport protocol includes a connection control protocol, which is needed to establish a connection if data have to be transmitted, and to terminate it after ensuring that all user data have been properly exchanged. In this section, we present a complete specification of a transport protocol, including a data-transfer protocol and a connection-establishment protocol.

6.3.1 Specification of the Transport Protocol

A transport protocol provides reliable bidirectional transmission of packets over an unreliable network. For simplicity, we consider communication only in one direction. So the system is viewed as an agent that neither creates, loses, nor permutes packets (see Section 6.1).

The following specification is similar to the one presented in Section 6.1.1 describing the services provided by the AB protocol. The main difference is that we assume, for technical reasons (see Section 6.3.5), that packets on the streams S and R are colored with nonequivalent colors, rather than distinct colors.

$$([S \mathbf{xmt} x] \wedge \diamond [S \mathbf{xmt} y] \Rightarrow x \not\sim y) \quad \text{and} \quad ([R \mathbf{xmt} x] \wedge \diamond [R \mathbf{xmt} y] \Rightarrow x \not\sim y)$$

Note that this modification does not change the actual behavior of the system. So the following specification is equivalent to the specification presented in Section 6.1.1: the two specifications are represented by the same class of models.

```

SPECIFICATION TP-Specification
IMPORT Messages ONLY msg
SIGNATURE
  FUN Transport : stream[msg] → stream[msg]
NETWORK

```



PROPERTIES $\forall x, y : \text{msg}$

$$\text{AXM (TP}_1\text{)} : [\mathbf{R\ xmt\ x}] \Rightarrow \blacklozenge [\mathbf{S\ xmt\ \tilde{x}}]$$

$$\text{AXM (TP}_2\text{)} : [\mathbf{S\ xmt\ x}] \Rightarrow \blacklozenge [\mathbf{R\ xmt\ \tilde{x}}]$$

$$\text{AXM (TP}_3\text{)} : ([\mathbf{R\ xmt\ x}] \wedge \tilde{\mathcal{O}}_r [\mathbf{R\ xmt\ y}]) \Rightarrow \blacklozenge ([\mathbf{S\ xmt\ \tilde{x}}] \wedge \tilde{\mathcal{O}}_s [\mathbf{S\ xmt\ \tilde{y}}])$$

The property (TP₁) is a safety property stating that messages cannot be created: the agent can only send what it has received. The second property (TP₂) is a liveness property. It ensures that every received message will eventually be sent. The property (TP₃) is a safety property which states that messages will be delivered in the same order as they were received.

6.3.2 Specification of the Transmission Medium

The network used by the transport protocol may be the source of a variety of errors. A packet may be lost, corrupted, or duplicated. In addition, the network may permute packets. We assume the existence of a mechanism for detecting corrupted packets, in order to make the protocol not responsible for dealing with them; so corrupted messages will be discarded and considered lost.

We model the network by a transmission medium that satisfies the following properties:

- Messages may be lost, duplicated, or permuted, but they cannot be created. Hence, the property (Net₁) below states that the net can only send what it has received.
- The second property (Net₂) states that messages cannot be filtered forever: if a message is entered into the medium infinitely many times then it will eventually pass through.

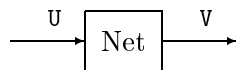
SPECIFICATION Network

IMPORT Message ONLY msg

SIGNATURE

$$\text{FUN Net} : \text{stream[msg]} \rightarrow \text{stream[msg]}$$

NETWORK



PROPERTIES $\forall x, y : \text{msg}$

$$\text{AXM (Net}_1\text{)} : [\mathbf{Net\ snd\ x}] \Rightarrow \blacklozenge [\mathbf{Net\ rcv\ \tilde{x}}]$$

$$\text{AXM (Net}_2\text{)} : \square \blacklozenge [\mathbf{Net\ rcv\ \tilde{x}}] \Rightarrow \blacklozenge [\mathbf{Net\ snd\ x}]$$

We assume that the messages on the streams U and V may have equivalent colors. In this case, the properties (Net₁) and (Net₂) are respectively equivalent to the properties (Med₁) and (Med₂) in the specification Medium presented in Section 6.1.2.

6.3.3 Formal Derivation of the Transport Protocol

In this section, we deal with the data-transfer part of the protocol: we concentrate on the specification of the protocol behavior after a connection has been established. Based on the specification **TP-Specification**, we develop a more detailed specification describing how the transport protocol provides its services using the services offered by the network. In our development step, we first add to the network two agents A and B , which represent the protocol entities; then we design the behavior of these agents such that externally we get reliable transmission of messages. So we try to perform the proofs of (TP_1) , (TP_2) , and (TP_3) , taking the properties (Net_1) and (Net_2) as axioms. In this way, we determine the properties (concerning the behavior of A and B , needed to complete the proofs.

The proofs of the properties (TP_1) and (TP_2) proceed in the same way as the proofs of the properties (AB_1) and (AB_2) performed in Section 6.1.3, because their proofs do not depend on whether the transmission medium preserves the order of messages or not. For this reason, we import the properties (about the behavior of A and B , and the design decisions established while proving the properties (AB_1) and (AB_2)): we start from the following specification:

SPECIFICATION TP-Implementation

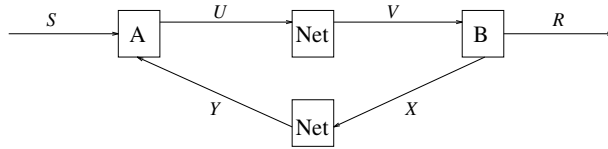
IMPORT Network ONLY Net msg

SIGNATURE

FUN A : stream[msg] \rightarrow stream[msg]

FUN B : stream[msg] \rightarrow stream[msg]

NETWORK



PROPERTIES $\forall x : \text{msg}$

AXM (A_1) : $[A \text{ snd } x] \Rightarrow \tilde{\bullet}_S [A \text{ rcv } \tilde{x}]$

AXM (A_2) : $\triangleleft_S [A \text{ rcv } x] \Rightarrow \diamond_U [A \text{ snd } \tilde{x}]$

AXM (A_3) : $[A \text{ snd } x] \wedge \square_V \neg [A \text{ rcv } \tilde{x}] \Rightarrow \square \diamond [A \text{ snd } \tilde{x}]$

AXM (B_1) : $\triangleleft_R [B \text{ snd } x] \Rightarrow \tilde{\bullet}_V [B \text{ rcv } \tilde{x}]$

AXM (B_2) : $\triangleleft_X [B \text{ snd } x] \Rightarrow \blacklozenge [B \text{ rcv } \tilde{x}]$

AXM (B_3) : $[B \text{ rcv } x] \Rightarrow \diamond \blacklozenge_R [B \text{ snd } \tilde{x}]$

AXM (B_4) : $[B \text{ rcv } x] \Rightarrow \diamond_X [B \text{ snd } \tilde{x}]$

Now it remains to prove the property (TP_3) claiming that messages will be delivered in the same order as they were received. Obviously, the behavior of the agents A and B , which are responsible for putting the messages in the right order, depends on whether the network permutes messages or not. Because the network used does not preserve the order of messages, the retrieval the order of the messages should follow other disciplines than those of the AB protocol. This will be discussed in the next section.

6.3.4 Preserving the Order of Messages

In this section, we will show that the behavior of the agents A and B can be arranged in such a way that the order of messages is retained during transmission. A network that may duplicate messages, lose them, or permute them offers no possibility of reconstructing on the receiver side the original order of messages, unless the receiver can at least recognize whether of two given messages one is the successor of the other or not.

Introducing Sequence Numbers

Generally, transport protocols cope with this problem by introducing so-called “sequence numbers” (see, e.g. [Ste76]), and descriptions of the protocol usually treat this aspect by indicating on which bit positions these sequence numbers are stored in the messages, and how long they are, and how they are accessed, modified, and set by the participating agents. But, following our overall principles, we consider such an approach to be at much too low-level and much too detailed. Therefore, we present a more abstract view (retaining, of course, the basic ideas of the solution), which is more amenable to our algebraic framework: *we index our messages with natural numbers*.

Further, we introduce the following predicates for sequence numbers i :

- $[A \text{ snd } x_i]$ means that A sends a message x that has the sequence number i . Analogously for $[B \text{ rcv } x_i]$ and for $[S \text{ xmt } x_i]$.
- $[A \text{ seq } i]$ means that the agent A is still transmitting a message with index i which has not yet been acknowledged.
- $[B \text{ exp } i]$ means that the agent B expects messages with sequence number i .

Intuitive Explanation

Before we begin with the formal derivations, we should get an intuitive feeling for the working of the protocol. The transmission of messages by the agents A and B now obeys the following rules (based on the network in Figure 6.2 below):

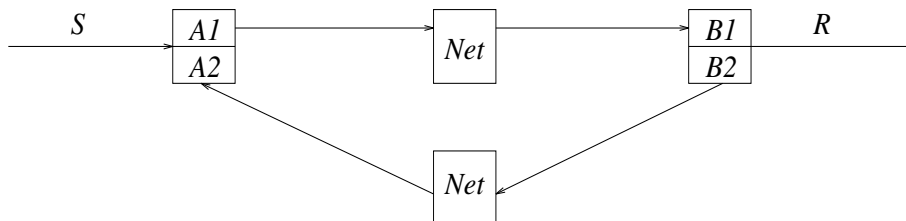


Figure 6.2:

- A is (repeatedly) sending messages with a sequence number expected by B (A maintains a window of size one). Consequently, the sub-agent B_2 is still acknowledging the previous sequence number.

- After B has received the expected message, it passes it on to its output stream R . Now the sub-agent B_1 is expecting messages with the next sequence number and B_2 starts to acknowledge the sequence number of the message just received.
- Eventually A will receive an acknowledgment for the messages it is still sending, which causes it to switch to the next sequence number. This enables A to receive the next message from its input stream S , and the game starts all over again.

Actually, we assume in this game that the messages on the input stream S are indexed in the right order, and that we output them to R also in indexed form. But, we could equally well consider the messages on these two streams without sequence numbers; adding sequence numbers is then a completely internal affair, because A adds numbers to the messages and B removes them. (Adding sequence numbers to the overall system is again a matter of algebraic data specifications.)

Formal Derivation

Based on the intuitive explanation above, we can now relatively easily express the necessary properties and perform the pertinent proofs. First of all, we assume that the sequence numbers simply express the sequence of the messages on the input stream S :

```
ENRICH TP-Implementation BY
IMPORT Seq-Number ONLY < suc
PROPERTIES  $\forall i, j : \text{Seq-Number}, \forall x, y : \text{msg}$ 
  AXM ( $S_1$ ):  $[S \mathbf{xmt} x_i] \wedge \tilde{O}_S [S \mathbf{xmt} y_j] \Rightarrow j = \text{suc}(i)$ 
```

This implies immediately the following property:

THM ($Order$): $(j = \text{suc}(i) \wedge \blacklozenge [S \mathbf{xmt} x_i] \wedge \blacklozenge [S \mathbf{xmt} y_j]) \Leftrightarrow \blacklozenge ([S \mathbf{xmt} x_i] \wedge \tilde{O}_S [S \mathbf{xmt} y_j])$

Preservation of the order can be proved if the messages on the stream R obey the property $Order$. This property follows immediately from the following theorem:

THM (R_1): $[R \mathbf{xmt} x_i] \wedge \tilde{O}_R [R \mathbf{xmt} y_j] \Rightarrow j = \text{suc}(i)$

Based on this lemma, we can now easily prove the property of order preservation (TP_3).

Proof of (TP_3):

$$\begin{aligned}
& [R \mathbf{xmt} x_i] \wedge \tilde{O}_R [R \mathbf{xmt} y_j] \\
\Rightarrow & \blacklozenge ([R \mathbf{xmt} x_i] \wedge \tilde{O}_R [R \mathbf{xmt} y_j]) && \text{[by temporal logic]} \\
\Rightarrow & j = \text{suc}(i) \wedge \blacklozenge [R \mathbf{xmt} x_i] \wedge \blacklozenge [R \mathbf{xmt} y_j] && \text{[by Order]} \\
\Rightarrow & j = \text{suc}(i) \wedge \blacklozenge [S \mathbf{xmt} \tilde{x}_i] \wedge \blacklozenge [S \mathbf{xmt} \tilde{y}_j] && \text{[by TP}_1\text{]} \\
\Rightarrow & \blacklozenge ([S \mathbf{xmt} \tilde{x}_i] \wedge \tilde{O}_S [S \mathbf{xmt} \tilde{y}_j]) && \text{[by Order]} \\
& q.e.d.
\end{aligned}$$

In the following, we prove the property (R_1):

Proof of (R₁):

$$\begin{aligned}
& [R \mathbf{xmt} x_i] \wedge \tilde{\mathcal{O}}_R [R \mathbf{xmt} y_j] \\
\Rightarrow & [B \mathbf{exp} suc(i)] \wedge \tilde{\mathcal{O}}_R [R \mathbf{xmt} y_j] && \text{[by axiom B}_5 \text{ below]} \\
\Rightarrow & [B \mathbf{exp} suc(i)] \mathbf{unless} [B_1 \mathbf{snd} x_{suc(i)}] \wedge \tilde{\mathcal{O}}_R [R \mathbf{xmt} y_j] && \text{[by Inv below]} \\
\Rightarrow & [B \mathbf{exp} suc(i)] \mathbf{until} [B_1 \mathbf{snd} y_j] && \text{[by temporal logic]} \\
\Rightarrow & \diamond ([B \mathbf{exp} suc(i)] \wedge [B \mathbf{exp} j]) && \text{[by axiom B}_5 \text{ below]} \\
\Rightarrow & j = suc(i) \\
& q.e.d.
\end{aligned}$$

The property *Inv* below states that the agent B should retain its expected sequence number until it sends a message indexed with this number. This property follows immediately from the axiom (B_6) below.

$$\text{THM (Inv)} : [B \mathbf{exp} i] \Rightarrow [B \mathbf{exp} i] \mathbf{unless} [R \mathbf{xmt} x_i]$$

This leads to the enrichment of the specification **TCP-Implementation** by the following two axioms. The first states that B immediately switches to the next sequence number after sending the message with the expected number. The second axiom claims that the expected number is the successor of the number of the message previously delivered on R .

ENRICH TP-Implementation BY

PROPERTIES

$$\text{AXM (B}_5\text{)} : [B_1 \mathbf{snd} x_i] \Rightarrow [B \mathbf{exp} suc(i)] \wedge \tilde{\bullet}_B [B \mathbf{exp} i]$$

$$\text{AXM (B}_6\text{)} : [B \mathbf{exp} suc(i)] \Rightarrow [R \mathbf{xmt} x_i] \vee \tilde{\bullet}_R [R \mathbf{xmt} x_i]$$

This essentially concludes the derivation: we have set up specifications for our agents A and B that ensure the desired behavior of the overall network. What remains to be done is to check whether these specifications are feasible.

Ensuring Implementability

The first issue to be considered about the specifications of A and B is their *consistency*. This is no problem for the properties (A_1) – (A_3) and (B_1) – (B_6); one can easily check that there are models for these specifications. Another important constraint that we have to check is the boundness of the memories of both A and B .

The properties (A_1) and (A_3) guarantee that the agent A can accept a new message from its input stream S only if the previous message has already been acknowledged. This means that the agent A maintains a window of size one and needs to store only the last message received. On the other side, the agent B needs only to remember the expected sequence number, because, if the agent B receives a message x_i , then either i is less than the expected sequence number, i.e. x_i has already been received and delivered, or i is the expected sequence number. It is impossible for i to be greater than the expected sequence number, because the message x_i cannot have been sent by A if B is still expecting a message with index i .

Moreover, it is necessary, for the proper functioning of the data-transfer protocol, that the agents A and B initially agree on the same sequence number, that is A starts to send messages with the sequence number expected by B . To achieve this, we need the so-called three-way

handshake protocol.

Summary of the Stenning Protocol

Summing up, we obtain the following specification for the agents A and B , which together constitute the so-called *Stenning protocol* [Ste76]. In this specification, we have already included a little optimization: for the acknowledgment it is clearly sufficient for the sequence number, and not the whole message, to be sent back. (We have marked those properties that follow the other properties with an asterisk).

SPECIFICATION TP-Implementation

⋮

PROPERTIES

$$\text{AXM } (S_1): [S \mathbf{xmt} x_i] \wedge \tilde{O}_S [S \mathbf{xmt} y_j] \Rightarrow j = \text{succ}(i)$$

$$\text{AXM } (A_1): [A \mathbf{snd} x_i] \Rightarrow \tilde{\bullet}_S [A \mathbf{rcv} \tilde{x}_i]$$

$$\text{AXM } (A_2): \triangle_S [A \mathbf{rcv} x_i] \Rightarrow \diamond_U [A \mathbf{snd} \tilde{x}_i]$$

$$\text{AXM } (A_3^*): [A \mathbf{snd} x_i] \wedge \square_Y \neg[A \mathbf{rcv} i] \Rightarrow \square \diamond [A \mathbf{snd} \tilde{x}_i]$$

$$\text{AXM } (B_1): \triangle_R [B \mathbf{snd} x_i] \Rightarrow \tilde{\bullet}_V [B \mathbf{rcv} \tilde{x}_i]$$

$$\text{AXM } (B_2^*): \triangle_X [B \mathbf{snd} i] \Rightarrow \blacklozenge [B \mathbf{rcv} x_i]$$

$$\text{AXM } (B_3): [B \mathbf{rcv} x_i] \Rightarrow \diamond \blacklozenge_R [B \mathbf{snd} \tilde{x}_i]$$

$$\text{AXM } (B_4^*): [B \mathbf{rcv} x_i] \Rightarrow \diamond_X [B \mathbf{snd} i]$$

$$\text{AXM } (B_5): [B_1 \mathbf{snd} x_i] \Rightarrow [B \mathbf{exp} \text{succ}(i)] \wedge \tilde{\bullet}_B [B \mathbf{exp} i]$$

$$\text{AXM } (B_6): [B \mathbf{exp} \text{succ}(i)] \Rightarrow [R \mathbf{xmt} x_i] \vee \tilde{\bullet}_R [R \mathbf{xmt} x_i]$$

Here, we have provided here a very rigorous specification, which makes an infinite number of attempts in the case of failure. In practice, one has a variant of this specification, which works with a timeout mechanism. That is, after a certain number of retransmissions without acknowledgments a failure is reported to the sender.

6.3.5 The Three-Way Handshake

Generally, in computer networks, connections between protocol entities are not maintained permanently, rather a connection will be established whenever the entities have data to exchange, and will be closed if all user data have been properly transmitted. The purpose of this section is the specification of the connection-establishment protocol. One of the primary functions of this protocol is to ensure that the agents A and B initially agree on the same sequence number. Unfortunately, this initial agreement also has to be established via the unreliable medium *Net*, which slightly complicates the process. In short, we need a double acknowledgment in order to guarantee that an agreement has indeed been reached.

Intuitive Explanation

Let us again illustrate the working of the protocol on intuitive grounds, before we commence with formal derivations and proofs. Under normal circumstances, the connection establishment proceeds as follows:

- The sender A initiates the establishment of a connection by sending a special message (“Init”), indicating A ’s initial sequence number (Figure 6.3).

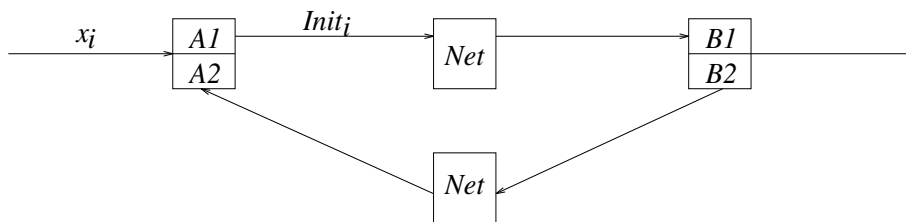


Figure 6.3:

- As long as the receiver B is not connected to A , every received message except an initiating one will be discarded. Intuitively speaking, if B receives an initiating message, it adopts its sequence number and acknowledges the receipt of the initiating message.

Unfortunately, this does not always work, due to the unreliability of the medium Net . A failure may arise if the agent B receives an old initiating message while A is trying to establish a new connection. This may lead to the following situation: immediately after receiving an old initiating message, the agent B receives an old message with the expected number. Accordingly, B will deliver this message, although it is obviously a duplicate of one which has already been delivered.

It is therefore very important for B to be able to distinguish between an old and a new initiating message. Hence, if B is not connected to A and receives an initiating message, it must first find out whether it is a left-over or not. We propose the following establishment process, based on protocols presented in [Tom75] and [SD78]:

- If B is not connected and receives an initiating message with the sequence number i , then B adopts this number and selects a second number j (its own), which it has never used before, i.e. B retains a pair of sequence numbers. From now on, B acknowledges the receipt of the initiating message by sending the pair $(i|j)$ until a message indexed with this combination arrives; this is clearly a new message, because j is unique.
- When A receives the acknowledgment $(i|j)$, it sets its sequence number to $(i|j)$ and the normal transmission (as described earlier) can start, with the exception that in the start phase A sends the messages which were originally indexed with i with the combination index $(i|j)$.
- If B receives a message indexed with $(i|j)$, it delivers the message, switches to the successor of i , and acknowledges i . From this point on, the transmission proceeds exactly as described in Section 6.3.4.

The greatest potential danger in this arrangement arises when, during the set-up process, B acknowledges – due to delays in the network – an earlier connection establishment. Then we encounter a situation of the following kind:

- During the set-up phase, the sender A receives a wrong acknowledgment from B , which tries to confirm an outdated connection message.
- A reacts to such erroneous messages by sending a “Reinit” message, carrying a number combination of the number that B apparently is looking for and the original sequence number selected by A . And B simply ignores the initiation attempts from A , because it is not yet connected.
- Upon receipt of the “Reinit” message (with the correct sequence number), B adopts the new number combination, because it is carrying the number that B has newly chosen. Hence, we establish the “normal” situation described above.

The “Init” messages will be replaced by the “Reinit” messages. The difference between them is that “Init” messages are indexed with a single sequence number, the “Reinit” messages with a pair of numbers.

Formal Derivation

Based on this intuitive understanding, we may now start with the formal derivation. This shall in particular clarify the exact requirements needed for the agents A and B , thus providing a sufficiently precise specification of their behavior. The goal we have to establish is the property stating that when the sender A initiates a connection and the receiver B is not connected ($[B \mathbf{exp} \perp]$), then both agents will eventually agree on the “same” sequence number; that is

$$\text{THM (TP}_4\text{)} : [A \mathbf{snd} \mathit{Init}_i] \wedge [B \mathbf{exp} \perp] \Rightarrow \exists j : \diamond([A \mathbf{seq} i|j] \wedge [B \mathbf{exp} i|j])$$

This is a *liveness* property; its derivation therefore depends on the feedback facility. But, of course, we can take advantage of the properties that have already been demonstrated in previous sections.

Proof of (TP₄): We have to distinguish two cases, viz. a nonconnected agent B (i.e. B is not expecting any sequence number) and an already connected agent B .

- **Case1:** Init_i is delivered to B without any delay, that is before any old initiating message arrives at B . In this case Init_i reaches B when B is not connected. Hence, we have

$$\begin{aligned} & [A \mathbf{snd} \mathit{Init}_i] \wedge [B \mathbf{exp} \perp] \\ \Rightarrow & [A \mathbf{snd} \mathit{Init}_i] \wedge \diamond([B \mathbf{exp} \perp] \wedge [B \mathbf{rcv} \mathit{Init}_i]) && \text{[by TP}_1\text{]} \\ \Rightarrow & [A \mathbf{snd} \mathit{Init}_i] \wedge \diamond(\exists j : [B \mathbf{exp} i|j]) && \text{[by axiom B}_7\text{ below]} \\ \Rightarrow & \exists j : \diamond([A \mathbf{seq} i|j] \wedge [B \mathbf{exp} i|j]) && \text{[by Connect below]} \end{aligned}$$

- **Case2:** An old initiating message Init_k reaches B before Init_i . We deduce:

$$\begin{aligned}
& \blacklozenge [A \text{ snd } x_k] \wedge \diamond ([B \text{ exp } \perp] \wedge [B \text{ rcv } \text{Init}_k]) \\
\Rightarrow & \diamond [B \text{ exp } k|j] \wedge \blacklozenge [A \text{ snd } x_k] && \text{[by axiom B}_7 \text{ below]} \\
\Rightarrow & \diamond [B \text{ exp } k|j] \wedge \square \neg [A \text{ snd } x_k|j] && \text{[by Unique below]} \\
\Rightarrow & \diamond ([B \text{ exp } k|j] \wedge \square \neg [B \text{ rcv } x_k|j]) && \text{[by temporal logic]} \\
\Rightarrow & \diamond (\square \diamond [B_2 \text{ snd } k|j] \wedge [B \text{ exp } j|k]) && \text{[by axiom B}_8 \text{ below]} \\
\Rightarrow & \diamond (\square \diamond [A_2 \text{ rcv } k|j] \wedge [B \text{ exp } k|j]) && \text{[by Thm}_2\text{]} \\
\Rightarrow & \diamond (\square \diamond [A_1 \text{ snd } \text{Reinit}_{i|j}] \wedge [B \text{ exp } k|j]) && \text{[by axioms A}_4, \text{ A}_5 \text{ below]} \\
\Rightarrow & \diamond (\diamond [B \text{ rcv } \text{Reinit}_{i|j}] \wedge [B \text{ exp } k|j]) && \text{[by IP}_1\text{]} \\
\Rightarrow & \diamond ([B \text{ rcv } \text{Reinit}_{i|j}] \wedge [B \text{ exp } k|j]) && \text{[by axiom B}_5\text{]} \\
\Rightarrow & \diamond [B \text{ exp } i|j] && \text{[by axiom B}_9 \text{ below]} \\
\Rightarrow & \diamond ([A \text{ seq } i|j] \wedge [B \text{ exp } i|j]) && \text{[by Connect below]} \\
& \text{q.e.d.}
\end{aligned}$$

This leads to the enrichment of the specification **TP-Implementation** with the following axioms. Axiom (A₄) states that if the agent A is trying to establish a connection, then A continues to do so until it receives an acknowledgment of its connection request. Axiom (A₅) claims that if A receives an unexpected acknowledgment, it reacts immediately by sending a “Reinit” message. Axiom (B₇) states that if B is not connected and receives an “Init” message, then B adopts the sequence number of this message. Axiom (B₈) ensures that B acknowledges the pair $(i|j)$ until it receives a message indexed with this pair. And, finally, axiom (B₉) states that if B receives a “Reinit” message whose second index is expected by B , then B adopts the pair of sequence numbers arriving with the “Reinit” message.

ENRICH TP-Implementation BY

PROPERTIES

$$\begin{aligned}
\text{AXM (A}_4\text{)} &: [A \text{ seq } \text{Init}_i] \wedge \square \neg [A \text{ rcv } i|j] \Rightarrow \square [A \text{ seq } \text{Init}_i] \\
\text{AXM (A}_5\text{)} &: [A \text{ seq } \text{Init}_i] \wedge [A_2 \text{ rcv } k|j] \Rightarrow \diamond [A_1 \text{ snd } \text{Reinit}_{i|j}] \\
\text{AXM (B}_7\text{)} &: [B \text{ exp } \perp] \wedge [B \text{ rcv } \text{Init}_i] \Rightarrow \exists j : \diamond [B \text{ exp } i|j] \\
\text{AXM (B}_8\text{)} &: [B \text{ exp } i|j] \wedge \square \neg [B_1 \text{ rcv } x_{i|j}] \Rightarrow \square \diamond [B_2 \text{ snd } i|j] \\
\text{AXM (B}_9\text{)} &: [B \text{ rcv } \text{Reinit}_{i|j}] \wedge [B \text{ exp } k|j] \Rightarrow \diamond [B \text{ exp } i|j]
\end{aligned}$$

Further, we have to prove the following property (Connect):

$$\text{THM (Connect)} : [A \text{ snd } \text{Init}_i] \wedge \diamond [B \text{ exp } i|j] \Rightarrow \diamond ([A \text{ seq } i|j] \wedge [B \text{ exp } i|j])$$

The theorem (Connect) is a property according to which the behavior of the agent A depends on some events that will happen on the receiver side B . Following our principles of development, we have to refine this property to individual properties.

Proof of (Connect):

$$\begin{aligned}
& [A \text{ snd } \text{Init}_i] \wedge \diamond [B \text{ exp } i|j] \\
\Rightarrow & \diamond [B \text{ exp } i|j] \wedge \diamond [A \text{ rcv } i|j] && \text{[by TP}_2\text{]} \\
\Rightarrow & \diamond [B \text{ exp } i|j] \wedge \diamond [A \text{ seq } i|j] && \text{[by A}_6 \text{ below]} \\
\Rightarrow & \diamond ([B \text{ exp } i|j] \text{ until } [B \text{ rcv } x_{i|j}]) \wedge \diamond ([A \text{ seq } i|j] \text{ until } [A_2 \text{ rcv } i]) && \text{[by B}_5 \text{ and A}_3\text{]} \\
\Rightarrow & \diamond ([B \text{ exp } i|j] \text{ until } [B \text{ rcv } x_{i|j}]) \wedge \diamond ([A \text{ seq } i|j] \text{ until } [B \text{ rcv } x_{i|j}]) && \text{[by TL]}
\end{aligned}$$

$$\begin{aligned} &\Rightarrow \diamond (([B \text{ exp } i|j] \wedge [A \text{ seq } i|j]) \text{ until } [B \text{ rcv } x_{i|j}]) && \text{[by temporal logic]} \\ &\quad \{\text{since } [B \text{ rcv } x_{i|j}] \Rightarrow \blacklozenge [A \text{ seq } i|j]\} \\ &\Rightarrow \diamond ([B \text{ exp } i|j] \wedge [A \text{ seq } i]) \\ &q.e.d. \end{aligned}$$

Consequently, we have to enrich the protocol specification with the following axiom stating that the agent A adopts the sequence numbers of the acknowledgement of its connection request.

ENRICH TP-Implementation BY

PROPERTIES

$$\text{AXM (A}_6\text{)} : [A \text{ seq } Init_i] \wedge \diamond [A_2 \text{ rcv } i|j] \Rightarrow \diamond [A \text{ seq } i|j]$$

To finish, we have to establish the following property:

$$\text{THM (Unique)} : [A \text{ snd } Init_i] \wedge \blacklozenge ([A \text{ snd } x_k] \vee [A \text{ snd } x_{k|j}]) \Rightarrow \square \neg [A \text{ snd } x_{k|j}]$$

Actually, the property above concerns only the behavior of the agent A . However, its realization depends strongly on the behavior of B : if the agent B always chooses a second sequence number which it has never chosen before, then the above property is satisfied without adding any property referring to the behavior of A . Therefore, we add the following axiom to the specification of the agent B :

ENRICH TP-Implementation BY

PROPERTIES

$$\text{AXM (B}_{10}\text{)} : [B_2 \text{ snd } i|j] \wedge \diamond [B_2 \text{ snd } k|1] \wedge i \neq k \Rightarrow j \neq 1$$

This finishes the proof that agreement between the agent A and B will eventually be reached. As a consequence, the data-transfer protocol has to be slightly modified such that just after establishing a connection both agents have to deal with pairs of sequence numbers instead of one number. This again is an internal arrangement, because the agent A adds a second sequence number to the messages received on the stream S and, on the other side, the agent B deletes this number before delivering a message on the stream R .

Ensuring Implementability

As shown in the previous section, sequence numbers are relevant in the arrangement of messages in the right order. Based on the network properties (Net_1) and (Net_2), the proper functioning of the protocol can only be assured if the set of sequence numbers is infinite. However, as soon as we decide to implement the protocol, we must consider a finite set of sequence numbers, because we have to deal with a bounded memory.

Finite Sequence Numbers: In our development, we have considered a network in which messages may be delayed indefinitely. In practice, however, a message may not be stored in the network beyond a maximum lifetime L . That is, if a message is introduced into the network at a time T , then if it has not been delivered to its destination by the time $(T + L)$ it is lost and will never be delivered. In order to achieve the proper functioning of the

protocol with a finite set of sequence numbers, we have to take this lifetime property into account. Therefore, we add a real-time property to the specification of the network, stating that a packet cannot reside longer than L time units in the network. Obviously, this requires quantitative reasoning.

In order to be able to express such a property, we introduce a *metric* d on the set of colors (abstract time-stamps). We denote by $d(x, y)$ the time interval between the transmissions of x and y . We enrich the specification `Network` with the following property:

ENRICH `Network` BY

PROPERTIES

$$\text{AXM } (\text{Net}_3) : [\text{Net rcv } x_i] \wedge \diamond [\text{Net snd } \tilde{x}_i] \Rightarrow d(x, \tilde{x}) < L$$

This property says that the time interval between introducing a message into the network and delivering it cannot exceed L time units. According to this extension, some modifications in the `TCP-implementation` are needed in order to adapt it to the new property of the underlying net. However, the properties of the agent A are not affected by this modification. All we need to do is modify the axiom B_{10} :

ENRICH `TP-Implementation` BY

PROPERTIES

$$\text{AXM } (B'_{10}) : [B_2 \text{ snd } i|j] \wedge \diamond [B_2 \text{ snd } k|\text{pred}(j)] \Rightarrow d(i, k) > L$$

Assuming that sequence numbers are also colored, this property guarantees that the agent B cannot select the same sequence number in a time interval which does not exceed L time units. Note that from now on the agents A and B use a cyclic set of sequence numbers. Consequently, we add the following properties to the specification of sequence numbers:

ENRICH `Seq-Number`

SIGNATURE

FUN `Max-Num`, `Min-Num` : `Seq-Number`

PROPERTIES

$$\text{AXM } \text{succ}(\text{Max-Num}) = \text{Min-Num}$$

$$\text{AXM } \text{pred}(\text{Min-Num}) = \text{Max-Num}$$

Because the safety property (TP_1) and the liveness property (TP_2) do not depend on sequence numbers, their proofs are affected by these modifications, whereas the property of preserving the order of messages (TP_3) must be verified again.

Keeping the Order of Messages: In Section 6.3.4, we proved that the whole system can preserve the order of messages using sequence numbers. This result is based on the assumption that we have an infinite set of numbers at our disposal. Because we now have only a finite set of numbers, the property of preserving the order of messages (TP_3) has to be verified again. So we have to modify the behavior of the agents A and B in such a way that the order of messages will be preserved, even if we use a finite set of sequence numbers.

Proof of TP₃:

Assuming that $[R \mathbf{xmt} x_i] \wedge \tilde{\mathcal{O}}_R [R \mathbf{xmt} y_j]$, we distinguish by axiom (B₅) and (B₉) two cases:

- **Case1:** There was no connection release between the acceptance of x_i and y_j by the agent B . So in this case j is the successor of i and we deduce:

$$\begin{aligned}
& \blacklozenge ([B \mathbf{rcv} x_i] \wedge [B \mathbf{exp} suc(i)] \mathbf{until} [B \mathbf{rcv} y_{suc(i)}]) \\
\Rightarrow & \blacklozenge ([A \mathbf{snd} \tilde{x}_i] \wedge \blacklozenge [A \mathbf{snd} \tilde{y}_{suc(i)}] \wedge [A_1 \mathbf{snd} \tilde{x}_i] \mathbf{until} [A_1 \mathbf{snd} \tilde{y}_{suc(i)}]) \quad [\text{by Thm}_{9,9}] \\
\Rightarrow & \blacklozenge ([A \mathbf{snd} \tilde{x}_i] \wedge \tilde{\mathcal{O}}_U [A \mathbf{snd} \tilde{y}_{suc(i)}]) \quad [\text{by TL}] \\
\Rightarrow & \blacklozenge ([S \mathbf{xmt} \tilde{x}_i] \wedge \tilde{\mathcal{O}}_S [S \mathbf{xmt} \tilde{y}_{suc(i)}]) \quad [\text{by axiom A}_1] \\
\Rightarrow & \blacklozenge ([S \mathbf{xmt} \tilde{x}_i] \wedge \tilde{\mathcal{O}}_S [S \mathbf{xmt} \tilde{y}_{suc(i)}])
\end{aligned}$$

- **Case2:** A connection release and establishment occurred between the receipt of x_i and y_j by B . We obtain the following situation:

$$[B \mathbf{rcv} x_i] \wedge \blacklozenge ([B \mathbf{exp} \perp] \wedge [B \mathbf{rcv} Init_{j[k]}] \wedge [B \mathbf{rcv} y_{j[k]}])$$

The proof proceeds similarly to case1 above.

The proof of (TP₃) is based on the properties (Thm_a) and (Thm_b), in which both agents A and B occur. Following our development principles, these properties should be replaced by individual properties. The first theorem states that if the agent B receives two messages with consecutive sequence numbers, then they have been once sent by A in the same order as they reached B .

$$\text{THM (Thm}_a\text{)} : ([B \mathbf{rcv} x_i] \wedge \blacklozenge [B \mathbf{rcv} y_{suc(i)}]) \Rightarrow \blacklozenge ([A \mathbf{snd} \tilde{x}_i] \wedge \blacklozenge [A \mathbf{snd} \tilde{y}_{suc(i)}])$$

Proof of Thm_a:

We perform this proof by contradiction. We start from the two assumptions:

$$\begin{aligned}
(*) & \quad ([B \mathbf{rcv} x_i] \wedge \blacklozenge [B \mathbf{rcv} y_{suc(i)}]) \\
(**) & \quad \neg \blacklozenge ([A \mathbf{snd} \tilde{x}_i] \wedge \blacklozenge [A \mathbf{snd} \tilde{y}_{suc(i)}])
\end{aligned}$$

Then we deduce

$$\begin{aligned}
& \Rightarrow \blacklozenge [A \mathbf{snd} \tilde{x}_i] \wedge \blacklozenge [A \mathbf{snd} \tilde{y}_{suc(i)}] \quad [\text{by } (*) \text{ and Net}_2] \\
& \Rightarrow \blacklozenge ([A \mathbf{snd} \tilde{y}_{suc(i)}] \wedge \blacklozenge [A \mathbf{snd} \tilde{x}_i] \wedge \neg [A \mathbf{snd} \tilde{y}_{suc(i)}] \mathbf{until} [B \mathbf{rcv} x_i]) \quad [\text{by } (**) \text{ and TL}] \\
& \Rightarrow d(\tilde{y}, \tilde{x}) > L \wedge d(y, \tilde{y}) = d(\tilde{y}, \tilde{x}) + d(\tilde{x}, y) \quad [\text{by axiom A}_7] \\
& \Rightarrow d(y, \tilde{y}) > L
\end{aligned}$$

$$\{\text{by Net}_3 \text{ we have } d(y, \tilde{y}) < L\}$$

contradiction!

q.e.d.

This leads to the addition of the following property to the specification of the behavior of A . This property states that whenever the agent A sends a message with sequence number i and later a message with the predecessor of i , then the time interval between their transmissions must be greater than L time units. Note that this reasoning is based on the property (S₀) which now means that messages on the stream S are indexed with cyclic sequence numbers. Moreover, we assume that the set of sequence numbers includes at least two elements.

ENRICH TCP-Implementation BY

PROPERTIES

$$\text{AXM (A}_7\text{)} : [A \text{ snd } x_i] \wedge \diamond [A \text{ snd } y_{\text{pred}(i)}] \Rightarrow d(x, y) > L$$

The following theorem claims that if the agent B remains connected between the acceptances of two messages with consecutive sequence numbers, then the agent A was also connected during the transmission of these messages.

THM (Thm_b) :

$$([B \text{ rcv } x_i] \wedge [B \text{ exp } \text{suc}(i)] \text{ until } [B \text{ rcv } y_{\text{suc}(i)}]) \Rightarrow \blacklozenge([A_1 \text{ snd } \tilde{x}_i] \text{ until } [A_1 \text{ snd } \tilde{y}_{\text{suc}(i)}])$$

Proof of Thm_b:

We perform proof of (Thm_b) by contradiction. We start from the two assumptions:

$$(*) \quad \blacklozenge([B \text{ rcv } x_i] \wedge [B \text{ exp } \text{suc}(i)] \text{ until } [B \text{ rcv } y_{\text{suc}(i)}])$$

$$(**) \quad \neg([A_1 \text{ snd } \tilde{x}_i] \text{ until } [A_1 \text{ snd } \tilde{y}_{\text{suc}(i)}])$$

Then we deduce

$$\begin{aligned} & \blacklozenge([A \text{ snd } \tilde{x}_i] \wedge [A \text{ snd } \tilde{y}_{\text{suc}(i)}]) && \text{[by } (*), \text{ Thm}_a \text{]} \\ \Rightarrow & \exists z_j : \blacklozenge([A \text{ snd } \tilde{x}_i] \wedge \diamond([A \text{ snd } z_j] \wedge \diamond([A \text{ snd } \tilde{y}_{\text{suc}(i)}]))) && \text{[by } (**), \text{ TL]} \\ \Rightarrow & \blacklozenge([A \text{ snd } \tilde{x}_i] \wedge \diamond([A \text{ snd } z_j] \text{ until } [A_2 \text{ rcv } j])) && \text{[by axiom A}_4\text{]} \\ \Rightarrow & \blacklozenge([A \text{ snd } \tilde{x}_i] \wedge \diamond([A \text{ snd } z_j] \text{ until } ([B \text{ exp } j] \wedge [B \text{ rcv } z_j]))) && \text{[by TL]} \\ \Rightarrow & \blacklozenge([A \text{ snd } \tilde{x}_i] \wedge \diamond([A \text{ snd } z_j] \text{ until } ([B \text{ exp } j] \wedge [B \text{ rcv } z_j]))) && \text{[by } (*), \text{ TL]} \\ \Rightarrow & \blacklozenge([A \text{ snd } \tilde{x}_i] \wedge \diamond([A \text{ snd } z_j] \text{ until } [B \text{ rcv } y_{\text{suc}(i)}])) && \text{[by } (*), \text{ TL]} \\ \Rightarrow & \blacklozenge(\blacklozenge([A \text{ snd } y_{\text{suc}(i)}]) \wedge [A \text{ snd } \tilde{x}_i] \wedge \neg[A \text{ snd } \tilde{y}_{\text{suc}(i)}] \text{ until } [B \text{ rcv } y_{\text{suc}(i)}]) \\ \Rightarrow & d(\tilde{y}, \tilde{x}) > L \wedge d(y, \tilde{y}) = d(\tilde{y}, \tilde{x}) + d(\tilde{x}, y) && \text{[by axiom A}_7\text{]} \\ \Rightarrow & d(y, \tilde{y}) > L \end{aligned}$$

$$\{\text{by Net}_3 \text{ we have } d(y, \tilde{y}) < L\}$$

contradiction!

q.e.d.

The proof of case2 (TP₃) proceeds similarly to the proof of case1. However, we have to add to the specification of the agent A a similar axiom to (A₇) dealing with “Init”-messages. Thus, we enrich the TP-implementation by the following axiom, which states that if the agent A sends a message with the sequence number i and later it sends an “Init”-message with this number, then the time interval between these two transmissions should be greater than L time units.

ENRICH TCP-Implementation BY

PROPERTIES

$$\text{AXM (A}_8\text{)} : [A \text{ snd } x_i] \wedge \diamond [A \text{ snd } \text{Init}_i] \Rightarrow d(x, \text{Init}) > L$$

This completes the specification of the transport protocol.

Chapter 7

Conclusions and Related Works

We believe that the approach presented in this work is a valuable contribution to the formal development of communication protocols. It provides a formalism based on a combination of algebraic specifications and temporal logic. This combination results in a formal language for describing data aspects and behavioral aspects in a unified framework. Many attempts have been made to combine process description and data-type specification techniques in developing concurrent and distributed systems. One of the most important examples is the specification language LOTOS [Inf87], which is based on the combination of CCS and the algebraic specification language ACT ONE [EM85]. LOTOS has been applied to the specification of protocol standards. From a practical point of view, the specification language presented here can be applied to verify properties of protocol standards written in LOTOS, because it can describe protocols at a more abstract level. Another interesting approach is the introduction of temporal logic in Z notation (see e.g. [DS89] and [Fid92]). Although temporal logic does not increase the expressive power of the notation, the unification of Z and temporal logic provides an elegant way for reasoning about concurrent and distributed systems [DS89]. Just as interesting is the unification of algebraic specifications and Petri nets, as suggested in [Vau86]. In this combination, data elements are used instead of tokens, so that the size of classical Petri nets can be reduced and they become more manageable [EPR93].

One of the main objectives of our work is the extension of temporal logic, by indexing temporal operators, in order to achieve modularity. The obtained logic gives rise to a formalism that supports hierarchical development of communication protocols in a modular style. Furthermore, a justification rule for composing specifications and a proof rule for verifying refinements are given. In our approach, the composition of temporal specifications is made by conjunction, and refinements are verified by logical consequences. Moreover, refinements meet the requirements of [ZCdR92], which state that transformations should be vertically as well as horizontally composable. This was shown in Section 5.3. Vertical and horizontal composability enable us to develop system components independently.

Many attempts have been made to achieve compositionality and modularity of temporal logic. One important approach uses the notion of “stuttering”, which was introduced by Lamport [Lam83c]. In this approach, the behavior of a system is considered to be closed under stuttering, i.e. finite repetition of states in sequences. In [AL89] Abadi and Lamport proved that composition of relay/guarantee temporal specifications is made by conjunction

if behaviors are closed under stuttering. The idea of stuttering has also been applied to the TLA (temporal logic of actions) proposed in [Lam94]. In contrast to other temporal logics, action formulas refer to a pair of states instead of to a single state. Such an action formula describes the action that transforms the first states into the second in one step. In order to achieve compositionality, action formulas are indexed with program variables, allowing the modification of these variables to be delayed. In this context, it is necessary to add to the specification an assertion which ensures that the action should not be infinitely delayed. Composition of specifications is then made by conjunction, and refinement by implication. In the approach presented in [BKP84], modularity is achieved for temporal logic in that propositions which distinguish between transitions effected by a module and those performed by the environment are introduced. It should be noted that our proof rules are simpler and easier to apply than comparable proof rules based on state-transition models, e.g. [BK83, MP83, BKP84]. In our opinion, indexing of temporal operators is an elegant way to achieve modularity and compositionality of temporal logic.

Apart from the extension made to achieve modularity, we had to extend our temporal logic in order to be able to express desired properties of communication protocols. It has been shown that a large class of protocol properties cannot be expressed in linear temporal logic. This inexpressiveness problem is caused by the inability of temporal logic to uniquely identify messages on an infinite stream. Most attempts, e.g. [Koy92] and [Pnu92], overcome this problem by assuming that messages transmitted on channels are pairwise distinct. In other approaches, such as [NGO86] and [Hai82], unique identification of messages is made implicitly by reasoning about histories instead of detached messages. In our approach, we have extended our semantical model by introducing colors, which serve as unique identifiers for messages on streams. In the course of the investigation, it became clear that we need a hierarchy of equivalence classes of colors in order to be able to describe both reliable and unreliable systems. This extension does actually increase the expressive power of the logic, while not complicating the description of protocol properties. It should be stressed that colors were considered only at the conceptual level, and they do not influence in any fashion the proper functioning of agents. In fact, their introduction at the syntactical level depends on whether agents may test colors or not.

Chapter 6 served to demonstrate that the methodology can be applied to problems that approach real-world complexity. One important point is that a protocol specification is developed step by step, starting from a requirements specification which describes the services provided by the protocol. Another point is that in the final product, which is a refinement of the requirements specification, the properties of the individual protocol entities are stated in isolation: an agent occurs only in the formulas that concern its individual behavior. This is very important in the context of distributed systems, because the protocol entities will be implemented in separate locations. Although the Alternating-Bit protocol, the CSMA/CD protocol, and the Three-Way Handshake are relatively simple, we feel that the methodology could be applied in the development of more complex protocols.

A limitation of our formalism is its unsuitability for reasoning about real-time properties of protocols, such as time-out and performance. The temporal operators used here are suitable for reasoning about qualitative properties, but cannot deal with quantitative temporal requirements. However, this does not cause any difficulties, because our methodology can also be applied when using a formalism that can express real-time properties instead of linear

temporal logic. Actually, such a formalism is no more than a slightly extended version of the temporal logic used here. Among many candidates, we cite the TPTL introduced in [AH89], the real-time temporal logic defined in [Ost87], and the metric temporal logic proposed in [Koy92].

Bibliography

- [Abr85] J. R. Abrial. Programming as a mathematical exercise. In C.A.R. Hoare, editor, *Mathematical Logic and Programming Languages*. Prentice-Hall International, 1985.
- [AGR88] E. Astesiano, A. Giovini, and G. Reggio. Date in a concurrent environment. In *Proc. Concurrency '88 Conference*, volume 335 of *Lecture Notes in Computer Science*, Hamburg, 1988. Springer.
- [AH89] R. Alur and T.A. Henzinger. A really temporal logic. In *Proceedings of the Thirtieth Symposium on the Foundation of Computer Science*, pages 164–169, 1989.
- [AL89] M. Abadi and L. Lamport. Composing specifications. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness*, volume 430 of *Lecture Notes in Computer Science*, pages 1–41, Berlin, 1989. Springer.
- [Ame85] American National Standard: **ISO DIS 8802/3**. *Local Area Networks: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, 1985. ANSI/IEEE Standard 802.3.
- [AS85] B. Alpern and F.B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, 1985.
- [BAPM81] M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. In *Proceedings of the Eight ACM Symposium on Principles of Programming Languages*, pages 164–176, Williamsburg, 1981.
- [BFG⁺93] M. Broy, C. Facci, R. Grosu, H. Hußmann R. Hettler, D. Nazareth, F. Regensburger, and K. Stølen. The requirement and design specification language SPECTRUM - an informal introduction, version 1.0. Technical report, Technische Universität München, 1993.
- [BJ87] D. Bjørner and C. B. Jones. *The vienne development method: the meta-language*, volume 280 of *Lecture Notes in Computer Science*. Springer, Berlin, 1987.
- [BK83] H. Barringer and R. Kuiper. Towards the hierarchical temporal logic specification of concurrent systems. In G. Goos and J. Hartmanis, editors, *The Analysis of Concurrent Systems*, volume 207 of *Lecture Notes in Computer Science*, pages 157–183. Springer, Berlin, 1983.

- [BKP84] H. Barringer, R. Kuiper, and A. Pnueli. Now you may compose temporal specifications. In *Proceeding of the 16th ACM Symposium on Theory of Computing*, pages 51–63, 1984.
- [Bro88] M. Broy. Requirement and design specification for distributed systems. In F.H. Vogt, editor, *Concurrency*, volume 335 of *Lecture Notes in Computer Science*, pages 33–62, Berlin, 1988. Springer.
- [BS80] G. Bochmann and C. Sunshine. Formal methods in communication protocol design. *IEEE Transaction on Communication*, 28(4):624–631, April 1980.
- [BSW69] K.A. Bartlett, R.A. Scantleburg, and P.T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communication of the ACM*, 12(5):260–261, May 1969.
- [Bur74] R.M. Burstall. Program proving as hand simulation with little induction. In *Proceedings of IFIP Congress*, pages 308–312, Stockholm, 1974. Amsterdam: North-Holland.
- [Bur82] J. Burgess. Axioms for tense logic. ‘since’ and ‘until’. *Notre Dame J. of Formal Logic*, 23:367–374, 1982.
- [CHR91] Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letter*, 40(5):269–276, 1991.
- [DFG⁺94] Klaus Didrich, Andreas Fett, Carola Gerke, Wolfgang Grieskamp, and Peter Pepper. OPAL: Design and Implementation of an Algebraic Programming Language. In Jürg Gutknecht, editor, *Programming Languages and System Architectures, International Conference, Zurich, Switzerland, March 1994*, LNCS 782, pages 228–244. Springer, 1994.
- [DS89] R. Duke and G. Smith. Temporal logic and Z specifications. *Australian Computer Journal*, 21(2):62–69, May 1989.
- [EFH83] H. Ehrig, W. Fey, and H. Hansen. ACT ONE: An algebraic specification language with two levels of semantics. Technical Report 83-03, Dept. of Computer Science, Technical University of Berlin, 1983.
- [EH86] A.E. Emerson and J.Y. Halpern. “sometimes” and “not never” revisited on branching versus linear time temporal logic. *Journal of the ACM*, 33:151–178, 1986.
- [EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1*. EATCS Monographs on Theoretical Computer Science. Springer, Berlin, 1985.
- [EPR93] H. Ehrig, J. Padberg, and L. Ribeiro. Algebraic high-level nets. In *Proc. of the ADT-COMPASS Workshop’92*, Lecture Notes in Computer Science. Springer, 1993.
- [Fid92] C. J. Fidge. Specification and verification of real-time behavior using Z and RTL. In *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 571 of *Lecture Notes in Computer Science*, New York, 1992. Springer.

- [Flo67] R. W. Floyd. Assigning meaning to programs. *Mathematical Aspects of Computer Science, XIX American Mathematical Society*, pages 19–32, 1967.
- [GB84] J. Goguen and R.M. Burstall. Introducing institutions. In *Proc. Logics of Programming Workshop, Carnegie-Mellon*, volume 164 of *Lecture Notes in Computer Science*, pages 221–256, Berlin, 1984. Springer.
- [GB92] J.A. Goguen and R.M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of ACM*, 39(1):95–146, January 1992.
- [GTW78] J.A. Goguen, J.W. Thatcher, and E.G. Wagner. An initial algebra approach to the specification, correctness, and implementation of abstract data types. In *Data Structuring*, volume 4 of *Current Trends in Programming Methodology*, pages 80–144. Prentice-Hall, Englewood Cliffs, NJ, 1978.
- [Hai82] B. T. Hailpern. *Verifying Concurrent Processes Using Temporal Logic*, volume 129 of *Lecture Notes in Computer Science*. Springer, Berlin, 1982.
- [HC68] G.E. Hughes and M.J. Cresswell. *An Introduction to Modal Logic*. Methuen and CO LTD, London, 1968.
- [Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *CACM*, 12:105–120, 1969.
- [Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.
- [Inf87] Information Processing Systems - Open Systems Interconnection **ISO DIS 8807**. *LOTOS - A formal description technique based on temporal ordering of observational behaviour*, Jul. 1987. (ISO/TC 97/SC 21N).
- [Jma94] M. Jmaiel. Specifying communication protocols with temporal logic. Technical Report 94/16, Technische Universität Berlin, Fachbereich Informatik, 1994.
- [Jma95a] M. Jmaiel. An algebraic-temporal specification of a CSMA/CD-protocol. In *Proceedings of the IFIP WG 6.1 Fifteenth International Symposium on Protocol Specification, Testing and Verification*, Warsaw, Poland, June 1995. Chapman and Hall.
- [Jma95b] M. Jmaiel. Development of communication protocols by composing and refining temporal specifications. In *Proceedings of the 4th Software Quality Conference*. University of Abertay Dundee, Scotland, UK, July 1995.
- [JP94] M. Jmaiel and P. Pepper. Development of communication protocols using algebraic and temporal specifications. In *Proceedings of the International Workshop on Advanced Software Technology, Shanghai, Sept. 15-16 1994.*, Shanghai, 1994. Jiao Tong University.
- [Kam68] H.W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, ULCA, Los Angeles, 1968.

- [KHMP94] A. Kapur, T. A. Henzinger, Z. Manna, and A. Pnueli. Proving safety properties of hybrid systems. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *Lecture Notes in Computer Science*, pages 431–454. Springer, 1994.
- [Koy92] R. Koymans. *Specifying Message Passing and Time-Critical Systems with Temporal Logic*, volume 651 of *Lecture Notes in Computer Science*. Springer, Berlin, 1992.
- [Kri63] S.A. Kripke. Semantical analysis of modal logic I, normal propositional calculi. *Zeitschrift für math. Logik und Grundlagen der Mathematik*, 9:67–96, 1963.
- [Krö76] F. Kröger. Logical rules of natural reasoning about programs. In *ICALP 76*, pages 87–98, Edinburgh, 1976. Edinburgh University Press.
- [Krö78] F. Kröger. A uniform logical basis for the description, specification and verification of programs. In *Proceeding of IFIP Workshop: Formal Description of Programming Concepts*, pages 441–457, Canada, 1978. Amsterdam: North-Holland.
- [Krö87] F. Kröger. *Temporal Logic of Programs*. EATCS Monographs on Theoretical Computer Science. Springer, Berlin, 1987.
- [Lam77] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Transaction on Software Engineering SE-3*, 2:125–143, 1977.
- [Lam80] L. Lamport. ‘sometimes’ is sometimes ‘not never’. In *Proceeding of the 7th ACM Symposium on Principles of Programming Languages*, pages 174–185, 1980.
- [Lam83a] L. Lamport. Specifying concurrent program modules. *ACM Transactions on Programming Languages and Systems*, 5(2):190–222, 1983.
- [Lam83b] L. Lamport. Stl/serc problems. In *Proceedings of a Tutorial and Workshop, Cambridge University, September 1983*, volume 207 of *Lecture Notes in Computer Science*, Berlin, 1983. Springer.
- [Lam83c] L. Lamport. What good is temporal logic. In *Proceedings of IFIP Congress, Paris 83*, pages 657–668, Amsterdam, 1983. North-Holland.
- [Lam85] L. Lamport. Logical foundation. In M. Paul and H.J. Siegart, editors, *Distributed Systems - Methods and Tools for Specification*, volume 190 of *Lecture Notes in Computer Science*. Springer, 1985.
- [Lam94] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.
- [LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Proc. of the Workshop on Logics of Programs 85*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218, Berlin, 1985. Springer.
- [LT87] K. Lodaya and P.S. Thiagarajan. A modal logic for a subclass of event structures. In *Proceedings of the 14th ICALP*, volume 267 of *Lecture Notes in Computer Science*, pages 290–303. Springer, 1987.

- [Mil80] R. Milner. *A Calculus for Communication Systems*, volume 90 of *Lecture Notes in Computer Science*. Springer, Berlin, 1980.
- [MM84] B. Moszkowski and Z. Manna. Reasoning in interval temporal logic. In *Proceedings of AMC/NSF/ONR Workshop on Logics of Programs*, volume 164 of *Lecture Notes in Computer Science*, pages 371–383. Springer, 1984.
- [Möl85] B. Möller. On the algebraic specification of infinite objects, ordered and continuous models of algebraic types. *Acta Informatica*, 22:537–578, 1985.
- [Mos85] B. Moszkowski. A temporal logic for multi-level reasoning about hardware. *IEEE Computer*, 18(2):10–19, 1985.
- [MP83] Z. Manna and A. Pnueli. How to cook a temporal proof system for your pet language. In *Proc. of the 10th ACM Symposium on the Principles of Programming Languages*, pages 141–154, 1983.
- [MP89] Z. Manna and A. Pnueli. The anchored version of the temporal framework. In J.W. de Bakker, W.P. de Reover, and G. Rozenberg, editors, *Linear time, Branching time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 201–284. Springer, 1989.
- [MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer, 1991.
- [MP93] Z. Manna and A. Pnueli. Verifying hybrid systems. In R.L. Grossman, A. Nerode, A. Ravn, and A. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science, pages 371–356. Springer, 1993.
- [NGO86] V. Nguyen, D. Gries, and S. Owicki. A model and temporal proof system for networks of processes. *Distributed Computing*, 1:7–25, 1986.
- [Ost87] J.S. Ostroff. *Real-Time Computer Control of Discrete Event Systems Modelled by Extended State Machines: A Temporal Logic Approach*. PhD thesis, Department of Electrical Engineering, University of Toronto, 1987.
- [Par91] H. Partsch. *Requirements Engineering*. Handbuch der Informatik, Oldenburg, 1991.
- [Pet62] C. A. Petri. Fundamentals of a theory of asynchronous information flow. In *Proc. of the IFIP Congress 1962, Munich*, pages 386–390, Amsterdam, 1962. North Holland Publishing Company.
- [Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the Eighteenth Symposium on the Foundations of Computer Science*, pages 46–57, 1977.
- [Pnu92] A. Pnueli. System specification and refinement in temporal logic. In *Lecture Notes in Computer Science*, volume 652, Berlin, 1992. Springer.
- [Pri57] A.N. Prior. Time and modality. *Oxford, University Press*, 1957.
- [Pri67] A.N. Prior. Past, present and future. *Oxford, University Press*, 1967.

- [PW84] S. Pinter and P. Wolper. A temporal logic for reasoning about partially ordered computations. In *Proceedings of the Third ACM Symposium on Principles of Distributed Computing*, pages 45–60, Vancouver, Canada, 1984.
- [PW94] P. Pepper and M. Wirsing. KORSO: a Methodology for the Development of Correct Software. In M. Broy and S. Jähnichen, editors, *KORSO, Correct Software by Formal Methods*, Lecture Notes in Computer Science. Springer, 1994.
- [Rei88] W. Reisig. Temporal logic and causality in concurrent systems. In F.H. Vogt, editor, *Proc. of Concurrency 88*, volume 335 of *Lecture Notes in Computer Science*, pages 603–627, Berlin, 1988. Springer.
- [RU71] N. Rescher and A. Urquhart. *Temporal Logic*. Springer, 1971.
- [SCFG82] A.P. Sistla, E.M. Clarke, N. Francez, and Y. Gurevich. Can message buffers be characterized in linear temporal logic? In *Proceedings of the First ACM Symposium on Principles of Distributed Computing*, pages 148–156, 1982.
- [SCFM84] A.P. Sistla, E.M. Clarke, N. Francez, and A.R. Meyer. Can message buffers be axiomatized in linear temporal logic? *Information and Control*, 63:88–112, 1984.
- [SD78] C.A. Sunshine and Y.K. Dalal. Connection management in transport protocols. *Computer Networks*, 2:454–473, 1978.
- [Sis85] A.P. Sistla. On characterization of safety and liveness properties in temporal logic. In *Proceedings of the 4th Annual ACM Symposium on Principles of Distributed Computing*, pages 39–48, 1985.
- [SMSV83] R.L. Schwartz, P.M. Melliar-Smith, and F. Vogt. Interval logic: A higher level temporal logic for protocol specification. In *Proceedings of the IFIP WD6.1 Third International workshop on Protocol Specification, Testing and Verification*, pages 3–18, 1983.
- [Ste76] N.V. Stenning. A data transfer protocol. *Computer Networks*, 1:99–110, 1976.
- [Tom75] R.S. Tomlinson. Selecting sequence numbers. In *Proceedings of the ACM SIGCOM/SIGOPS Interprocess Communications Workshop*, pages 11–23, Santa Monica, California, March 1975.
- [Tur49] A. Turing. On checking a large routine. In *Report of a conference on high-speed automatic calculating machines*, University Mathematical Laboratory, Cambridge, 1949.
- [Vau86] J. Vautherin. Parallel specification with colored Petri nets and algebraic data types. In *Proceedings of the 7th European Workshop on Application and Theory of Petri nets*, pages 5–23, Oxford, England, July 1986.
- [Wir90] M. Wirsing. Algebraic specification. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 13, pages 675–788. Elsevier Science Publishers B.V., 1990.

- [Wol86] P. Wolper. Expressing interesting properties of programs in propositional temporal logic. In *Proceedings of the Thirteenth ACM Symposium on the Principles of Programming Languages*, pages 184–193, 1986.
- [ZCdR92] J. Zwiers, J. Coenen, and W.-P. de Roever. A note on compositional refinement. In *Proc. of the 5-th Refinement Workshop*, 1992.
- [Zwi89] J. Zwiers. *Compositionality, Concurrency and Partial Correctness: Proof Theories for Networks of Processes, and their Connection*, volume 321 of *Lecture Notes in Computer Science*. Springer, Berlin, 1989.