Symbolic Bisimulation Checking and Decomposition of Real-Time Service Specifications

Akio Nakata

January 1997

Symbolic Bisimulation Checking and Decomposition of Real-Time Service Specifications

Akio Nakata

January 1997

A Dissertation submitted to the Faculty of the Engineering Science
of Osaka University for the Degree of Doctor of Philosophy in Engineering

Abstract

This thesis summarizes the work of the author during bachelor/master/doctor student of Osaka University on formal design of reliable real-time distributed systems. In this thesis, we present a formal specification language for real-time distributed systems, a verification method of their equivalence, and a decomposition method of formally specified real-time services into a set of specifications for distributed nodes.

In the first part of this thesis, we propose a language LOTOS/T, which is an enhancement of LOTOS, an international standard formal description language for distributed systems and communication protocols. LOTOS/T enables us to describe not only various behaviour of systems such as sequential compositions, choices, parallel executions, and interruptions, but also time constraints of each action in formulas of 1st-order predicate logic on time domain. The user has only to describe when each action must be executed. The user can also specify an assignment of executed time of each action into some variable. which can be referred in the time constraints of succeeding actions. Use of equality ($=$) and inequality ($\leq$) as the time constraints enables us to describe intervals, timeout and delay easily. We define the syntax and semantics of LOTOS/T formally. The semantic model of LOTOS/T is the Labelled Transition System (LTS). We give the inference rules for constructing the LTSs mechanically from given LOTOS/T expressions. We also define timed/untimed bisimulation equivalence for real-time systems. Timed bisimulation equivalence is a kind of bisimulation equivalence where timing of each action is also equal, whereas untimed bisimulation equivalence ignores the timing. It is easily proved that they are decidable if the corresponding LTSs are finite-state.

Verification of timed bisimulation equivalence is generally difficult due to the state explosion caused by concrete time values. Therefore, in the second part of this thesis, we propose a verification method of timed bisimulation equivalence where its verification cost is independent of the concrete time values described in the specifications. We first propose a new model of real-time systems, Alternating Timed Symbolic Labelled Transition System(A-TSLTS). In an A-TSLTS, each state has some parameter variables, whose values determine its behaviour. Each transition in an A-TSLTS has a guard predicate. The transition is executable if and only if its guard predicate is true under the specified parameter values. For a given state-pair of a finite A-TSLTS, the proposed method produces the weakest condition for the parameter values to make the state-pair be timed/untimed bisimulation equivalent. A method to convert LOTOS/T expressions into A-TSLTSs is also given.

In the third part of this thesis, we propose a method to decompose specifications of real-time services written in LOTOS/T into a set of specifications of distributed nodes automatically. Here we assume that there is a reliable communication channel between any two nodes and the maximum communication delay for each channel is bounded by a constant. Moreover we assume service specifications have no deadlocks. Under our simulation policy, a specification S' is derived from a given service specification S and a given maximum communication delay of each channel. In S', some time-constraints are added in order to make sure synchronization messages between nodes can reach in time. Our method firstly check if S and S' can carry out the same behaviour, i.e., if S and S' are untimed bisimulation equivalent. If they are equivalent, then we derive a correct protocol specification for simulating S from S' automatically.

List of Major Publications

1. Akio Nakata, Teruo Higashino, and Kenichi Taniguchi: "LOTOS enhancement to specify time constraint among non-adjacent actions using first order logic," In Proc. of IFIP TC6/WG6.1 6th Int'l Conf. on Formal Description Techniques (FORTE'93), pp.451-466, IFIP, Elsevier Science Publishers B.V. (North-Holland), Oct. 1993.

2. Akio Nakata, Teruo Higashino, and Kenichi Taniguchi: "Protocol synthesis from timed and structured specifications," In Proc. of 1995 Int'l Conf. on Network Protocols (ICNP'95), pp.74-81, IEEE Computer Society Press, Nov. 1995.

3. Akio Nakata, Teruo Higashino, and Kenichi Taniguchi: "An extension of LOTOS for specifying time constraints among non-adjacent actions and verification of equivalence," Journal of Japan Society of Software Science and Technology, Vol.12, No.6, pp.3-16, Nov. 1995. (In Japanese)

4. Akio Nakata, Teruo Higashino, and Kenichi Taniguchi: "Deriving Protocol Specification from Timed Service Specifications Written in LOTOS," Trans. of Information Processing Society of Japan, Vol.37, No.5, pp.672-686, May 1996. (In Japanese)

5. Akio Nakata, Teruo Higashino, and Kenichi Taniguchi: "Time-Action Alternating Model for Timed LOTOS and its Symbolic Verification of Bisimulation Equivalence," in Proc. of IFIP TC6/WG6.1 Joint Int'l Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification (FORTE/PSTV'96), pp.279-294, Chapman & Hall, Oct. 1996.

6. Akio Nakata, Teruo Higashino, Kenichi Taniguchi: "Time-Action Alternating Model for Timed Processes and its Symbolic Verification of Bisimulation Equivalence", IEICE Trans. on Fundamentals (to appear).

Acknowledgment

This work could be achieved owing to a great deal of helps of many individuals.

First, I would like to thank my supervisor Professor Kenichi Taniguchi of Osaka University, for his continuous support, encouragement and guidance of the work.

I'm very grateful to Professor Mamoru Fujii and Professor Toru Kikuno for their invaluable comments and helpful suggestions concerning this thesis. I'm also very grateful to Professor Hideo Miyahara and Professor Nobuki Tokura, for their valuable comments on this thesis.

I would like to express my sincere gratitude to Associate Professor Teruo Higashino of Osaka University for his adequate guidance, valuable suggestions and discussions throughout this work. This work could not be achieved without his continuous support, encouragement and guidance.

Many of the courses that I have taken during my graduate career have been helpful to prepare this thesis. I would like to acknowledge the guidance of Professors Seishi Nishikawa, Kenichi Hagihara, Katsuro Inoue, Toshinobu Kashiwabara, Masaru Sudo, and Akihiro Hashimoto.

I'd like to express my thanks to Assistant Professor Masahiro Higuchi of Osaka University for his helpful comments and suggestions.

I'd like to express my thanks to Professor Hans A. Hansson of Uppsala University, Sweden for his kindly sending me his doctoral dissertation and some related references.

I also wish to thank Assistant Professor Keiichi Yasumoto of Shiga University and Research Associate Dr. Kozo Okano of Osaka University for their helpful suggestions.

Finally, I would like to thank all the members of Taniguchi Laboratory of Osaka University for their helpful advice.

Contents

Chapter 1

Introduction

1.1 Motivations

In recent years, distributed systems such as online services (telephone network, online bank services) and transport controlling systems (airplanes, trains) have become popular to our daily life with the progress of information networks. However, it is a difficult problem to make such a system reliable because of its nature of concurrency. Aside from bugs of each individual computer system and/or errors of communication media, mismatch of the communications among multiple computers may result some failures — errors, deadlocks, or incorrect computations of entire services. Because our daily life is heavily dependent on such a system, such a failure may cause serious damage to our society.

To cope with this problem, an approach so-called Formal Description Technique (FDT) for distributed systems and communication protocols are proposed. In this approach, services are formally specified by rigorous models or specification languages. Then the service specifications are modified or refined during design processes. In each steps, correctness of the modified or refined specification w.r.t. service specifications is verified with the help of computers. Alternatively, the refined specifications which is guaranteed correct w.r.t service specifications are automatically derived. Because it is aimed at verification of the correctness of communications between distributed nodes, input/output actions are mainly described in such an FDT.

Formal description languages such as CCS [Mil80], CSP [Hoa85], ACP [BK85], LOTOS [ISO89] and so on, have been proposed to specify communication protocols and distributed systems formally. Although these languages can express temporal ordering of the actions, they cannot express explicit time constraints among the actions. It is necessary for the real-time systems and communication protocols to specify quantitative time because of some reasons. Firstly, in some systems like air traffic controlling and train controlling, their correctness strongly depends on the *time* when the operation (or action) is performed. It is critical or even fatal if some operation does not finish in time. It is desirable to specify an external service, including its time constraints, of such a real-time application formally and verify whether its implementation satisfies the specification of the service. Secondly, in some applications like online bank systems and database systems, delay of each operation is not so critical. However, excessive delay is undesirable because it gives some inconvenience to users. In design processes of such a real-time service, time constraints of each operation may be frequently altered depending on its implementations. In such cases, we must guarantee that the system's essential behaviour would remain correct.

Moreover, when describing time constraints of service specifications of real-time distributed systems, it is frequently the case that only the time constraint between the initial request and the corresponding final response is required. In this case, specifying time constraints among some intermediate actions may be too restrictive. For example, someone would like to specify that the response c must be issued within 3 seconds after the request a was received, but the intermediate action b can be executed at any time between a and c. In this case, time constraints between b and c is unnecessary. In general, capability of describing time constraints of only interested set of (possibly non-adjacent) actions is required for specification languages of real-time systems. Generally, expressive power of a language and its complexity are trade-off. In order to check equivalence between specifications mechanically, we must keep the complexity of equivalence checking problems tractable when designing such a specification language.

Sometimes verification of equivalence between service and refined specifications is too difficult due to the complexity of communications and/or time-constraints. In this case, instead of writing the refined specifications by hand, it is useful to decompose the formal specification of its services into the more concrete specification of each distributed node automatically. Such a technique is known as *protocol synthesis*. A set of decomposed specifications is called a *protocol specification*. Protocol synthesis methods aim at decomposing a *wider* class of service specifications into *correct* and *better* protocol specifications. When we make the decomposed specifications of all nodes work together, their behaviour must be at least correct w.r.t. the service specification in a certain sense. Moreover, it is better to leave as many implementation possibilities as possible for the decomposed specification. It is also better that the performance of the decomposed specification is feasible. Specifically, we would like to reduce the number of messages exchanged between nodes.

However, it is difficult to consider protocol synthesis for real-time distributed systems. That is, the higher-level service of a real-time distributed system may have some timing constraints arisen from a user side (e.g. maximum response time requirements), while the lower-level implementation may also have some timing constraints imposed by physical reasons (e.g. propagation delay of communication media). It is difficult or even impossible in some cases to find a correct implementation of the service because it must satisfy the time constraints of both higher and lower levels. At least two different approaches to this problem are possible — a *lower level restriction* and a *higher level restriction*. In a lower level restriction approach, we derive both a protocol specification and time constraints of lower level communication media necessary to implement the given service specification. In a higher level restriction approach, we first restrict the given service specification in order to execute it correctly on distributed nodes and a given communication medium, and then derive a protocol specification from the restricted service specification. Since the delay of communication media generally depends on physical lines and difficult to change, the higher level restriction approach seems more applicable to systems of the real-world. To derive a better protocol specification in this approach, we must keep the service restriction minimum. Moreover, it is desirable to decompose a service specification written in a structured parallel language such as LOTOS, because such a structured language is more appropriate to write large practical system specifications than traditional state transition system models.

1.2 Specification Language and Equivalence

In this thesis, firstly we propose a formal language, LOTOS/T, for describing both service specifications and one-step-forward refined specification of its lower level implementations (protocol specifications) of real-time distributed systems. The proposed language LOTOS/T is a timed extension of LOTOS [ISO89]. The language LOTOS has been proposed as an international standard formal description language for specifying distributed systems and communication protocols by ISO. In LOTOS, we can specify temporal ordering of input/output/internal actions in a structural way using several operators like choice, parallel, sequential composition and interruption. In addition to LOTOS, LOTOS/T has a capability of specifying time constraints among possibly non-adjacent actions by a formula of first order arithmetics on time domain (may be either integers or real-numbers). For example, we can specify an action a is executable just when the current value of the clock is less than $3 + x$, where x is the time when the previous action b is executed, and after the execution of a, the executed time is assigned to a variable y. Specifying time constraints in this way is very flexible. Furthermore, if the logic describing time constraints is restricted to Presburger Arithmetics, that is, only addition($+$) and subtraction($-$) are used as functions and only equality($=$) and inequality($\leq$) as atomic predicates, we still obtain simple and tractable models for LOTOS/T in spite of its expressiveness, which is explained later.

Relating real-time aspects, we introduce two kinds of equivalence between two distributed communicating systems. One is timed bisimulation equivalence, and the other is untimed bisimulation equivalence. Moreover, we give the two kinds of approaches to the verification of timed/untimed bisimulation equivalence. One is a naive but natural extension of existing methods, and the other is a symbolic method which is an extension of [HL95].

Bisimulation equivalence (also called observation equivalence) is firstly defined in [Par81] for reasoning equivalence of communicating systems. Intuitively, two systems are bisimulation equivalent (bisimilar) if and only if they are indistinguishable by external communications (an observation). More formally, it is defined as an equivalence relation on states in a state transition model where each transition is labelled by the name of the action (called Labelled Transition System, LTS for short). Two states in an LTS are bisimilar if one state can perform some action, say a, and then reach some state, say s, then the other state can also perform the action a and then reach some state s' which is bisimilar to s, and vice versa. Bisimulation equivalence is good for a criterion of correctness of manipulation or refinement of specifications by the following reasons:

- It has a good compositionality. If two subsystems are bisimulation equivalent and each of them is embedded in the same environment, two instances of the environment (each of which contains possibly different but bisimulation equivalent subsystems) are also bisimulation equivalent (algebraically this property is known as a *congruence*). So, process algebras [HM85, Hoa85, BK85] have been established for reasoning semantics of communicating processes in algebraic (axiomatic) approaches.

- Its verification cost is feasible, in comparison with other reasonable equivalence relations for communicating systems proposed so far. In fact, its complexity is deterministic polynomial-time [KS90]. The verification algorithm can be applied to finite-state communicating systems, that is, the corresponding LTS contains only a finite number of states. (On the other hand, we can prove bisimulation equivalence of even infinite-state systems using algebraic (axiomatic) methods.)

Timed bisimulation equivalence is a time-sensitive version of bisimulation equivalence. Two real-time systems are timed bisimulation equivalent if and only if they carry out the same behaviour (in the sense of bisimulation equivalence) and each corresponding pair of actions can be performed at the same time. Untimed bisimulation equivalence is a time-insensitive version of bisimulation equivalence (for real-time systems). Two real-time systems are untimed bisimulation equivalent if and only if they carry out the same behaviour if the timing of actions are ignored. Timed bisimulation equivalence inherits all of the benefits of bisimulation equivalence—verification feasibility and compositionality. However, untimed bisimulation equivalence does not have compositionality [LW93, NHT94, ACH94], that is, axiomatic approaches to the proving untimed bisimulation equivalence are unlikely. Thus, in this thesis we verify these equivalence relations by

1. constructing the state transition models of real-time system specifications, and
2. checking the equivalence of the constructed models.

1.3 Semantic Models and Symbolic Verification of Equivalence

An LTS is adopted as a semantic model for LOTOS. In the LOTOS standard [ISO89], the inference rules are provided to derive transition relations between LOTOS expressions. Using the inference rules we can construct the corresponding LTS mechanically, and we can also verify bisimulation equivalence mechanically if it is finite-state. However, it is not capable for expressing timing informations. In this thesis, we propose two kinds of state transition models for LOTOS/T specifications. The first one is naive but it naturally extends the untimed model for LOTOS. We refer to the first model as tick-LTS. In tick-LTS, time domain is discrete, non-negative integers. One unit of time progress is expressed by the special action "tick". We consider any other (ordinary) actions take no time when performed. Timed bisimulation equivalence is defined as bisimulation equivalence of the tick-LTSs. Similarly, untimed bisimulation equivalence is defined as weak bisimulation equivalence of tick-LTSs where all tick actions are interpreted as internal actions. Both of timed and untimed equivalence can be checked similarly to the traditional bisimulation equivalence if the corresponding tick-LTSs are finite.

We have a serious problem using the tick-LTS for verifying equivalence of large specifications of practical systems— a state explosion problem. The size of a tick-LTS depends on not only the complexity of the control structure of the system but also the contents of the time constraints. For example, if we add a time constraint like "action a must be executed within 10,000 seconds", then 10,000 states are appeared in the corresponding tick-LTS. The problem itself seems to be solved if we select the most appropriate unit of time. However, if we have an action which is executable within an infinite time interval, the corresponding tick-LTS becomes infinite. Reducing redundant states may solve this problem in some cases. For example, it is frequently the case that after some time instant, further progress of time does not change the essential behaviour (including timing) of the system. But in the other case such a reduction is not possible due to the expressive power of our language. For example, if we have a time constraint such as "execute the action b within $2x$ (x times 2) seconds where x is the time the action a is executed (a is executable at any time)", we also have an infinite number of unreducible states in the tick-LTS. Moreover, we cannot take real numbers as a time domain in this approach.

Therefore, we introduce the second modeling for LOTOS/T specifications— A-TSLTS (Alternating Timed Symbolic Labelled Transition System) models. Each state in an A-TSLTS may have some parameter variables (e.g. x, y). Each transition in an A-TSLTS has a guard predicate such as 'execute the transition a when time between $x + 5$ to y seconds has elapsed.' The guard predicate of a transition may contain any parameter variable associated to its source state, any numerical operation on time domain, and any atomic predicate. We can use any logic. The logic only needs to be decidable in order to verify the equivalence in a proposed method. In this thesis, only timed transitions are considered (data-passing is ignored).

We model a time transition by a delay transition $\xrightarrow{e(d)}$ with a delay variable d, which stands for an amount of the delay (duration). This is the same as [HLW91]. This modeling has a merit that we can treat durations equally as input-output data. So, although we only handle time here, we can easily extend the result to the model which handles both time and data-passing. Moreover, each delay transition $\xrightarrow{e(d)}$ and action transition $\xrightarrow{a}$ have guard predicates which may contain the delay variables and the parameter variables at their source state (they possibly include some delay variables in previously executed delay transitions). We refer to such a model as "Timed Symbolic Labelled Transition System (TSLTS)."

Still we have a problem for verifying timed bisimulation equivalence symbolically using TSLTS. That is, a delay transition $\xrightarrow{e(d)}$ whose amount of delay is d, is equivalent to a sequence of delay transitions $\xrightarrow{e(d_1)}\xrightarrow{e(d_2)} \cdots \xrightarrow{e(d_n)}$ where $d_1 + d_2 + \cdots + d_n = d$. Note that in TSLTS, it is possible that after $\xrightarrow{e(d_1)}$ is executed, both $\xrightarrow{e(d_2)}$ and $\xrightarrow{a}$ are executable. So in general, the sequence $\xrightarrow{e(d_1)}\xrightarrow{e(d_2)}$ cannot be simply reduced to one transition $\xrightarrow{e(d_1+d_2)}$. In order to make a matching between two transitions which form a bisimulation, we must make a (possibly finitely many) sequence-to-sequence matching, which makes the problem difficult. Therefore, in this thesis, we assume our model to have *alternating* property. Each state of a TSLTS must belong to one of the two kinds of sets of states, the one is a set of *idle states*, and the other is a set of *active states*. From an idle state, only a delay transition is possible and then it moves to an active state. From an active state, some action transitions are possible. After one of them is executed, it comes back to an idle state. We call such a restricted TSLTS as an Alternating TSLTS (A-TSLTS). In an A-TSLTS model, we can make the bisimulation matching of delay transitions one-to-one.

Using an algorithm similar to [HL95], from a given state-pair we obtain the weakest condition (we refer to the condition as the *most general boolean*, *mgb* for short) which makes the chosen two states be timed bisimulation equivalent. If the condition is universally valid, the pair of states is timed bisimulation equivalent for any set of parameter values. If it is satisfiable, there is some set of parameter values which makes the pair of states be timed bisimulation equivalent. Otherwise, the pair of states is not timed bisimulation equivalent.

For example, let us consider the following two processes, P and Q. The process P may execute the action a when time between $x + 5$ to y seconds has elapsed, or execute the action b when time between y to $x+10$ seconds has elapsed. The process Q may execute the action a when time between 10 to z seconds has elapsed. In order to make P and Q bisimilar, the condition "$(x+5 = 10) \wedge (y = z) \wedge (y > x+10)$" must hold (if $(y > x + 10)$, then P cannot execute the action b). On the other hand, the condition is also a sufficient condition to make P and Q bisimilar. Such a condition is the mgb. In a proposed method, even if P and Q are infinite processes, if the corresponding A-TSLTS has finite states and variables, we can obtain the mgb for any pair of states. Once we obtain the mgb, we can verify whether the two states are timed bisimulation equivalent w.r.t. the specified parameter values by checking whether the values satisfy the mgb.

The algorithm we present takes a finite A-TSLTS and its state-pair as an input, and it outputs the mgb for the state-pair. We also show that the algorithm can be easily extended to verify untimed bisimulation equivalence.

1.4 Decomposition of Real-Time Services

Lastly we propose a method to synthesize a specification of each distributed node from a given service specification written in LOTOS/T and given time constraints on communication media. The problem to synthesize a specification of each node (which may contain some communicating actions) from a given service specification is known as a *protocol synthesis problem* [PS91]. We refer to each distributed entity as a *protocol entity* and the specification of each node as a *protocol entity specification*. And the set of all protocol entity specifications in the system is referred to as a *protocol specification*. Many proposals for synthesizing protocol entity specifications from a given service specification described in various models or specification languages have been appeared for untimed cases, but very few proposals for timed cases [KBD95]. This is due to the difficulty to consider the time constraints of both service and communication media, as noted previously. Here we adopt a higher level restriction approach and define the correctness criteria as follows. Informally, we say a protocol specification is correct w.r.t. a given service specification if and only if they are untimed bisimulation equivalent and executed time of each action of protocol specification satisfies the time constraints of the service specification. Then we give a solution to this problem as follows.

In our method, we assume that (a) each communication channel is error-free and its maximum propagation delay is bounded by a constant, and that (b) all nodes with their clocks can start their executions simultaneously and the clocks always synchronize each other. Under this assumption, we give a simulation policy for each node to execute actions in exactly the same order as specified in a given service specification. Basically, the simulation policy is based on the method which we have proposed in [KHB96, YHT94]. That is, after executing each action, say a, a synchronization message is sent to the node which executes a succeeding action, say b, to inform that a has been executed. If the execution time of a is needed, the time is also transmitted. The action b must be executed after the message is received. We derive protocol specifications under the above policy. However, if we consider time-constraints, many problems arise. For example, if a service specification states "the action a must be executed before time 3 at node 1, and then the action b must be executed before time 5 and $x + 3$ at node 2, where x is the time a is executed," and if the maximum communication delay from node 1 to node 2 is 3 units of time, the synchronization message sent from node 1 after a is executed may not reach node 2 before time 5. To cope with this kind of problem, we restrict, for example, the time constraint of the action a to "before time 2" so that we can guarantee the synchronization message reaches node 2 in time. As another example, suppose that a service specification states "the action a must be executed between time 1 and 3 at node 1, and after that the action b must be executed between time 4 and 5 at node 2". If the maximum communication delay from node 1 to node 2 is 3 units of time, the same observation as the previous example holds, i.e., the synchronization message from node 1 to node 2 may not reach in time. But as for the above case, a different solution is possible. Since each node has its own clock and all clocks synchronize each other, the ordering of actions a and b is guaranteed without any message exchange. That is, the temporal ordering as the total system is guaranteed if each node decides the execution time of its action a (or b) using its own clock.

In our derivation method, first, from a given service specification S and a given maximum delay of each channel, we derive a specification S' where additional time constraints are appended to S so that the message exchanges are carried out in time. We make only the weakest timing restrictions to S so that each node can simulate S under the above policy. If S and S' can execute the same behaviour while timing of actions are ignored (note that the transformation from S to S' does not necessarily preserve the equivalence), i.e., if they are untimed bisimulation equivalent when sending/receiving actions of synchronization messages are considered unobservable, then a protocol specification which is correct w.r.t. S is derived automatically from S'.

1.5 Related Work

1.5.1 Formal Description Languages for Real-Time Systems

In the latest years many languages have been proposed to describe real-time properties of systems [MT90, Wan91, HR91, BB91, QAF90, vHTZ90, AD90]. For example, timed extensions of CCS [MT90, Wan91, HR91], introduced several primitive operators such as delay and timeout operators to describe real-time properties. However, in these languages, even describing a simple time constraint that some action has to be done within a given time interval yields to a complicated description. Although ACP$_\rho$ [BB91, FK95], based on ACP, has an expressive power closer to ours. It can associate any time intervals ranged over reals to any action, and assign the executed time of actions into variables. However, the type of time constraints are still restricted to intervals, which is a less generic approach than ours. TIC [QAF90], based on LOTOS, is restricted to specify time constraints between only adjacent two actions. CELOTOS [vHTZ90] introduced clocks, which can be read or reset to zero, to describe time constraints among arbitrary actions. However, CELOTOS cannot specify urgency of actions[1]. Timed Automata [AD90] have been recognized as the most general models for real-time systems. A timed automaton has several clock variables, each of which can be read, reset, or compared to some integer constants to resolve a time constraint of each transition. Although time domain of Timed Automata is real numbers, only comparison between clock variables and integer constants are allowed in order to make the state space tractable. On the other hand, using A-TSLTS models, our language is more expressive because comparison among any combinations of variables, real constants, and linear expressions which may contain operators like addition($+$) and subtraction($-$) are allowed, while there still exists the decision procedure of the bisimulation equivalence.

1.5.2 Verification of Real-Time Properties

There are some proposals to solve the state explosion problem in verification of real-time properties (e.g. [HLW91, Ćer92, Che92, ACH94]) . But they all have some stronger restriction in describing time constraints of actions (The case for Timed Automata is already noted above. The other models used in [HLW91, Ćer92, Che92] are all less general than Timed Automata). On the other hand, for data-passing processes, a verification method of bisimulation equivalence is proposed [HL95, Lin96]. This method has some merits: (1). Its verification cost does not depend on the data domain which we choose or the amount of constants used in data constraints, and (2). the method does not depend on the logic which we choose for describing data

[1]We say that an action is urgent if the action must necessarily be executed at the current time. The urgency issue is mentioned in many papers including [BLT90, LL93].

constraints (although they should be decidable in order to verify the equivalence). Our method is based on [HL95].

1.5.3 Decomposing Real-Time Services

Several methods for synthesizing correct protocol specifications from given service specifications mechanically have been proposed so far for FSM, EFSM, LOTOS and Petri Net models [BG86, CL88, GB90, HOIT93, Hul95, KHB96, KBK89, Lan90, YOHT95]. However those proposals do not consider quantitative time constraints for the systems. Recently, in [KBD95], a method to derive protocol specifications from timed service specifications written in a FSM model has been proposed. [KBD95] adopts a lower level restriction approach, that is, time constraints of the communication media may be restricted while that of the services remain unchanged. In comparison with [KBD95], our method has an advantage that we can specify complicated ordering of actions in a structural way. Moreover we adopted a higher level restriction approach, which is appropriate for the situation that the delay of media is unchangeable.

1.6 Outline of This Thesis

The rest of this thesis is organized as follows.

In Chapter 2, syntax and semantics of the proposed specification language LOTOS/T are defined formally. The inference rules for deriving tick-LTSs are also given. Here, it is shown that tick-LTSs are constructed mechanically from any LOTOS/T expressions if time-constraints are described in Presburger Arithmetics. Moreover, definition of both timed and untimed bisimulation equivalence on the discrete timed models are defined.

In Chapter 3, an alternative approach to the verification of bisimulation equivalence of LOTOS/T expressions avoiding state explosion is presented. Firstly, A-TSLTS models are formally defined. Then, timed bisimulation equivalence on an arbitrary time domain is defined and the verification method for timed bisimulation equivalence is presented. The definition and verification method for untimed bisimulation equivalence is also given. Finally, inference rules for deriving A-TSLTSs from any LOTOS/T expressions are given.

In Chapter 4, the protocol synthesis method from a service specification described in LOTOS/T and maximum delay of each communication medium, is presented. Firstly, the protocol synthesis problem for real-time services is formalized. Then a restriction algorithm of a given service specification, which restricts the time constraints according to the maximum communication delays and structure of the service specifications, is presented. After that, the synthesis algorithm of the specification of each node is given. The limitation and possible extensions of the proposed algorithm is also discussed.

Chapter 5 concludes this thesis and presents some future work.

Chapter 2

LOTOS/T — A Formal Specification Language for Real-Time Distributed Systems

2.1 Introduction

In this chapter, we propose a language 'LOTOS/T'. LOTOS/T is a timed enhancement of Basic LOTOS. It allows us to describe time constraints by the 1st-order predicate logic formulas. The 1st-order predicate logic is well-studied, suitable for automatic verification and makes it easy to describe complicated constraints in 'as is' way. Time is considered discrete. Each process has its own time-table(clock), which is started when it is invoked. Time is expressed as a non-negative integer. The semantic model of LOTOS/T is the Labelled Transition System (LTS) used in LOTOS. Unit time progress is expressed by the action **tick**. We give the inference rules for constructing the LTS's from given LOTOS/T expressions. Time constraints are described by predicates on integers, which must contain a special free variable t (denotes the current time) and may contain other free variables, associated to each action. Use of equality($=$) and inequality ($\leq$) in the predicate will enable us to describe intervals, timeout or delay easily and naturally. Moreover, time at which an action occurred can be assigned to a variable. So it is possible to describe time constraints against actions which are not direct successors. For upward compatibility, if no predicate are associated to the action, the predicate 'true' is assumed for its time constraint. In this case, the action is considered executable at any moment (not urgent). The LTS's can be constructed from given LOTOS/T expressions mechanically using the inference rules.

Two equivalences are introduced, the first is timed (strong/weak) bisimulation equivalence and the last is untimed bisimulation equivalence. Timed bisimulation equivalence is used for checking whether two systems are equivalent and have the same time constraints. Untimed bisimulation equivalence is used for checking whether two systems are equivalent in spite of the different time constraints. If the corresponding LTS's are finite, we can easily check the two bisimulation equivalences by the algorithms in [KS90, SKTN90].

This chapter is organized as follows. In Section 2.2, the syntax and semantics of LOTOS/T are defined formally. In Section 2.3, the definition of equivalences related to timed semantics is given. In Section 2.4, a simple but practical example is provided. Section 2.5 concludes this chapter.

2.2 Definition of LOTOS/T

2.2.1 Syntax

The syntax of LOTOS/T is defined as follows.

Definition 2.1 *Behaviour expressions* of LOTOS/T are defined as follows (the priority of operators are analogous to LOTOS):

$$\begin{array}{rll}
E := & \mathbf{stop} & \text{(non-temporal deadlock)} \\
\mid & \mathbf{exit} & \text{(successful termination)} \\
\mid & a; E & \text{(untimed action prefix)} \\
\mid & a[P(t,\bar{x})]; E & \text{(time constrained action prefix)} \\
\mid & E[]E & \text{(choice)} \\
\mid & E|||E & \text{(interleaving)} \\
\mid & E||E & \text{(synchronization)} \\
\mid & E|[A]|E & \text{(generic parallel composition)} \\
\mid & E[> E & \text{(disabling)} \\
\mid & E >> E & \text{(enabling)} \\
\mid & \text{hide } A \text{ in } E & \text{(hiding)} \\
\mid & \text{asap } A \text{ in } E & \text{(``as soon as possible'' execution)} \\
\mid & P[g_1, \ldots, g_k](\bar{e}) & \text{(process invocation)}
\end{array}$$

where $a \in Act \cup \{i\}$ (Act denotes a finite set of all observable actions, i denotes an internal action) , $A \subset Act$, $k \in N$ (N denotes a set of natural numbers), and $P(t, \bar{x})$ stands for a predicate which has a free variable t, denoting the current time, and other variables $\bar{x}$ ($\bar{x}$ denotes a vector of the variables). $\bar{e}$ denotes a vector of the value-expressions.

Predicates are well-formed formulas of 1st-order theory of integers containing $=$, $\leq$ as atomic predicates, $+$, $-$ as functions. Var denotes a set of all variables of the 1st-order theory. Note that this 1st-order theory is decidable because it is, essentially, a subset of Presburger Arithmetics[HU79]. □

First, we will give an informal explanation of LOTOS/T.

Example 2.1 $B = a[2 \le t \le 3 \land x_0 = t]; b[t = x_0 + 3]; \mathbf{stop}$
B denotes a process which executes a between time 2 and 3 and executes b after 3 unit of time elapsed. The predicate $x_0 = t$ denotes that the executing time of a is assigned to the variable x_0. □

The semantic model of LOTOS/T is the LTS. We intend that the LTS in Figure 2.1 denotes the operational semantics of B.

This LTS is obtained as follows. In Figure 2.1, the root node corresponds to B. First, only the unit time progress action **tick** is executable for B. Therefore, the edge $\xrightarrow{\mathbf{tick}}$ is appended to the root node. If the **tick** is executed, then one unit time elapsed. Since the current time is incremented, $[t+1/t]B$ is obtained as the new behaviour expression. Here, $[e/x]B$ denotes a behaviour expression B whose every occurrence of the variable x is replaced with the expression e.

At the state $[t+1/t]B$, only **tick** is executable. Then $[t+1/t]B \xrightarrow{\mathbf{tick}} [t+2/t]B$ is appended. At the state $[t+2/t]B$, the **tick** and action a are executable. If the **tick** is executed, then $[t+3/t]B$, that is, $a[2 \le t+3 \le 3 \land x_0 = t+3]; b[t+3 = x_0+3]; \mathbf{stop}$ is obtained. If the **tick** was executed for $[t+3/t]B$, then the action a could never be executed. In this case, we say that the action a is urgent, that is, the action a must be executed immediately (before the **tick** is executed). Then only a is executable. If a is executed, then "0" is assigned[1] to the variable t in the predicate "$2 \le t+3 \le 3 \land x_0 = t+3$", which has already been aged by 3 units of time from the initial predicate "$2 \le t \le 3 \land x_0 = t$" through the operation $[t+3/t]$. Since $x_0 = t+3$, the value of the variable x_0 is fixed to 3, and $b[t+3 = 3+3]; \mathbf{stop}$ is obtained as the new state (behaviour expression). So b is executed after 3 units of time are elapsed.

Next, we will give a formal definition of LOTOS/T. First, we will introduce the notion of the predicate contexts and defined/undefined variables. In order to discuss whether satisfiability of predicates are decidable, we must define which variables have some fixed values and which ones are not. Consider a predicate "$t = x^5 - 7x^3 + 4$". If some value is assigned to x, satisfiability of the predicate is easy to decide for any given values of t. However, if no values are assigned to x, for a given value of t, a 5th-degree equation must be solved to decide satisfiability. So we need a formal definition of whether or not a variable's value is defined. We also need a notion of predicate contexts because the definition of a defined/undefined variable depends on where the variable occurs in a behaviour expression of LOTOS/T.

[1]Please note that assigning 0 to t in the 3 units of time aged predicate "$2 \le t+3 \le 3 \land x_0 = t+3$" is equivalent to assigning 3 to the variable t in the initial predicate "$2 \le t \le 3 \land x_0 = t$." We only have to check satisfiability of the predicate in a case of $t = 0$, because behaviour expressions are properly aged when $\xrightarrow{\mathbf{tick}}$ is added, in order to treat current time as 0.

Definition 2.2 *Predicate contexts* are syntactically defined by the following BNF. Here, E is the syntactical component representing a behaviour expression which is used in Definition 2.1.

$$\begin{array}{rcl} C & := & a[\bullet]; E \mid a[P(t,\bar{x})]; C \\ & & \mid C[]E \mid E[]C \mid C|[A]|E \mid E|[A]|C \\ & & \mid C|||E \mid E|||C \mid C||E \mid E||C \\ & & \mid C[> E \mid E[> C \mid C >> E \mid E >> C. \end{array}$$ □

For example, let us consider the behaviour expression B in Example 2.1. For this behaviour expression B, the following two predicate contexts are possible:

$$C = a[\bullet]; b[t = x_0 + 3]; \mathbf{stop}$$
$$C' = a[2 \leq t \leq 3 \wedge x_0 = t]; b[\bullet]; \mathbf{stop}$$

Here, "$\bullet$" denotes a time constraint of the current action. In the context C, the variable "x_0" is undefined because the value of the variable "x_0" is not fixed until a is executed. However, in the context C', the variable x_0 is defined because the value of x_0 has been fixed before b is executed.

Formally, the defined/undefined variable are decided as follows. Here, $DVar(C)$ and $UVar(C)$ denote the sets of defined/undefined variables for a predicate context C, respectively.

Definition 2.3 For any predicate context C, $DVar(C) \subset Var$ is defined recursively as follows.

$$\begin{array}{rcl} DVar(a[\bullet]; E) & \overset{def}{=} & \emptyset \\ DVar(a[P(t,\bar{x})]; C) & \overset{def}{=} & \{y | y \text{ is an element of } \bar{x}\} \cup DVar(C) \\ DVar(C \triangle E) & \overset{def}{=} & DVar(C) \\ DVar(E \triangle C) & \overset{def}{=} & DVar(C) \\ & & (\triangle \in \{[], |[A]|, [>, >>\}) \end{array}$$

And $UVar(C) \overset{def}{=} Var - DVar(C)$. □

Hereafter, we define the set of predicates $P(t, \bar{x})$ which can be used in the predicate context C. We believe that the class of predicates $Pres(C)$ defined in Definition 2.4 is reasonably wide, because time interval, whose bounds are expressed in linear expressions, can be written and the executed time of any preceded actions are referred to in the expressions.

Definition 2.4 A set of predicates allowed to use in the predicate context C, denoted as $Pres(C)$, is defined as a minimum set which satisfies the following conditions:

- "$e_l \leq t \leq e_u$", "$e_l \leq t$" and "$t \leq e_u$" are in $Pres(C)$. Here, e_l and e_u denote arbitrary terms consisting of only integers, the variables in $DVar(C)$, and operators $+$ and $-$. If e_l and e_u are the same, then "$e_l \leq t \leq e_u$" is abbreviated to "$t = e_u$".

- if $P \in Pres(C)$ and $x \notin FVar(P) \cup DVar(C)$, then "$P \wedge (x = t)$" is in $Pres(C)$.

- if $P_1, P_2 \in Pres(C)$ and $FVar(P_1) \cap FVar(P_2) \cap UVar(C) = \emptyset$, then both "$P_1 \vee P_2$" and "$P_1 \wedge P_2$" are in $Pres(C)$.

- if $P \in Pres(C)$ and $FVar(P) \cap UVar(C) = \emptyset$, then "$\neg P$" is in $Pres(C)$.

where $FVar(P)$ denotes a set of all free variables occurred in a predicate P. $\square$

Note that the predicate $P(t, \bar{x})$ may be described as $P(t, \bar{x}_d, \bar{x}_u)$ if necessary, where the 2nd parameter $\bar{x}_d$ denotes a vector of the defined variables in $\bar{x}$ and the 3rd parameter $\bar{x}_u$ denotes a vector of the undefined variables in $\bar{x}$ under C.

Next, we will explain that $Pres(C)$ defined in Definition 2.4 is restrictive in spite of its expressive power, that is, satisfiability of a predicate in the class is still decidable. For convenience, we refer to having such a desirable property as *normal*.

First of all, normal predicates are defined.

Definition 2.5 A predicate $P(t, \bar{x})$ is *normal* under a context C if $P(t, \bar{x})$ satisfies the following conditions.

1. (decidability) For any $n \in N$ and $\bar{v}$, satisfiabilities of the two formulas $P(n, \bar{v}, \bar{x}_u)$ and $(\mathcal{F}P)(n, \bar{v}) \overset{def}{=} \exists t' \exists \bar{x}_u [t' \geq n \wedge P(t', \bar{v}, \bar{x}_u)]$ are decidable.

2. (uniqueness of substitution) For any $n \in N$ and $\bar{v}$, there exist unique values $\bar{c}$ such that $P(n, \bar{v}, \bar{c})$ holds if the formula $\exists \bar{x}_u P(n, \bar{v}, \bar{x}_u)$ is satisfiable. Also such values $\bar{c}$ are computable from n and $\bar{v}$ i.e. there exists a partial recursive function $\phi_P(n, \bar{v})$ such that $\exists \bar{x}_u P(n, \bar{v}, \bar{x}_u)$ implies $P(n, \bar{v}, \phi_P(n, \bar{v}))$.

Remark: Condition 1 is needed to make sure that we can construct the semantical model of the expression mechanically. Condition 2 is needed to avoid ambiguity of assigned value to be assigned to variables. $\square$

We say a behaviour expression B is normal iff all predicates appeared in B are normal under its contexts, i.e. for any C and P such that $B = C(P)$, P is normal under C.

For the elements of $Pres(C)$, the following property holds.

Proposition 2.1 *For any context C, all the predicates in $Pres(C)$ are normal.*

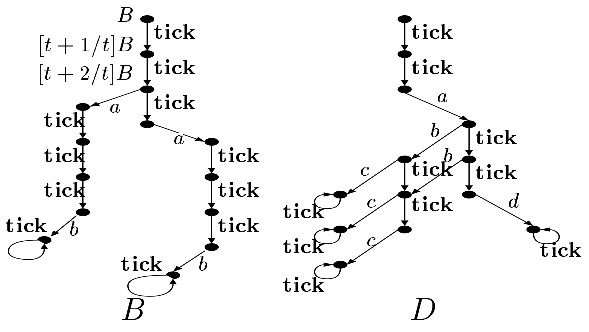

Figure 2.1: The semantics of B and D

Proof. Since each predicate P in $Pres(C)$ is described as a logical combination of some integer linear inequalities, P and $\mathcal{F}P$ in Definition 2.5 are expressions in Presburger Arithmetics [HU79]. Since it is known that satisfiability of Presburger Arithmetics is decidable [HU79], satisfiabilities of P and $\mathcal{F}P$ are also decidable. Therefore, Condition 1 in Definition 2.5 holds. Condition 2 also holds by the following observation. The minimum predicate which contain undefined variables in $Pres(C)$ is "$P \wedge (x = t)$." Appearently Condition 2 holds for this predicate. Suppose both P_1 and P_2 satisfy condition 2 and $P_1 \wedge P_2 \in Pres(C)$. Since $FVar(P_1) \cap FVar(P_2) \cap UVar(C) = \emptyset$, there are no common undefined variables in P_1 and P_2. Thus, Condition 2 holds for $P_1 \wedge P_2$. The case for $P_1 \vee P_2$ is similar. Moreover, if $\neg P \in Pres(C)$, then $FVar(P) \cap UVar(C) = \emptyset$, i.e., P contains no undefined variables. Thus, Condition 2 hold clearly for $\neg P \in Pres(C)$. □

$Pres(C)$ is useful for describing normal predicates. If other class of 1st-order theory is considered, the conditions in Definition 2.5 does not always hold.

Example 2.2 Under the predicate context "$a[t = x]; b[\bullet]; \mathbf{stop}$", "$t = x^2 + 2x + 1 \wedge y = 2t$" satisfies the conditions 1 and 2 of Definition 2.5. However, "$t = x^2 + 2x + 1 \wedge y > t$" violates the condition 2, and "$t = y^5 + 9y^2z^3 + z^4$" (Diophantine polynomial) violates both. □

Example 2.3 The following example is also possible. The LTS for D is shown in Figure 2.1. In this example, time constraints between non-adjacent actions (the actions a and c) are described.

$$D = a[t = 2 \wedge x_0 = t]; (b[x_0 \leq t \leq x_0 + 1]; c[x_0 \leq t \leq x_0 + 2]; \mathbf{stop}$$
$$[]d[t = x_0 + x_0]; \mathbf{stop})$$ □

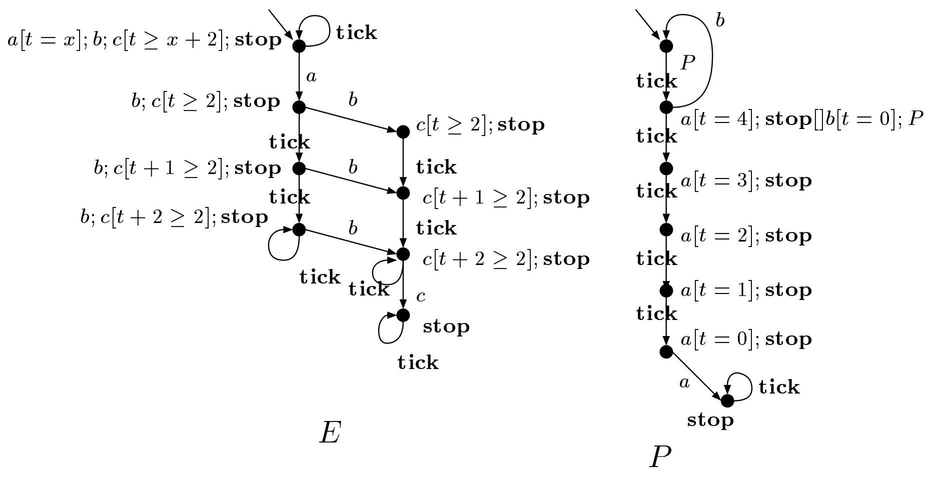

Figure 2.2: The semantics of E and P

Example 2.4 Other examples are given below. The first one contains untimed action sandwiched between time-constrained actions, and infinite interval for the time constraint. The second one describes an infinite behaviour.

1. $E = a[x = t]; b; c[t \geq x + 2]; stop$

2. $P = a[t = 5]; stop[]b[t = 1]; P$

The corresponding LTS's are shown in Figure 2.2. □

In LOTOS/T, untimed or infinite behaviours may be described (for example, the processes E and P in Example 2.4).

2.2.2 Operational Semantics

In this section, we will give the formal semantics of LOTOS/T. The operational semantics of LOTOS/T is an extension of LOTOS. The difference is the treatment of transitions of the extra action **tick**. Here we define the operational semantics of LOTOS/T by giving an inference system of the transition relation (see Tables 2.1 and 2.2).

Inaction

The behaviour expression **stop** is extended to express *non-temporally deadlocked process*, which cannot do any other computations except the infinite sequence of

tick. The behaviour expression **exit** is extended to execute **tick** actions any times before executing δ action[2].

Action Prefix

The behaviour expression $a[P(t,\bar{x})];B$ means that the action a can occur at time n if $P(n,\bar{c})$ holds for some $\bar{c}$. Because the predicate P is assumed to be normal, satisfiability of $P(n,\bar{x})$ is decidable (from condition 1 of Definition 2.5), and the value $\bar{c}$ which satisfies $P(n,\bar{c})$ is uniquely computable (from condition 2).

In order to express urgency, we define that the action **tick** cannot occur if the action cannot happen in the future, i.e. $\mathcal{F}P(1) \equiv \exists t' \exists \bar{x}[t' \geq 1 \wedge P(t',\bar{x})]$ does not hold. Satisfiability of $\mathcal{F}P(1)$ is also decidable (from condition 1).

The semantics of the untimed action prefix, $a;B$, is the same as that of $a[true];B$.

Internal Action

For the behaviour expression $i;B$, the internal action is considered always urgent, so its execution is prior to **tick** action. The rest is similar to action prefix.

Choice

We define the choice operator be weak-choice[MT90], so our choice operator is non-persistent. For example, "$a[t=1];$**stop**" and "$b[t=2];$**stop**" are equivalent to "**tick**$;a;$**stop**" and "**tick**$;$**tick**$;b;$**stop**", respectively. However, "$a[t=1];$**stop**$[]b[t=2];$**stop**" is not equivalent to "**tick**$;a;$**stop**$[]$**tick**$;$**tick**$;b;$**stop**" because the choice is occurred at time 0. It must be equivalent to "**tick**; $(a;$**stop** $[]$ **tick**; b; **stop**)". The inference rules in Table 2.1 are introduced to construct the latter semantic model.

Parallel

Parallel operators ($|||$, $||$, $|[A]|$) always synchronize **tick** actions in LOTOS/T. Consequently, the time constraint of interaction is the logical product of the time constraints of the actions in both processes.

ex.) In $a;b[2 \leq t \leq 4];$**stop**$|[b]|c;b[3 \leq t \leq 5];$**stop**, the time constraint of the interaction b is $3 \leq t \leq 4$.

Disable

The definition is similar to LOTOS except the **tick** action.

[2] δ denotes a successful termination, which is the same as LOTOS [ISO89].

Table 2.1: The inference rules of transition relation: Part 1

Inaction			
$\dfrac{—}{\mathbf{stop} \xrightarrow{\mathbf{tick}} \mathbf{stop}}$	(2.1)	$\dfrac{—}{\mathbf{exit} \xrightarrow{\delta} \mathbf{stop}}$	(2.2)
$\dfrac{—}{\mathbf{exit} \xrightarrow{\mathbf{tick}} \mathbf{exit}}$	(2.3)		
Action Prefix			
$\dfrac{P(0,\bar{c})}{a[P(t,\bar{x})]; B \xrightarrow{a,\bar{c}/\bar{x}} B}$	(2.4)	$\dfrac{\mathcal{F}P(1)}{a[P(t,\bar{x})]; B \xrightarrow{\mathbf{tick}} a[P(t+1,\bar{x})]; [t+1/t]B}$	(2.5)
$\dfrac{—}{a; B \xrightarrow{a} B}$	(2.6)	$\dfrac{—}{a; B \xrightarrow{\mathbf{tick}} a; [t+1/t]B}$	(2.7)
Internal Action			
$\dfrac{P(0,\bar{c})}{i[P(t,\bar{x})]; B \xrightarrow{i,\bar{c}/\bar{x}} B}$	(2.8)	$\dfrac{\neg P(0,\bar{x}) \quad \mathcal{F}P(1)}{i[P(t,\bar{x})]; B \xrightarrow{\mathbf{tick}} i[P(t+1,\bar{x})]; [t+1/t]B}$	(2.9)
$\dfrac{—}{i; B \xrightarrow{i} B}$	(2.10)		
Choice			
$\dfrac{B_1 \xrightarrow{\beta} B_1'}{B_1 [] B_2 \xrightarrow{\beta} B_1'}$ iff $\beta \in Act \cup \{\delta, i\}$	(2.11)	$\dfrac{B_2 \xrightarrow{\beta} B_2'}{B_1 [] B_2 \xrightarrow{\beta} B_2'}$ iff $\beta \in Act \cup \{\delta, i\}$	(2.12)
$\dfrac{B_1 \xrightarrow{\mathbf{tick}} B_1' \quad B_2 \xrightarrow{\mathbf{tick}} B_2'}{B_1 [] B_2 \xrightarrow{\mathbf{tick}} B_1' [] B_2'}$	(2.13)		
$\dfrac{B_1 \xrightarrow{\mathbf{tick}} B_1' \quad B_2 \overset{\mathbf{tick}}{\not\longrightarrow}}{B_1 [] B_2 \xrightarrow{\mathbf{tick}} B_1'}$	(2.14)	$\dfrac{B_2 \xrightarrow{\mathbf{tick}} B_2' \quad B_1 \overset{\mathbf{tick}}{\not\longrightarrow}}{B_1 [] B_2 \xrightarrow{\mathbf{tick}} B_2'}$	(2.15)

Enable

Similar to LOTOS, except **tick** synchronizes unconditionally and enabling is prior to the **tick** action.

Hide

Similar to LOTOS. In order to express uncertain delay of internal messages, we define hidden internal actions are not executed as soon as they are enabled. Instead, maximal progress property of actions are expressed by **asap** construct, which is defined in the next subsection.

“As Soon As Possible” Execution

To describe maximal progress property of actions, we define a construct “**asap** A in B,” representing the same behaviour B except the actions in A must be executed as soon as possible they are enabled.

Process Invocation

A process invocation behaves exactly the same as the behaviour at time 0, no matter when it is invoked.

2.2.3 Consistency of the inference system

It is very important to notice that our inference system, used for defining operational semantics, are consistent. An inference system is called consistent if the existence of a transition is never deduced from the non-existence of the transition itself. If an inference system is inconsistent, the semantic model cannot exist. Unfortunately, our inference system contains negative premises in some inference rules. So consistency is not self-evident. However, our inference rules can be proved consistent by using the *stratification* technique described in [Gro90].

2.2.4 Example of LTS construction

By applying the inference rules shown in this section, we can construct the corresponding LTS as follows. Let us consider the process E in Figure 2.2.

- $E = a[t = x]; b; c[t \geq x + 2]; \mathbf{stop} \xrightarrow{a} b; c[t \geq 2]; \mathbf{stop}$ (by rule (2.4)),
- $b; c[t \geq 2]; \mathbf{stop} \xrightarrow{\mathbf{tick}} b; c[t + 1 \geq 2]; \mathbf{stop}$ (by rule (2.5)),

and so on.

For the process P in Figure 2.2, the following actions are possible:

- $P \xrightarrow{\mathbf{tick}} a[t = 4]; \mathbf{stop}[]b[t = 0]; P$ (by rules (2.34), (2.13), (2.5)),
- $a[t = 4]; \mathbf{stop}[]b[t = 0]; P \xrightarrow{\mathbf{tick}} a[t = 3]; \mathbf{stop}$ (by rules (2.14) and (2.5)),

and so on.

Note that we regard two states as the same if satisfiability of the corresponding predicates for each t on $0 \leq t < \infty$ are equivalent. For instance, w.r.t. E in Figure 2.2, $a[t = x]; b; c[t \geq x + 2]; \mathbf{stop} \xrightarrow{\mathbf{tick}} a[t + 1 = x]; b; c[t + 1 \geq x + 2]; \mathbf{stop}$ holds by the inference rules. Here, satisfiabilities of two predicates of the action a, $t = x$ and $t+1 = x$, are equivalent, i.e.

$$\forall t[0 \leq t \Rightarrow \exists x[t = x] \equiv \exists x'[t+1 = x']] \tag{2.35}$$

holds (Note that x in "$t = x$" and x in "$t+1 = x$" have no longer the same value. So we describe the latter formula as "$t+1 = x'$").

Furthermore, for any value assignment of x and x', satisfying (2.35), into two predicate two predicates of the action c,

$$\forall t'[0 \leq t' \Rightarrow [t' \geq x+2] \equiv [t'+1 \geq x'+2]] \tag{2.36}$$

holds.

To summarize the idea above, we can verify whether E and $[t+1/t]E$ are representing the same state by checking satisfiability of the following predicate:

$$\begin{aligned} \forall t_1[0 \leq t_1 \Rightarrow [\exists x(t_1 = x) \equiv \exists x'(t_1+1 = x')] \wedge \\ \forall x \forall x'[(t_1 = x) \wedge (t_1+1 = x') \Rightarrow \\ \forall t_2[0 \leq t_2 \Rightarrow [(t_2 \geq x+2) \equiv (t_2+1 \geq x'+2)]]]] \end{aligned} \tag{2.37}$$

So we can state $a[t = x]; b; c[t \geq x+2]; \mathbf{stop} \xrightarrow{\mathbf{tick}} a[t = x]; b; c[t \geq x+2]; \mathbf{stop}$ (i.e. this node has a self loop of **tick**).

Aging (replacing t with $t+1$) does not have an effect on process invocation, since process name does not have the variable t literally. For example, w.r.t. P in Figure 2.2, $P \xrightarrow{\mathbf{tick}} a[t = 4]; \mathbf{stop}[]b[t = 0]; P \xrightarrow{b} P$ holds by the inference rules. So the corresponding LTS has a cycle, as shown in Figure 2.2.

2.3 Equivalence

2.3.1 Timed Bisimulation Equivalence

Definition 2.6 A relation $\mathcal{R}$ is *timed strong bisimulation* if the following condition holds.

if $B_1 \mathcal{R} B_2$, then for any $a \in Act \cup \{\delta, \mathbf{tick}\}$, the following two conditions hold:

1. if $B_1 \xrightarrow{a} B_1'$, then $\exists B_2'[B_2 \xrightarrow{a} B_2'$ and $B_1' \mathcal{R} B_2']$
2. if $B_2 \xrightarrow{a} B_2'$, then $\exists B_1'[B_1 \xrightarrow{a} B_1'$ and $B_1' \mathcal{R} B_2']$ □

Definition 2.7 The behaviour expressions B and B' are *timed strong bisimulation equivalent*, denoted by $B \sim_t B'$, iff there exists a timed strong bisimulation $\mathcal{R}$ such that $B \mathcal{R} B'$. □

Timed weak bisimulation equivalence ($\approx_t$), where the internal action i is considered unobservable, can also be defined similarly.

Example 2.5 The following two behaviour expressions are timed strong bisimulation equivalent:

$$\begin{array}{l} B = a[2 \leq t \leq 3 \wedge x_0 = t]; b[t = x_0 + 3]; B \\ C = a[t = 2]; b[t = 5]; C[]a[t = 3]; b[t = 6]; C \end{array}$$

2.3.2 Untimed Bisimulation Equivalence

Here we introduce an *untimed bisimulation equivalence* where **tick** is considered unobservable. Using this equivalence, we can prove whether two timed expressions execute the same observable event sequences. Like timed bisimulation equivalence, untimed bisimulation equivalence has two definitions, one is *untimed strong bisimulation equivalence*, where only **tick** is considered unobservable, and the other is *untimed weak bisimulation equivalence*, where both **tick** and i are considered unobservable.

Definition 2.8 For each action $a \in (Act \cup \{\delta\} - \{\textbf{tick}\}) \cup \{\epsilon\}$, the relation $\overset{a}{\Longrightarrow}$ over behaviour expressions is defined as follows:

$$B \overset{a}{\Longrightarrow} B' \overset{def}{=} \begin{cases} B(\overset{\textbf{tick}}{\longrightarrow})^* \overset{a}{\longrightarrow} (\overset{\textbf{tick}}{\longrightarrow})^* B', & \text{if } a \in Act \cup \{\delta\} - \{\textbf{tick}\} \\ B(\overset{\textbf{tick}}{\longrightarrow})^* B' & \text{if } a = \epsilon \end{cases} \qquad \square$$

Definition 2.9 A relation $\mathcal{R}$ is *untimed strong bisimulation* if the following condition holds:

if $B_1 \mathcal{R} B_2$, then for any $a \in (Act \cup \{\delta\} - \{\textbf{tick}\}) \cup \{\epsilon\}$, the following conditions hold:

1. if $B_1 \overset{a}{\Longrightarrow} B_1'$, then $\exists B_2' [B_2 \overset{a}{\Longrightarrow} B_2'$ and $B_1' \mathcal{R} B_2']$
2. if $B_2 \overset{a}{\Longrightarrow} B_2'$, then $\exists B_1' [B_1 \overset{a}{\Longrightarrow} B_1'$ and $B_1' \mathcal{R} B_2']$ $\square$

Definition 2.10 The behaviour expressions B and B' are *untimed strong bisimulation equivalent*, denoted by $B \sim_u B'$, iff there exists a weak bisimulation $\mathcal{R}$ such that $B \mathcal{R} B'$. $\square$

Untimed weak bisimulation equivalence, denoted by $\approx_u$, can be defined similarly.

Proposition 2.2 *The behaviour expressions which are timed strong[weak] bisimulation equivalent are untimed strong[weak] bisimulation equivalent, respectively, i.e.:*

$$\begin{array}{l} B \sim_t B' \Rightarrow B \sim_u B' \\ B \approx_t B' \Rightarrow B \approx_u B' \end{array} \qquad \square$$

Proposition 2.3 $\sim_u$ *is not a congruence, i.e.:*

$$\exists B_1, B_2[(B_1 \sim_u B_2) \wedge (B[]B_1 \not\sim_u B[]B_2)]$$

Proof. Choose $B_1 = a[t = 0]; \mathbf{stop}$, $B_2 = a[t = 2]; \mathbf{stop}$ and $B = b[t = 1]; \mathbf{stop}$. $\square$

Note that from Proposition 2.3, untimed bisimulation equivalence is hardly suitable for axiomatic proof system.

Example 2.6 Let B and D denote the following expressions, respectively:

$$B = a[2 \le t \le 3 \wedge x_0 = t]; b[t = x_0 + 3]; \mathbf{stop}$$
$$D = a[t = 2]; \mathbf{stop}|||b[3 \le t \le 5]; \mathbf{stop}$$

Then, B and D are untimed strong bisimulation equivalent because

$$\begin{aligned}\mathcal{R} = \{&([t+k/t]B, [t+l/t]D) | 0 \le k \le 3 \wedge 0 \le l \le 2\} \\ \cup \{&(b[t+k = m+3]; \mathbf{stop}, b[3 \le t+l \le 5]; \mathbf{stop}) | 2 \le m \le 3 \wedge \\ &\quad k \le m+3 \wedge 3 \le l \le 5\} \\ \cup \{&(\mathbf{stop}, \mathbf{stop})\}\end{aligned}$$

is an untimed strong bisimulation which satisfies $B\mathcal{R}D$. $\square$

In the following Proposition, we mention the decidability of these equivalences.

Proposition 2.4 *If the corresponding LTS's of both B_1 and B_2 are finite, then all the equivalences defined above are decidable.*

Proof. Analogous to [KS90, SKTN90]. $\square$

Note that the corresponding LTS of a behaviour expression is not always finite, but if the LTS is finite, then equivalences are decidable from Proposition 2.4.

2.4 Example

Here we introduce a more practical example. The example shown in Figure 2.3 models a remote controller or something that has only one press button for input and executes 4 output actions according to the timing patterns of pressing button. The timing patterns are:

- long click once,
- short click once,

```
ONE_KEY_CONTROLLER[p,r,lc,sc,dc]
     := p[t1p=t];
          (lc[t1p+d1<=t<=t1p+d4];r;ONE_KEY_CONTROLLER
           [] r[t<t1p+d1];
               (p[t<t1p+d2 and t2p=t];
                     (r[t<t2p+d3];dc[t2p+d3<=t<=t1p+d4];ONE_KEY_CONTROLLER
                     [] slc[t2p+d3<=t<=t1p+d4];r;ONE_KEY_CONTROLLER)
                [] sc[t1p+d2<=t<=t1p+d4];exit)
           )

+ variables
  t1p: time when the first press occurred.
  t2p: time when the second press occurred.
+ constants
  d1: threshold for the first short or long click
  d2: timeout for the second click
  d3: threshold for the second short or long click
  d4: required maximum total delay between button press and result action
```

Figure 2.3: Timed specification of one-key controller

- double short click and
- short click followed by long click.

The second one is used for terminating, while others are continued to be accepted infinitely. Pressing button is modeled by the sequence of the actions `p` (short for 'press') and `r` (short for 'release'). The corresponding output actions are `lc` (short for 'long click'), `sc` (short for 'short click') and `dc` (short for 'double click') and `slc` (short for 'short and long click'). If `d2+d3>d4`, it may cause violation of time constraint (temporal deadlock). And if `d1>=d2`, it may cause second click be lost. So the sound implementation must satisfy `d1<d2` and `d2+d3<=d4`.

This will be checked by constructing the LTS for some values to `d1`,`d2`,`d3` and `d4` satisfying above. In the LTS, the temporally deadlocked state has no outgoing arc including **tick**. Whether or not the behaviour has been modified because of the time constraint is checked by verifying untimed bisimulation equivalence with the untimed specification like Figure 2.4.

2.5 Conclusion

We have proposed a language LOTOS/T, a timed enhancement of Basic LOTOS. LOTOS/T enables us to describe time constraints among actions in a flexible way using formulas of 1st-order theory.

```
UNTIMED_ONE_KEY_CONTROLLER[p,r,lc,sc,dc]
     :=p;(i;lc;r;UNTIMED_ONE_KEY_CONTROLLER
          [] r;(p;(r;dc;UNTIMED_ONE_KEY_CONTROLLER
                   []i;slc;r;UNTIMED_ONE_KEY_CONTROLLER)
                []i;sc;exit)
          )
```

Figure 2.4: Untimed specification of one-key controller

In order to construct the LTS from a given LOTOS/T expression mechanically, we need a decision procedure for Presburger Arithmetics. We have developed the decision procedure[HKT92] on a Sun SparcStation ELC. For the predicates given in this chapter as examples, satisfiabilities of the predicates can be decided within one second. For even more complex predicates such as the logical combinations of ten integer linear inequalities, their satisfiabilities can be decided within a few seconds in most cases. Therefore, LOTOS/T is enough powerful for practical purposes and suitable for mechanical proof method. We have developed LOTOS interpreter[YHMT92] and a test system for LOTOS with data parameters[HBL$^+$92]. Using these systems, we can construct the LTS from a given LOTOS expression mechanically. Now we have a plan to develop the decision procedure for proving the timed/untimed bisimulation equivalences described in Section 2.3 by using the above tools.

We did not introduce timing-interaction operator defined in [BLT90]. The strength of this is that locality of specification is preserved, as mentioned in [BLT90] (but differs from Timed-Action LOTOS[BLT90] because urgency is still supported in ours). Urgency of interaction can still be expressed in LOTOS/T by hiding the interaction from outside, but urgency of observable interaction cannot be expressed. So expressive power of LOTOS/T is weaker than Timed-Interaction LOTOS and Timed Petri Nets.

Untimed bisimulation equivalence is introduced in order to consider the two processes, which behave the same but in different time constraints (e.g. in different speed), be equivalent. Similar but more advanced investigations are made for CCS in [MT91, AKH92].

Table 2.2: The inference rules of transition relation: Part 2

Parallel	
$\dfrac{B_1 \xrightarrow{\beta} B_1' \quad B_2 \xrightarrow{\beta} B_2'}{B_1 \vert [A] \vert B_2 \xrightarrow{\beta} B_1' \vert [A] \vert B_2'}$ iff $\beta \in A \cup \{\delta\}$ (2.16)	$\dfrac{B_1 \xrightarrow{\textbf{tick}} B_1' \quad B_2 \xrightarrow{\textbf{tick}} B_2'}{B_1 \vert [A] \vert B_2 \xrightarrow{\textbf{tick}} B_1' \vert [A] \vert B_2'}$ (2.17)
$\dfrac{B_1 \xrightarrow{a} B_1'}{B_1 \vert [A] \vert B_2 \xrightarrow{a} B_1' \vert [A] \vert B_2}$ iff $a \notin A \vee a = i$ (2.18)	$\dfrac{B_2 \xrightarrow{a} B_2'}{B_1 \vert [A] \vert B_2 \xrightarrow{a} B_1 \vert [A] \vert B_2'}$ iff $a \notin A \vee a = i$ (2.19)
$\dfrac{B_1 \vert [\emptyset] \vert B_2 \xrightarrow{\alpha} B'}{B_1 \vert\vert\vert B_2 \xrightarrow{\alpha} B'}$ iff $\alpha \in Act \cup \{\delta, \textbf{tick}, i\}$ (2.20)	$\dfrac{B_1 \vert [Act] \vert B_2 \xrightarrow{\alpha} B'}{B_1 \vert\vert B_2 \xrightarrow{\alpha} B'}$ iff $\alpha \in Act \cup \{\delta, \textbf{tick}, i\}$ (2.21)
Disable	
$\dfrac{B_1 \xrightarrow{a} B_1'}{B_1 [> B_2 \xrightarrow{a} B_1' [> B_2}$ (2.22)	$\dfrac{B_2 \xrightarrow{\beta} B_2'}{B_1 [> B_2 \xrightarrow{\beta} B_2'}$ iff $\beta \in Act \cup \{\delta, i\}$ (2.23)
$\dfrac{B_1 \xrightarrow{\delta} B_1'}{B_1 [> B_2 \xrightarrow{\delta} B_1'}$ (2.24)	$\dfrac{B_1 \xrightarrow{\textbf{tick}} B_1' \quad B_2 \xrightarrow{\textbf{tick}} B_2'}{B_1 [> B_2 \xrightarrow{\textbf{tick}} B_1' [> B_2'}$ (2.25)
Enable	
$\dfrac{B_1 \xrightarrow{a} B_1'}{B_1 >> B_2 \xrightarrow{a} B_1' >> B_2}$ (2.26)	$\dfrac{B_1 \xrightarrow{\delta} B_1'}{B_1 >> B_2 \xrightarrow{i} B_2}$ (2.27)
$\dfrac{B_1 \xrightarrow{\textbf{tick}} B_1' \quad B_2 \xrightarrow{\textbf{tick}} B_2' \quad B_1 \not\xrightarrow{\delta}}{B_1 >> B_2 \xrightarrow{\textbf{tick}} B_1' >> B_2'}$ (2.28)	
Hide	
$\dfrac{B \xrightarrow{\beta} B'}{\text{hide } A \text{ in } B \xrightarrow{\beta} \text{hide } A \text{ in } B'}$ iff $\beta \in (Act - A) \cup \{\delta, i\}$ (2.29)	
$\dfrac{B \xrightarrow{a} B'}{\text{hide } A \text{ in } B \xrightarrow{i} \text{hide } A \text{ in } B'}$ iff $a \in A$ (2.30)	$\dfrac{B \xrightarrow{\textbf{tick}} B'}{\text{hide } A \text{ in } B \xrightarrow{\textbf{tick}} \text{hide } A \text{ in } B'}$ (2.31)
"As soon as possible" Execution	
$\dfrac{B \xrightarrow{a} B'}{\text{asap } A \text{ in } B \xrightarrow{a} \text{asap } A \text{ in } B'}$ (2.32)	$\dfrac{B \xrightarrow{\textbf{tick}} B' \quad B \not\xrightarrow{a} \text{ for all } a \in A}{\text{asap } A \text{ in } B \xrightarrow{\textbf{tick}} \text{asap } A \text{ in } B'}$ (2.33)
Process Invocation	
$\dfrac{[\bar{e}/\bar{x}]B\{g_1'/g_1, \dots, g_k'/g_k\} \xrightarrow{\alpha} B'}{P[g_1', \dots, g_k'](\bar{e}) \xrightarrow{\alpha} B'}$ iff	$\alpha \in Act \cup \{\textbf{tick}, \delta, i\}$ and $P[g_1, \dots, g_k](\bar{x}) := B$ is a definition (2.34)

Chapter 3

A Symbolic Approach to the Bisimulation Checking of Real-Time System Specifications

3.1 Introduction

In this chapter, we propose a new model for timed processes, A-TSLTS, and give a method for checking bisimilation equivalence between two A-TSLTS states symbolically. A method for mapping LOTOS/T expressions into A-TSLTSs are also presented.

This chapter is organized as follows. In Section 3.2, the model of timed processes, A-TSLTS, is defined. In Section 3.3, timed bisimulation equivalence of states in an A-TSLTS is defined. In Section 3.4, an algorithm is presented to construct the mgb for two states in an A-TSLTS w.r.t. timed bisimulation equivalence. In Section 3.5, untimed bisimulation equivalence of states in an A-TSLTS is defined and an extension of the algorithm to verify untimed bisimulation equivalence is presented. In Section 3.6, we apply our verification method to the language LOTOS/T defined in Chapter 2. Finally, in Section 3.7, we conclude this chapter.

3.2 A-TSLTS model

A *TSLTS* is an LTS where each state s has a set of parameter variables $DVar(s)$, and each transition is either an action transition, represented as $s \xrightarrow{a,P} s'$ or a delay transition represented as $s \xrightarrow{e(d),P} s'$. a is an action name. d is a variable which stands for a duration. For each delay transition $s \xrightarrow{e(d),P} s'$, $d \notin DVar(s)$ is assumed. Each P is a transition condition. The transition condition P is a formula of a (decidable)

1st-order arithmetic on any (dense or discrete) time domain. P may contain any variable in $DVar(s)$ (s is a source state of the transition). In a delay transition $s \xrightarrow{e(d),P} s'$, P may also contain the variable d.

Intuitively, a delay transition $s \xrightarrow{e(d),P} s'$ represents a state-transition only by delay. Its duration is d and d must satisfy P under a current assignment for the parameter variables in $DVar(s)$. The delay is possible up to the maximum value of d's which satisfy P. The delay over the maximum value of d is not allowed (time-deadlock[MT90],urgency[BL92]). When the delay transition is completed, the actual duration (which satisfies P) is assigned to the variable d. $DVar(s')$ may contain the variable d. So the value of d may be used in conditions of any succeeding transitions. An action transition $s \xrightarrow{a,P} s'$ represents an execution of an action a when P holds under a current assignment for parameter variables in $DVar(s)$. The execution of an action is considered instantaneous, since we take interleaving semantics to express concurrency[Wan91, Che92]. The state s may have multiple outgoing action transitions. In that case, one of executable action transitions is nondeterministically chosen and then executed.

Example 3.1 We show an example of a TSLTS in Fig. 3.1. In Fig. 3.1, for convenience, the names $s_1, s_2, \ldots$ are assigned to states and $t_1, t_s, \ldots$ for transitions. The set associated with each state s_i represents $DVar(s_i)$. $a[P]$ (or $e(d)[P]$) associated with each transition represents an action name a (or a delay with its duration of d, respectively) with a transition condition P. When a value v is assigned to the parameter variable x at state s_1, the TSLTS in Fig. 3.1 behaves as follows. First, $x = v$ units of time have elapsed (the value v is assigned to d_{t_1}) and then the action a is executed. Next, before 4 units of time have elapsed, the action b or c is executed. The action b is executable when the duration is within 3 units of time. The action c is executable when the duration is more than or equal to 2 units of time. In the case c is executed, the TSLTS moves its state to s_1 and then repeats the behaviour from the beginning. $\Box$

In the TSLTS model, it is possible that a sequence of multiple consecutive delay transitions is equivalent to one delay transition. This fact makes it difficult to consider bisimulation without concrete values (symbolic bisimulation[HL95]). Thus, in order not to execute two consecutive delay transitions, we restrict a TSLTS so that its states fall into two categories of states, idle states and active states. Each idle state has only a delay transition as an outgoing transition and the destination is an active state. An active state has only action transitions as outgoing transitions and all the destinations are idle states. We call this restricted TSLTS as an *Alternating TSLTS (A-TSLTS)*. The notion of A-TSLTS is inspired by [Han91].

In the rest of this chapter, we assume that each TSLTS is a finite A-TSLTS, and it is time-deterministic, i.e., every state has at most one outgoing delay transition.

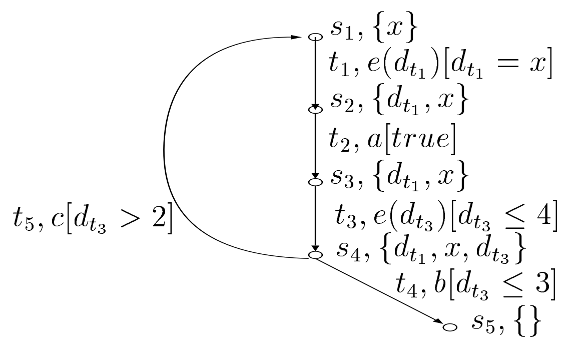

Figure 3.1: An example of TSLTS

Time-determinacy is a reasonable assumption when we consider processes of real-world. Many other studies also assume time-determinacy[MT90, Wan91, Che92].

Example 3.2 The TSLTS of Example 3.1 is an A-TSLTS because a division into $\{s_1, s_3, s_5\}$ (idle states) and $\{s_2, s_4\}$ (active states) is possible. It is also time-deterministic.

3.3 Timed Bisimulation Equivalence — case for arbitrary time domain

In this section, we define timed bisimulation equivalence for A-TSLTSs. The definition here is different from that of Section 2.3, because we consider arbitrary time domains. Before all, we need some preliminary definitions.

Definition 3.1

- We denote *assignments* of values to variables by $\rho, \rho', \ldots$.
- For a predicate P and an assignment ρ, we denote $\rho \models P$ iff P is true under an assignment ρ.
- We denote $\rho[x = e]$ the same assignment as ρ except that the value of the expression e is assigned to the variable x.
- We denote a tuple (s, ρ) of a state s in a TSLTS and an assignment ρ, as $\rho(s)$. $\rho(s)$ stands for a state with some parameter *values*(not variables) associated with s. We call it an *instance* of s w.r.t. ρ. □

The actual moves of a TSLTS are formally defined by considering the corresponding (traditional) LTS, whose states are all instances of TSLTS states, and whose transitions are labelled by either an action name or a concrete value of a duration.

Definition 3.2 For a TSLTS M, its corresponding *semantic LTS* M' is defined as follows:

- The set of states in M' is the set of all instances of M, i.e. $\{\rho(s) | \rho\text{:an assignment}, s\text{:a state of } M\}$.
- Each transition in M' is labelled by either an action name a of M, or any non-negative time value t.
- For each transition $s \xrightarrow{a,P} s'$ in M and each assignment ρ, M' has a transition $\rho(s) \xrightarrow{a} \rho(s')$ iff $\rho \models P$.
- For each transition $s \xrightarrow{e(d),P} s'$ in M, each assignment ρ, and any non-negative time value t, M' has a transition $\rho(s) \xrightarrow{t} \rho[d = t](s')$ iff $\rho[d = t] \models \exists d'[d \leq d' \wedge P\{d'/d\}]$ ($P\{d'/d\}$ denotes P whose any occurrence of a free variable d is replaced by d'). Moreover, for any non-negative time value t' which satisfies $t' \leq t$, M' has a transition $\rho[d = t'](s') \xrightarrow{t-t'} \rho[d = t](s')$. $\square$

Remark: Note that the semantic LTS defined above may not be alternating even if the corresponding TSLTS is alternating. It may contain consecutive delay transitions associated with concrete time values which satisfy time associativity, that is, $s \xrightarrow{t_1} s_1 \xrightarrow{t_2} s_2$ implies $s \xrightarrow{t_1+t_2} s_2$ (the fourth condition in Definition 3.2 ensures this property). It may also contain infinite delay transitions.

The method for modeling real-time processes by considering a delay transition with an associated time value is similar to [Wan91, HLW91, Che92].

For a given TSLTS, timed bisimulation equivalence of its two instances of states is defined by considering a traditional bisimulation equivalence on its semantic LTS.

Definition 3.3 A *timed bisimulation relation* R is a binary relation on a set of instances of TSLTS states $\{\rho(s) | s\text{:a TSLTS state}, \rho\text{:an assignment}\}$, which satisfies all of the following conditions:

- If $(\rho_i(s_i), \rho_j(s_j)) \in R$, then all of the following conditions hold:
  - For any time value t, if $\rho_i(s_i) \xrightarrow{t} \rho'_i(s'_i)$, then there exist some s'_j and ρ'_j such that $\rho_j(s_j) \xrightarrow{t} \rho'_j(s'_j)$ and $(\rho'_i(s'_i), \rho'_j(s'_j)) \in R$,
  - For any action name a in the TSLTS, if $\rho_i(s_i) \xrightarrow{a} \rho'_i(s'_i)$, then there exist some s'_j and ρ'_j such that $\rho_j(s_j) \xrightarrow{a} \rho'_j(s'_j)$ and $(\rho'_i(s'_i), \rho'_j(s'_j)) \in R$.
- R is a symmetric relation.

If there exists some timed bisimulation relation R such that $(\rho_i(s_i), \rho_j(s_j)) \in R$, the two instances $\rho_i(s_i)$ and $\rho_j(s_j)$ are called *timed bisimulation equivalent*, which is denoted by $\rho_i(s_i) \sim_t \rho_j(s_j)$. Especially, if $\rho(s_i) \sim_t \rho(s_j)$, then the two states s_i and s_j are called *timed bisimulation equivalent w.r.t. an assignment ρ.* □

Remark: The definition of timed bisimulation above is equivalent to the traditional definition of bisimulation, because of the condition that R must be symmetric relation. We use this alternative definition in order to make the definition more compact.

3.4 Verification of Timed Bisimulation Equivalence

For any state-pair (s_i, s_j) in an A-TSLTS, we call the weakest condition P such that if $\rho \models P$ then s_i and s_j are timed bisimulation equivalent w.r.t. ρ, as the *mgb* of (s_i, s_j). If we can obtain the mgb P for any state-pair (s_i, s_j), then the verification of timed bisimulation equivalence of $\rho(s_i)$ and $\rho(s_j)$ is reduced to the verification to check whether $\rho \models P$.

To keep track of the correspondences between variables during matching, it is useful to replace some different variables of two states with some common name, standing for their matched common value which equates the two states. In order to do so, we consider the mgb for a pair of *terms* instead of states in A-TSLTS. This is similar to [HL95]. A *term* is a tuple of a state and a *substitution*. A *substitution* is a mapping from variables to variables. We denote a term (s, σ) as $s\sigma$, where s is a state of A-TSLTS and σ is a substitution. We also denote a substitution which maps the variable d to d' as $[d \rightarrow d']$. If σ is an identity substitution, we abbreviate $s\sigma$ to s and we do not distinguish between the term $s\sigma$ and the state s. Note that if the set of variables is a finite set, then the set of all possible substitutions are finite. A transition between terms is defined as $s\sigma \xrightarrow{e(\sigma(d)), P\sigma} s'\sigma$ ($s\sigma \xrightarrow{a, P\sigma} s'\sigma$) iff $s \xrightarrow{e(d), P} s'$ ($s \xrightarrow{a, P} s'$, respectively) in an A-TSLTS. We denote the mgb of a term-pair (s_i, s_j) as $mgb(s_i, s_j)$. If the A-TSLTS has only finite states and finite variables, $mgb(s_i, s_j)$ is obtained by the algorithm in Fig. 3.2.

Note that the algorithm in Fig. 3.2 is just a fragment of [HL95]. Only the difference is that we consider delay transitions instead of input transitions. The topology of the semantic LTS of an A-TSLTS is quite different from that of symbolic transition graph in [HL95]: infinite concrete delay transitions are sequentially connected in timed case, while infinite concrete data transitions are branching from the same state in data case. However, to check bisimulation, we are only interested in (1). what amount of delay/data transitions are possible, and (2). whether the branching structures are equivalent after the transitions of the same amount of delay/data

```
$mgb(s_i, s_j) \overset{def}{=} mgb1(s_i, s_j, \emptyset)$
$mgb1(s_i, s_j, W) \overset{def}{=}$ if $(s_i, s_j) \in W$ then return true
                    else if $(s_i, s_j)$ is a pair of idle states,
                        then return $match\_delay(s_i, s_j, W)$
                    else if $(s_i, s_j)$ is a pair of active states,
                        then return $match\_action(s_i, s_j, W)$
                    else return false
$match\_delay(s_i, s_j, W) \overset{def}{=}$ if $s_i \xrightarrow{e(d_i),P_i} s_{i'}$ and $s_j \xrightarrow{e(d_j),P_j} s_{j'}$
        then let $\{d = new(DVar(s_i) \cup DVar(s_j)),$
                    $M_{i',j'} = mgb1(s_{i'}[d_i \to d], s_{j'}[d_j \to d], W \cup \{(s_i, s_j)\})\}$ in
                return $\forall d[P_i\{d/d_i\} \Rightarrow [P_j\{d/d_j\} \wedge M_{i',j'}]]$
                    $\wedge \forall d[P_j\{d/d_j\} \Rightarrow [P_i\{d/d_i\} \wedge M_{i',j'}]]$
        else if $s_i \overset{e(d_i),P_i}{\not\longrightarrow}$ and $s_j \overset{e(d_j),P_j}{\not\longrightarrow}$ then return true else return false
$match\_action(s_i, s_j, W) \overset{def}{=}$ return $\bigwedge_{a \in Act}\{match\_action1(a, s_i, s_j, W)\}$
$match\_action1(a, s_i, s_j, W) \overset{def}{=}$ let $\{K = \{k | s_i \xrightarrow{a,P_k} s_{i_k}\}$, $L = \{l | s_j \xrightarrow{a,Q_l} s_{j_l}\}$,
                            $M_{k,l} = mgb1(s_{i_k}, s_{j_l}, W \cup \{(s_i, s_j)\})\}$ in
            return $\bigwedge_{k \in K}\{P_k \Rightarrow \bigvee_{l \in L}\{Q_l \wedge M_{k,l}\}\} \wedge$
                $\bigwedge_{l \in L}\{Q_l \Rightarrow \bigvee_{k \in K}\{P_k \wedge M_{k,l}\}\}$
```

where, for a set V of variables, $new(V)$ denotes a function which returns an appropriate new variable x such that $x \notin V$. Moreover, $s_i[d_i \to d]$ represents that every occurrence of the variable d_i in the transition conditions of any s_i's outgoing transitions and further succeeding transitions, is replaced by d.

Figure 3.2: The algorithm to calculate $mgb(s_i, s_j)$.

transitions. These are not dependent on the topology of the semantic LTS. Therefore, we can simply reduce the problem to [HL95]. [1] In the following, we explain the algorithm just for readers' convenience.

[1]To be exact, in contrast to A-TSLTS, the Hennessy-Lin's model, symbolic transition graph[HL95] cannot have an input transition whose possible range of input values is limited by its transition condition. However, without any problems, their result can be easily extended to the model which have an input transition which has a limited range of input values. This is because the form of the mgb is like "$\forall x[P \Rightarrow (Q \wedge \ldots)] \wedge \forall x[Q \Rightarrow (P \wedge \ldots)]$" and whether the input variable x has a range limitation is expressed by whether the transition conditions P and Q have x's as free variables. To make P and Q have x as free variables does not affect the fact that the entire expression is the mgb. Therefore, their result is extended and our problem is reduced to the extended result.

The function $mgb(s_i, s_j)$ takes two arguments s_i and s_j, any two states in an A-TSLTS, and returns the mgb for (s_i, s_j). The function $mgb1(s_i, s_j, W)$ takes three arguments s_i, s_j and a set W of state-pairs. W is a set of *already visited* pairs, introduced to make sure the algorithm eventually terminates. For $(s_i, s_j) \in W$, it simply returns *true*. Otherwise, it returns $match\_delay(s_i, s_j, W)$ if (s_i, s_j) is a pair of idle states, or $match\_action(s_i, s_j, W)$ if (s_i, s_j) is a pair of active states. $match\_delay(s_i, s_j, W)$ ($match\_action(s_i, s_j, W)$) is a function which recursively calculates the mgb for (s_i, s_j), where we assume (s_i, s_j) is a pair of idle (active, respectively) states.

The function $match\_delay(s_i, s_j, W)$ calculates the mgb for two idle states s_i and s_j as follows. Firstly, from the definition of A-TSLTS and time-determinacy, delay transitions from s_i and s_j correspond to one-to-one, including duration values. So we unify the delay variables in the two transitions into one. We introduce a new variable d representing the common duration of delay. We choose $d = new(DVar(s_i) \cup DVar(s_j))$. W.r.t. a given assignment ρ, if s_i and s_j are timed bisimulation equivalent, and if any delay transition of duration v from s_i is possible, then there must exist a delay transition of the same duration v from s_j, and the destinations s'_i and s'_j must be timed bisimulation equivalent w.r.t. $\rho[d = v]$. For example, if $s_i \overset{e(d_i), d_i \le x}{\longrightarrow} s'_i$ and $s_j \overset{e(d_j), d_j \le y}{\longrightarrow} s_j$, then $\forall d[d \le x \Rightarrow [d \le y \wedge (\text{the mgb for } (s'_i[d_i \to d], s'_j[d_j \to d]))]$ holds. Here, in general, the mgb for (s'_i, s'_j) contains the variables d_i or d_j. To preserve the information that d_i and d_j are equal, we consider the mgb for $(s'_i[d_i \to d], s'_j[d_j \to d])$ instead. In general, the mgb for $(s'_i[d_i \to d], s'_j[d_j \to d])$ contains the variable d as a free variable. It represents the mgb for (s'_i, s'_j) in the case $d_i = d_j = d$ is assumed.

The above discussion must apply when s_i and s_j are exchanged. Therefore, ρ must satisfy the following condition if s_i and s_j are timed bisimulation equivalent w.r.t. ρ.

$$\forall d[P_i\{d/d_i\} \Rightarrow [P_j\{d/d_j\} \wedge M_{i',j'}]] \wedge \forall d[P_j\{d/d_j\} \Rightarrow [P_i\{d/d_i\} \wedge M_{i',j'}]]. \tag{3.1}$$

where $M_{i',j'} = mgb(s'_i[d_i \to d], s'_j[d_j \to d])$. On the other hand, if s_i and s_j are not timed bisimulation equivalent w.r.t. ρ, then, for example, a delay transition of some duration v' is possible from s_i, which is impossible on s_j, or otherwise s'_i and s'_j are not equivalent w.r.t. $\rho[d = v'']$ for some value v''. In any case, Expression (3.1) does not hold. Therefore, Expression (3.1) is the weakest condition such that ρ must satisfy in order to make $\rho(s_i)$ and $\rho(s_j)$ be timed bisimulation equivalent, i.e., the mgb for (s_i, s_j). The function $match\_delay(s_i, s_j, W)$ calculates $M_{i',j'} = mgb1(s'_i[d_i \to d], s'_j[d_j \to d], W \cup \{(s_i, s_j)\})$ recursively (where (s_i, s_j) is treated as an *already visited* pair) and then returns Expression (3.1) as the mgb for (s_i, s_j).

The function $match\_action(s_i, s_j, W)$ returns the mgb for active states s_i and s_j, which is calculated as follows. Firstly, if s_i and s_j are timed bisimulation equivalent w.r.t. an assignment ρ, for any action a in a set *Act* of all actions, the following condition holds. For any possible transition $s_i \xrightarrow{a,P_k} s_{i_k}$ whose transition condition P_k satisfies $\rho \models P_k$, if the action a is executable, there must exist some transition $s_j \xrightarrow{a,Q_l} s_{j_l}$ whose transition condition Q_l also satisfies $\rho \models Q_l$ and the destinations s_{i_k} and s_{j_l} must be timed bisimulation equivalent w.r.t. ρ (ρ must satisfy the mgb for (s_{i_k}, s_{j_l})). The above discussion must be true when s_i and s_j are exchanged. Therefore, when we let $K = \{k | s_i \xrightarrow{a,P_k} s_{i_k}\}$, $L = \{l | s_j \xrightarrow{a,Q_l} s_{j_l}\}$ and $M_{k,l} = mgb(s_{i_k}, s_{j_l})$, ρ must satisfy

$$\bigwedge_{k \in K} \{P_k \Rightarrow \bigvee_{l \in L} \{Q_l \wedge M_{k,l}\}\} \wedge \bigwedge_{l \in L} \{Q_l \Rightarrow \bigvee_{k \in K} \{P_k \wedge M_{k,l}\}\}. \tag{3.2}$$

A conjunction of Expression (3.2) over all actions $a \in Act$ is a condition that ρ must satisfy if s_i and s_j are timed bisimulation equivalent for any action w.r.t. ρ. On the other hand, if ρ does not make s_i and s_j be timed bisimulation equivalent, there must exist some action a' such that, for example, $s_i \xrightarrow{a',P_k} s_{i_k}$ is executable and for any l, either $s_j \xrightarrow{a',Q_l} s_{j_l}$ is not executable or s_{i_k} and s_{j_l} are not timed bisimulation equivalent w.r.t. ρ. In any case, Expression (3.2) does not hold. Therefore, a conjunction of Expression (3.2) over all actions $a \in Act$ is the weakest condition that ρ must satisfy to make s_i and s_j be timed bisimulation equivalent w.r.t. ρ, i.e. the mgb for (s_i, s_j). The function $match\_action1(a, s_i, s_j, W)$ calculates each M_{i_k,j_l} recursively (with (s_i, s_j) as an already visited pair), and then returns Expression (3.2). The function $match\_action(s_i, s_j, W)$ composes a conjunction of $match\_action1(a, s_i, s_j, W)$ over all $a \in Act$ and returns it as the mgb for (s_i, s_j).

The algorithm $mgb(s_i, s_j)$ terminates if the considered A-TSLTS has only a finite number of states and variables (thus it has only a finite number of pair of terms).

Formally, we obtain the correctness result by the following theorem.

Theorem 3.1 $\rho(s_i) \sim_t \rho(s_j)$ if and only if $\rho \models mgb(s_i, s_j)$. □

The complexity of our algorithm is estimated as follows. Our algorithm terminates when all pairs of terms in A-TSLTS are visited. Let n_t, n_v and n_s be the number of terms, variables and states of A-TSLTS. Then $n_t = n_s \times {n_v}^{n_v}$ holds because the number of all substitutions over variables are ${n_v}^{n_v}$. So the number of all pairs of terms are ${n_t}^2 = {n_s}^2 \times {n_v}^{2n_v}$. Therefore we obtain the following theorem:

Theorem 3.2 The time complexity of $mgb(s_i, s_j)$ is $O(n_s^2 \times 2^{cn_v \log n_v})$. □

Example 3.3 For a pair (s_1, s_3) of the A-TSLTS in Fig. 3.3, $mgb(s_1, s_3)$ is obtained as follows.

$$\begin{aligned} mgb(s_1, s_3) &= \forall d_1[d_1 = x \Rightarrow [d_1 = y \wedge M_{24}]] \\ &\quad \wedge \forall d_1[d_1 = y \Rightarrow [d_1 = x \wedge M_{24}]] \end{aligned}$$

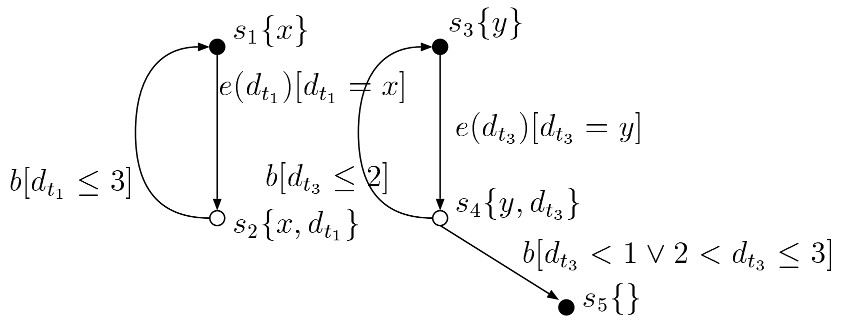

Figure 3.3: An example of A-TSLTS

where,

$$
\begin{aligned}
M_{24} &= mgb1(s_2[d_{t_1} \to d_1], s_4[d_{t_3} \to d_1], \{(s_1, s_3)\}) \\
&= [d_1 \leq 3 \Rightarrow [d_1 \leq 2 \wedge M_{13} \\
&\qquad \vee(d_1 < 1 \vee 2 < d_1 \leq 3) \wedge M_{15}]] \wedge \\
&\quad\ [d_1 \leq 2 \Rightarrow [d_1 \leq 3 \wedge M_{13}]] \wedge \\
&\quad\ [(d_1 < 1 \vee 2 < d_1 \leq 3) \Rightarrow [d_1 \leq 3 \wedge M_{15}]], \\
M_{13} &= mgb1(s_1, s_3, \{(s_1, s_3), (s_2, s_4)\}), \\
M_{15} &= mgb1(s_1, s_5, \{(s_1, s_3), (s_2, s_4)\}) = false.
\end{aligned}
$$

Since $mgb1(s_1, s_3, \{(s_1, s_3), (s_2, s_4)\}) = true$, the mgb after simplification is
$mgb(s_1, s_3) \equiv [x = y] \wedge [1 \leq x \leq 2 \vee x > 3]$. □

Remark: In the mgb, all duration variables are universally quantified, that is, they do not appear as free variables but as bound variables. In this example, they can be eliminated by simplifying the mgb by hand.

3.5 Untimed Bisimulation Equivalence and its Verification

For verification of time-constrained systems, it is also useful to verify whether its possible action sequences and their executability are changed when timing of each action is modified. Thus, in this section we extend the result of the previous section to the verification of untimed bisimulation equivalence, a bisimulation equivalence where the timing does not have to be exactly equal.

At first, we define untimed bisimulation equivalence precisely.

Definition 3.4 An *untimed bisimulation relation* R is a binary relation on a set of instances of TSLTS states $\{\rho(s)|s\text{:a TSLTS state}, \rho\text{:an assignment}\}$, which satisfies the following conditions:

- R is a symmetric relation, and
- if $(\rho_i(s_i), \rho_j(s_j)) \in R$, then all of the following conditions hold:
  - For any time value t, if $\rho_i(s_i) \xrightarrow{t} \rho'_i(s'_i)$, then there exist some s'_j, ρ'_j and some time value t' such that $\rho_j(s_j) \xrightarrow{t'} \rho'_j(s'_j)$ and $(\rho'_i(s'_i), \rho'_j(s'_j)) \in R$,
  - For any action name a in the TSLTS, if $\rho_i(s_i) \overset{a}{\Longrightarrow} \rho'_i(s'_i)$, then there exist some s'_j and ρ'_j such that $\rho_j(s_j) \overset{a}{\Longrightarrow} \rho'_j(s'_j)$ and $(\rho'_i(s'_i), \rho'_j(s'_j)) \in R$. Here $\overset{a}{\Longrightarrow} \overset{def}{=} \xrightarrow{t_1}\xrightarrow{a}\xrightarrow{t_2}$ for some time values t_1 and t_2.

If there exists some untimed bisimulation equivalence R such that $(\rho_i(s_i), \rho_j(s_j)) \in R$, the two instances $\rho_i(s_i)$ and $\rho_j(s_j)$ are called *untimed bisimulation equivalent*, which is denoted by $\rho_i(s_i) \sim_u \rho_j(s_j)$. $\square$

The result of the previous section can be extended to untimed bisimulation equivalence. To do this, we have only to modify functions *match_delay*() and *match_action*() to make durations not necessarily be equal when we match delay transitions.

The mgb of idle states is easily expressed by the following formula:

$$\forall d[P_i\{d/d_i\} \Rightarrow \exists d'[P_j\{d'/d_j\} \wedge M_{i',j'}]] \wedge \forall d'[P_j\{d'/d_j\} \Rightarrow \exists d[P_i\{d/d_i\} \wedge M_{i',j'}]]$$

where $M_{i',j'}$ is the mgb of the next pair of active states.

On the other hand, in order to consider the mgb of the active states for untimed bisimulation equivalence, we must solve the following problem. For the timed bisimulation equivalence, we have only to consider the executable actions at the specified time instant (for example, the action a is executable at time 2, the action b is executable at time 3, ...). However, it is not the case for the untimed bisimulation equivalence. Consider the two A-TSLTSs in Fig. 3.4. If we consider the executability of actions at time d only, the states s_1 and s_3 should be untimed equivalent, because for duration $d_1 = 2$ after which only a is executable, there exists a duration $d_2 = 3$ after which only a is also executable, and vice versa. However, for the above example, s_1 and s_3 are not untimed equivalent in the sense of Definition 3.4, because after the delay of 2.5 units of time, s_1 is in the state such that only b is executable (after more 0.5 units of time elapsed), whereas s_3 is in the state such that only a is executable. So, instead of the executability at the given time instant, we consider the executability at some time after the given time instant. For the above example, when the system is at state s_1 and 2 units of time have elapsed, a is executable

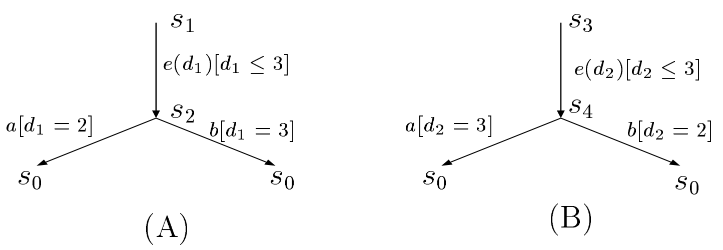

Figure 3.4: Example of A-TSLTS where s_1 and s_3 are not untimed bisimulation equivalent.

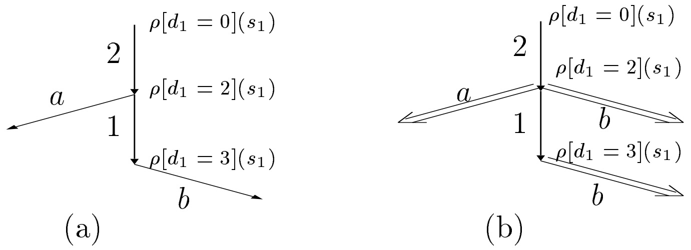

Figure 3.5: Illustration of (a) timed semantics and (b) untimed semantics of Fig. 3.4-(A).

now and b is executable after more $3 - 2 = 1$ unit of time elapses. In this case, s_1 is in the state such that both a and b are executable at some time in the future (see Fig. 3.5-(b)). In general, when d units of time have elapsed and a is executable after more $d' - d$ units of time elapse, i.e., d' satisfies both $d \leq d'$ and the transition condition of a, a is executable at some time in the future.

Because of the reasons above, we must loose the executability condition of actions in order to define the mgb of the untimed bisimulation equivalence. So we define that for a given duration d, an action is executable if and only if there exists some duration d' such that $d \leq d'$ and d' satisfies the transition condition of the action. Note that d', as well as d, must also satisfy the transition condition of the delay transition. Formally, let P denote the transition condition of an action a. Then we say that the action a is *untimedly executable after duration* d, if and only if $\exists d'[d \leq d' \wedge P\{d'/d\}]$. We refer to the condition as the *untimed transition condition*. Since we frequently consider the predicate of the form $\exists d'[d \leq d' \wedge P\{d'/d\}]$ w.r.t. P and the variable d, we abbreviate it as $\mathcal{F}_d P$. Note that if the transition condition of the most recent delay transition is D, then d ranges over the solutions of D. However, $\mathcal{F}_d P$ may have a solution d' which does not satisfy D, which is incorrect. (Consider the untimed transition condition of b in the sequence $s_0 \overset{e(d),d\leq 2}{\longrightarrow} s_1 \overset{b,d=3}{\longrightarrow} s_2$.) In this case, the untimed transition condition becomes $\mathcal{F}_d\{P \wedge D\}$.

Using the untimed transition conditions, the mgb of the active states (s_i, s_j) for untimed case is given as follows. Firstly, if s_i and s_j are untimed bisimulation equivalent w.r.t. an assignment ρ, for any action a in a set Act of all actions, the following condition holds. Suppose that the most recent delay transitions of s_i and s_j are $s_{i_0} \xrightarrow{e(d_i),D_i} s_i$ for some s_{i_0}, and $s_{j_0} \xrightarrow{e(d_j),D_j} s_j$ for some s_{j_0}, respectively. Note that the delay variable d_i (d_j) ranges over solutions of the predicate D_i $(D_j$, respectively). For any possible transition $s_i \xrightarrow{a,P_k} s_{i_k}$ whose untimed transition condition $\mathcal{F}_{d_i}[P_k \wedge D_i]$ satisfies $\rho \models \mathcal{F}_{d_i}[P_k \wedge D_i]$, if the action a is untimedly executable, there must exist some transition $s_j \xrightarrow{a,Q_l} s_{j_l}$ whose untimed transition condition $\mathcal{F}_{d_j}[Q_l \wedge D_j]$ also satisfies $\rho \models \mathcal{F}_{d_j}[Q_l \wedge D_j]$ and the destinations s_{i_k} and s_{j_l} must be untimed bisimulation equivalent w.r.t. $\rho[d_i \to d'_i, d_j \to d'_j]$ ($\rho[d_i \to d'_i, d_j \to d'_j]$ must satisfy the mgb for (s_{i_k}, s_{j_l})). Here $\rho[d_i \to d'_i, d_j \to d'_j]$ denotes the same assignment as ρ except the names of variables d_i and d_j are replaced with d'_i and d'_j, respectively. Note that since it is assumed that a is untimedly executed, the executed time of a at the state s_i is not d_i but d'_i. So the destinations s_{i_k} and s_{j_l} can be reached with the values of not d_i and d_j but d'_i and d'_j. That is why s_{i_k} and s_{j_l} must be untimed equivalent w.r.t. $\rho[d_i \to d'_i, d_j \to d'_j]$. The above discussions must be true when s_i and s_j are exchanged. Therefore, similar to the timed case, we obtain the mgb of active states s_i and s_j for untimed bisimulation equivalence as:

$$\begin{array}{l}\bigwedge_{k\in K}\{\mathcal{F}_{d_i}[P_k \wedge D_i \Rightarrow \bigvee_{l\in L}\{\mathcal{F}_{d_j}[Q_l \wedge D_j \wedge M_{k,l}]\}]\}\\ \wedge \bigwedge_{l\in L}\{\mathcal{F}_{d_j}[Q_l \wedge D_j \Rightarrow \bigvee_{k\in K}\{\mathcal{F}_{d_i}[P_k \wedge D_i \wedge M_{k,l}]\}]\}\end{array}$$

where, $K = \{k | s_i \xrightarrow{a,P_k} s_{i_k}\}$, $L = \{l | s_j \xrightarrow{a,Q_l} s_{j_l}\}$, $M_{k,l} = mgb(s_{i_k}, s_{j_l})$, $s_{i_0} \xrightarrow{e(d_i),D_i} s_i$ for some s_{i_0}, and $s_{j_0} \xrightarrow{e(d_j),D_j} s_j$ for some s_{j_0}.

In summary, the mgb for untimed bisimulation equivalence can be obtained by modifying a part of the algorithm in Section 3.4 as Fig. 3.6.

Example 3.4 Consider the two A-TSLTSs in Fig 3.4. The mgb of (s_1, s_3) for the untimed bisimulation equivalence can be obtained as follows:

$$mgb(s_1, s_3) \;=\; \forall d_1[d_1 \leq 3 \Rightarrow \exists d_2[d_2 \leq 3 \wedge M_{24}]] \wedge \forall d_2[d_2 \leq 3 \Rightarrow \exists d_1[d_1 \leq 3 \wedge M_{24}]],$$

where,

$$\begin{array}{rcl} M_{24} & = & \exists d'_1[d_1 \leq d'_1 \wedge d'_1 \leq 3 \wedge d'_1 = 2 \Rightarrow \exists d'_2[d_2 \leq d'_2 \wedge d'_2 \leq 3 \wedge d'_2 = 3 \wedge M_{00}]]\\ & & \wedge \exists d'_2[d_2 \leq d'_2 \wedge d'_2 \leq 3 \wedge d'_2 = 3 \Rightarrow \exists d'_1[d_1 \leq d'_1 \wedge d'_1 \leq 3 \wedge d'_1 = 2 \wedge M_{00}]]\\ & & \wedge \exists d'_1[d_1 \leq d'_1 \wedge d'_1 \leq 3 \wedge d'_1 = 3 \Rightarrow \exists d'_2[d_2 \leq d'_2 \wedge d'_2 \leq 3 \wedge d'_2 = 2 \wedge M_{00}]]\\ & & \wedge \exists d'_2[d_2 \leq d'_2 \wedge d'_2 \leq 3 \wedge d'_2 = 2 \Rightarrow \exists d'_1[d_1 \leq d'_1 \wedge d'_1 \leq 3 \wedge d'_1 = 3 \wedge M_{00}]],\\ M_{00} & = & true. \end{array}$$

```
$match\_delay(s_i, s_j, W) \overset{def}{=}$ if $s_i \xrightarrow{e(d_i),P_i} s_{i'}$ and $s_j \xrightarrow{e(d_j),P_j} s_{j'}$
    then let $d = new(DVar(s_i) \cup DVar(s_j))$,
             $d' = new(DVar(s_i) \cup DVar(s_j) \cup \{d\})$,
             $M_{i',j'} = match\_action(s_{i'}[d_i \to d], s_{j'}[d_j \to d'], W \cup \{(s_i, s_j)\},$
                                  $d, P_i\{d/d_i\}, d', P_j\{d'/d_j\})$ in
         return $\forall d[P_i\{d/d_i\} \Rightarrow \exists d'[P_j\{d'/d_j\} \wedge M_{i',j'}]]$
                $\wedge \forall d'[P_j\{d'/d_j\} \Rightarrow \exists d[P_i\{d/d_i\} \wedge M_{i',j'}]]$
    else if $s_i \overset{e(d_i),P_i}{\not\longrightarrow}$ and $s_j \overset{e(d_j),P_j}{\not\longrightarrow}$ then return true else return false
$match\_action(s_i, s_j, W, d_i, D_i, d_j, D_j) \overset{def}{=}$
    return $\bigwedge_{a \in Act}\{match\_action1(a, s_i, s_j, W, d_i, D_i, d_j, D_j)\}$
$match\_action1(a, s_i, s_j, W, d_i, D_i, d_j, D_j) \overset{def}{=}$
         let $\{K = \{k | s_i \xrightarrow{a,P_k} s_{i_k}\}$, $L = \{l | s_j \xrightarrow{a,Q_l} s_{j_l}\}$,
             $M_{k,l} = mgb1(s_{i_k}, s_{j_l}, W \cup \{(s_i, s_j)\})\}$ in
         return $\bigwedge_{k \in K}\{\mathcal{F}_{d_i}[P_k \wedge D_i \Rightarrow \bigvee_{l \in L}\{\mathcal{F}_{d_j}[Q_l \wedge D_j \wedge M_{k,l}]\}]\}$
             $\wedge \bigwedge_{l \in L}\{\mathcal{F}_{d_j}[Q_l \wedge D_j \Rightarrow \bigvee_{k \in K}\{\mathcal{F}_{d_i}[P_k \wedge D_i \wedge M_{k,l}]\}]\}$

where $\mathcal{F}_d P \overset{def}{=} \exists d'[d \leq d' \wedge P\{d'/d\}]$.
```

Figure 3.6: Definition of $match\_delay()$ and $match\_action()$ for untimed bisimulation equivalence

After simplifying the above formula, we obtain $M_{24} \equiv (d_1 \leq 2 \wedge d_2 \leq 2) \vee (d_1 > 3 \wedge d_2 > 3)$. So we get $mgb(s_1, s_3) \equiv false$, that is, s_1 and s_3 are not untimed bisimulation equivalent. □

3.6 Mapping LOTOS/T into A-TSLTS

In Chapter 2, a structured operational semantics of LOTOS/T expressions on a discrete (integer) time domain is defined. Our intention is to define another structured operational semantics of LOTOS/T which maps a LOTOS/T expression to an A-TSLTS. In the latter semantics, it does not matter which time domain we choose. To achieve this, firstly we define each state of the obtained A-TSLTS corresponds to an expression of LOTOS/T with a *mark* 'i' or 'a', indicating which category of states (idle or active) the state itself resides now. Secondly, for each idle state (B, i), where B is a LOTOS/T expression, we define an idle transition starting with (B, i) by inference rules. Finally, for each active state (B, a), we define an active transition starting with (B, a), and a complete inference system which derives A-TSLTS's from LOTOS/T expressions is given. Please note that we simply define $DVar((B, i))$ (or

$DVar((B,a)))$ as $DVar(B)$, where $DVar(B)$ represents the set of all defined (free) variables in B. Informally, a defined variable means the variable whose value has been already determined by previous execution of actions. For example, w.r.t. the behaviour expression $a[x=t]; b[t \le x+3]; \textbf{stop}$, the variable x in the subexpression $b[t \le x+3]; \textbf{stop}$ is a defined variable because the executed time of a has been assigned to x. On the other hand, x in $a[x=t]; b[t \le x+3]; \textbf{stop}$ is not a defined variable, since the value of the variable x has not been assigned at this moment. The formal definition of defined variables appears in [NHT94]. In the rest of the chapter, we assume $DVar(B)$ is a set of defined variables of the subexpression B w.r.t. the entire behaviour expression. Since it is always obvious which expression we assume as the entire behaviour expression (we always assume it is the behaviour expression of the initial state), we simply refer to the defined variables of the expression B as $DVar(B)$.

3.6.1 Delay Transitions of LOTOS/T

Basically, a delay transition from an idle state (B, i) is defined as follows.

- A new delay variable d, which is not used by the behaviour expression B, is introduced to represent the duration of the delay transition.
- The transition condition is defined so that it exactly expresses the possible range of delay of the behaviour expression B.
- The destination state (B', a) of the transition is defined so that it represents the behaviour after d units of time has elapsed. The behaviour expression B' may contain the variable d because the following behaviour might depend on how much time elapsed on this delay transition.

For example, consider a behaviour expression $B = a[2 \le t \le 3 \wedge x_0 = t]; b[t = x_0 + 3]; c[t = x_0 + 4]; \textbf{stop}$. From the definition of LOTOS/T, up to 3 units of time of delay are possible from the idle state (B, i). A delay variable d is introduced to represent the duration. Then, the transition condition of the outgoing delay transition of (B, i) is defined as '$d \le 3$'. To consider the state where d units of time have elapsed, every occurrence of t in B is replaced with $(t+d)$. This is the extension of [NHT94]. So, the delay transition from (B, i) can be defined as

$$(B, i) \xrightarrow{e(d), d \le 3} (a[2 \le (t+d) \le 3 \wedge x_0 = (t+d)]; b[(t+d) = x_0 + 3]; \\ c[(t+d) = x_0 + 4]; \textbf{stop}, a).$$

The condition such as $d \le 3$ is easily obtained from $[2 \le t \le 3 \wedge x_0 = t]$the transition condition of a. In this case, $\exists d' \exists x_0 [d \le d' \wedge [2 \le d' \le 3 \wedge x_0 = d']]$ is equivalent to $d \le 3$. In general, if $B = a[P(t,x)]; B'$, then the delay transition from (B, i) is defined as $(B, i) \xrightarrow{e(d), \exists d' \exists x [d \le d' \wedge P(d', x)]} (B\{(t+d)/t\}, a)$.

3.6.2 Action Transitions of LOTOS/T

From the previous section, each active state (B, a), which is reachable from any idle state by a delay transition, represent the behaviour where d units of time have elapsed. So, similar to [NHT94], the transition condition is defined as the condition whether the first action is executable at time 0. The transition condition may contain some undefined variable to which the executed time of the action will be assigned.

Since the action is considered instantaneous, we do not have to consider delay in action transition. So the destination behaviour is obtained similarly to LOTOS.

For example, for the behaviour expression

$$B'' = a[2 \le (t+d) \le 3 \wedge x = (t+d)]; b[(t+d) = x+3]; c[(t+d) = x+4]; \mathbf{stop},$$

the action transition

$$(B'', a) \xrightarrow{a, 2 \le (0+d) \le 3 \wedge x = (0+d)} (b[(t+d) = x+3]; c[(t+d) = x+4]; \mathbf{stop}, i)$$

is defined. Note that since $[2 \le (0+d) \le 3 \wedge x = (0+d)]$ holds after a is executed, the value of x is equal to the duration d in the succeeding behaviour.

When a process is defined recursively such as $P(x) := a[t \le x+3 \wedge y = t]; b[t \le y \wedge z = t]; P(z)$, the states $(P(x), i)$ and $(P(z), i)$ are essentially the same state if $x = z$. However, because the names of the variables are different, the two states are treated differently in the symbolic semantics. To unify the two states above, we replace these variables with the *minimum* one of all possible new variables (we assume some total order is defined on variables). This is similar to [JP89].

For example, assume that the set of variables is $\{x, y, z, d, d'\}$ and that a total order of the variables is defined as $x < y < z < d < d'$. For the above example $P(x)$, the corresponding A-TSLTS is obtained as follows.

$$(P(x), i) \xrightarrow{e(d), d \le x+3} (a[(t+d) \le x+3 \wedge y = (t+d)]; b[(t+d) \le y \wedge z = (t+d)]; P(z), a)$$

$$\begin{aligned}&(a[(t+d) \le x+3 \wedge y = (t+d)]; b[(t+d) \le y \wedge z = (t+d)]; P(z), a)\\ &\quad \xrightarrow{a, (0+d) \le x+3 \wedge \underline{y} = (0+d)} (b[(t+d) \le \underline{y} \wedge z = (t+d)]; P(z), i)\end{aligned}$$

The undefined variable y is replaced with y itself. Since the defined variable x is also contained in the condition, the minimum new variable is y.

$$\begin{aligned}&(b[(t+d) \le y \wedge z = (t+d)]; P(y), i)\\ &\quad \xrightarrow{e(d'), d'+d \le y} (b[(t+d+d') \le y \wedge z = (t+d+d')]; P(z), a)\end{aligned}$$

$$(b[(t+d+d') \le y \wedge z = (t+d+d')]; P(z), a) \xrightarrow{b, (0+d+d') \le y \wedge \underline{x} = (0+d+d')} (P(\underline{x}), i)$$

The undefined variable z is replaced with the minimum new variable x. Because of the replacement, the obtained A-TSLTS has a loop, which corresponds to the recursion.

3.6.3 Nondeterministic Choice and Parallel Execution

For all compositional operators of LOTOS/T, delay and action transitions are also defined. Although we only describe how the transitions of choice and parallel constructs are defined, the other cases are similar.

For choice constructs $B_1[]B_2$, time passing is allowed if and only if it is allowed by *either* B_1 or B_2 (non-persistent choice). This means that time may elapse until reaching the deadline of the first action of either B_1 or B_2. [2] So, in general, the delay transition can be defined as

$$(B_1[]B_2, i) \xrightarrow{e(d), P_1 \vee P_2} (B_1'[]B_2', a),$$

if $(B_1, i) \xrightarrow{e(d), P_1} (B_1', a)$ and $(B_2, i) \xrightarrow{e(d), P_2} (B_2', a)$. The action transitions are defined similar to LOTOS.

For example, if $B_1 = a[t \leq 2]; \mathbf{stop}$ and $B_2 = b[t \leq 3]; \mathbf{stop}$, then

$$(B_1[]B_2, i) \xrightarrow{e(d), d \leq 2 \vee d \leq 3} (a[(t+d) \leq 2]; \mathbf{stop}[]b[(t+d) \leq 3]; \mathbf{stop}, a),$$

$$(a[(t+d) \leq 2]; \mathbf{stop}[]b[(t+d) \leq 3]; \mathbf{stop}, a) \xrightarrow{a, (0+d) \leq 2} (\mathbf{stop}, i),$$

$$(a[(t+d) \leq 2]; \mathbf{stop}[]b[(t+d) \leq 3]; \mathbf{stop}, a) \xrightarrow{b, (0+d) \leq 3} (\mathbf{stop}, i).$$

For parallel constructs $B_1|[G]|B_2$, time of both B_1 and B_2 synchronizes each other in our semantics. In this case, the delay transition from $(B_1|[G]|B_2, i)$ is defined as

$$(B_1|[G]|B_2, i) \xrightarrow{e(d), P_1 \wedge P_2} (B_1'|[G]|B_2', a),$$

where $(B_1, i) \xrightarrow{e(d), P_1} (B_1', a)$ and $(B_2, i) \xrightarrow{e(d), P_2} (B_2', a)$. The case of synchronized action transition (the case where the action a is in G) is similar. That is, the transition condition is a logical product of the transition condition of each component. The case of interleaving action transition is similar to LOTOS.

3.7 Conclusions

In this chapter, we proposed a model A-TSLTS which can describe timed processes, and a verification method of timed verification equivalence for an A-TSLTS using a method similar to [HL95].

[2]Note that if we use persistent choice semantics instead, we have only to modify the guard predicate of the delay transition from $P_1 \vee P_2$ to $P_1 \wedge P_2$ (i.e., time passing is allowed as long as it is allowed *both* B_1 and B_2).

In contrast to other proposals for timed processes, our model allows arbitrary decidable 1st-order logic on any time domain for describing time constraints. In the model we can describe time constraints in a very flexible manner and still we can verify timed bisimulation equivalence whose cost is independent of the amount of constants used in the time constraints. Although we do not handle value-passing in this thesis, our model can be easily extended to the model with both time and value-passing by extending action transitions to have input/output values.

The future work is to extend the result to the verification of timed weak bisimulation equivalence (internal actions are considered), and to implement the algorithm and evaluate the cost of the verification for practically large processes.

Chapter 4

Decomposition of Structured Specifications of Real-Time Services

4.1 Introduction

In this chapter, we propose a method for synthesizing correct protocol specifications automatically from given service specifications written in a sub-class of LOTOS/T.

This chapter is organized as follows. In Section 4.2 we explain the protocol synthesis method. Section 4.3 discusses possible extensions of our synthesis method. Section 4.4 concludes this chapter.

4.2 Protocol Synthesis

4.2.1 Protocol Synthesis Problem

In this section, we define a protocol synthesis problem from timed service specifications. First we introduce some notations. Let $place(a)$ denote a node assignment for the action a. In the rest of this chapter, we assume that a^k stands for an action a with $place(a) = k$. Moreover, we use some notations $SP(B)$, $EP(B)$, $AP(B)$, whose intuitive meanings are the sets of the starting nodes of B, the ending nodes of B, all the participating nodes in B, respectively.

For example, if $B = a^1; b^2; \textbf{exit} ||| e^3; d^2; \textbf{exit}$, then $SP(B) = \{1, 3\}$, $EP(B) = \{2\}$ and $AP(B) = \{1, 2, 3\}$. We can derive them from B and $place()$ mechanically. The formal definitions of these notations appeared in [KHB96].

[Protocol Synthesis Problem]

Assumptions: 1. there exists a reliable(error-free), asynchronous, full-duplex communication channel between every two nodes.

2. there's no limitations on contents of messages exchanged among nodes.
3. all nodes have their own clocks and they always synchronize each other.

Inputs:

- A service specification S.
- A node assignment $place(a)$ for each action a.
- An upper bound of delay $d_{ij_{max}}$ for each channel from node i to j, such that $d_{ii_{max}} = 0$ and $\forall k\ d_{ij_{max}} \leq d_{ik_{max}} + d_{kj_{max}}$.

Here, we give the following restrictions for simplifying the derivation.

Restriction 1. S does not contain any deadlock states.

Restriction 2. If S contains $B_1[> B_2$ as a subexpression, B_1 must be a finite process, and there exists a constant t_0 such that B_1 can execute no action after time t_0 and B_2 can execute any action only after time t_0.

Restriction 3. If S contains $B_1 >> B_2$ as a subexpression, B_1 must be a finite process.

Restriction 4. Every process invocation in S must not have any process parameters, i.e. the behaviour of each invoked process does not depend on the previous behaviour.

Restriction 5. The context of each process invocation P must be either $a; P$ or $a[P(t, \bar{x})]; P$, so that just one action precedes P.

Restriction 6. For every subexpression $B_1[]B_2$ of S, there exists a node p such that $SP(B_1) = SP(B_2) = \{p\}$, and $EP(B_1) = EP(B_2)$[KHB96].

Restriction 7. For every subexpression $B_1[> B_2$ of S, $EP(B_1) = EP(B_2)$[KHB96].

Restriction 8. For every subexpression $B_1|[A]|B_2$ of S, each occurrence of action $a \in A$ which has time constraint is only in (at most) one of B_1 or B_2.

Outputs: Protocol entity specifications $Node_1$, $Node_2$, ... , $Node_n$ for all nodes, which are *correct* in the following meaning:

Let I be the composite system which connects $Node_1$, $Node_2$, ... , $Node_n$ together with a communication medium which has channels from node i to j with maximum delay of $d_{ij_{max}}$. Intuitively, $\{Node_i\}_{i=1,2,\ldots,n}$ are correct w.r.t. S when S can strictly simulate I including timing properties, whereas I can simulate S if time is ignored. In this case, a set of executable time of each action in I is a nonempty subset of that of the corresponding action in S.

Formally, the correctness is defined as follows. Let

$$I = \textbf{hide } G \text{ in } (\textbf{asap } G_s \text{ in}$$
$$((Node_1|||Node_2|||\ldots|||Node_n)|[G]|Medium)),$$

where G is a set of all sending/receiving actions of synchronization messages

$\{s_{ij}(m), r_{ij}(m) \mid i,j \in \{1,2,\ldots,n\},\ m \in M\}$ and G_s is a set of all sending actions of synchronization messages $\{s_{ij}(m) | i,j \in \{1,2,\ldots,n\},\ m \in M\}$, and $Medium$ is a specification of the communication medium defined as follows:

$$\begin{aligned}
&Medium = |||_{i,j\in\{1,2,\ldots,n\}} Channel_{ij}\\
&Channel_{ij} = |||_{m\in M}(s_{ij}(m)[x=t];\\
&\quad r_{ij}(m)[x \le t \le x + d_{ij_{max}}]; Channel_{ij})
\end{aligned}$$

Note that under the asynchronous communication medium, the sending actions are executed as soon as possible they are enabled, because they are spontaneous. In contrast, the receiving actions are not spontaneous, so they are not executed as soon as possible.

Before defining the correctness, we need some preliminary definitions.

Definition 4.1 Relations $\overset{\alpha}{\Longrightarrow}_t, \xrightarrow{\alpha}_u, \overset{\alpha}{\Longrightarrow}_u$ are defined as follows:

$$B \overset{\alpha}{\Longrightarrow}_t B' \overset{def}{=} \begin{cases} B(\xrightarrow{i})^* \xrightarrow{\alpha} (\xrightarrow{i})^* B', \\ \qquad \alpha \in Act \cup \{\delta, \mathbf{tick}\} \\ B(\xrightarrow{i})^* B', \qquad \alpha = \epsilon \end{cases}$$

$$B \xrightarrow{\alpha}_u B' \overset{def}{=} \begin{cases} B(\xrightarrow{\mathbf{tick}})^* \xrightarrow{\alpha} (\xrightarrow{\mathbf{tick}})^* B', \\ \qquad \alpha \in Act \cup \{\delta, i\} \\ B(\xrightarrow{\mathbf{tick}})^* B', \qquad \alpha = \epsilon \end{cases}$$

$$B \overset{\alpha}{\Longrightarrow}_u B' \overset{def}{=} \begin{cases} B(\xrightarrow{i}_u)^* \xrightarrow{\alpha}_u (\xrightarrow{i}_u)^* B', \\ \qquad \alpha \in Act \cup \{\delta\} \\ B(\xrightarrow{i}_u)^* B', \qquad \alpha = \epsilon \end{cases}$$

□

Definition 4.2 A binary relation $\sqsubseteq_t$ on behaviour expressions is defined as a maximum one of relations $\mathcal{R}$ satisfying the following condition:

- If $I\mathcal{R}S$, then for all $\alpha \in Act \cup \{\delta, \epsilon\}$, all of the following conditions hold:
  1. If $I \overset{\alpha}{\Longrightarrow}_t I'$, then there exists some S' s.t. $S \overset{\alpha}{\Longrightarrow}_t S'$ and $I'\mathcal{R}S'$.
  2. If $I \overset{\mathbf{tick}}{\Longrightarrow}_t I'$, then there exists some S' s.t. $S \overset{\mathbf{tick}}{\Longrightarrow}_t S'$ and $I'\mathcal{R}S'$.
  3. If $S \overset{\alpha}{\Longrightarrow}_u S'$, then there exists some I' s.t. $I \overset{\alpha}{\Longrightarrow}_u I'$ and $I'\mathcal{R}S'$. □

Here we define the correctness.

Definition 4.3 We call a derived protocol specification $\{Node_i\}_{i=1,2,\ldots,n}$ as *$\sqsubseteq_t$-correct w.r.t. S* if the following relation holds:

$$\begin{aligned}
&\mathbf{hide}\ G \text{ in } (\mathbf{asap}\ G_s \text{ in } (\\
&(Node_1|||Node_2|||\ldots|||Node_n)|[G]|Medium)) \sqsubseteq_t S
\end{aligned}$$

□

4.2.2 Synthesis Method

Now we describe our method for synthesizing protocol specifications from timed service specifications.

Basically, we follow a similar idea to our previous work[KHB96, YHT94]. Thus, after each node executed an action, it sends messages to the nodes which execute the succeeding actions, informing them that it has finished. We refer this kind of messages as *synchronization messages.* To handle time constraints between actions on different nodes, we naturally assume that synchronization messages may also contain, if needed, information about the time at which preceding actions were executed. One major problem is that the communication delay may make it impossible to execute an action in time. In general, all realistic communication media have propagation delay, and we cannot neglect uncertainty of such a delay in most cases. To overcome this problem, we propose the following method. First, for a given service specification S, we decide where to insert actions sending or receiving synchronization messages to simulate S, according to the policy similar to [KHB96, YHT94]. Then we restrict time-constraints of some actions in S in order to guarantee the execution of succeeding actions are possible at the worst case of communication delay, keeping the restriction to a minimum. We represent the obtained specification as $Restr(S)$. Finally, from the restricted specification $S' = Restr(S)$, we derive protocol entity specifications for all nodes. If S and S' are equivalent[NHT94], the derived protocol specifications are guaranteed correct w.r.t. S.

In the following subsections, we describe how the simulation of the service specification S is done, and how we can define the transformation $Restr()$, for each construct of LOTOS/T.

4.2.2.1 Action Prefix

We can simulate Action Prefix $a^p[P(t,\bar{x})]; B$ by sending a synchronization message from node p to all the nodes in $SP(B)$.

If time constraints are specified by assignment and reference of the variables, nodes at which such variables are assigned to some values must propagate the values to the succeeding nodes.

Example 4.1

$$\begin{aligned} S \;=\; & a^1[x=t]; b^2[t \le x+5 \wedge y=t]; \\ & c^3[t \le x+7 \wedge t \le y+5]; \mathbf{exit} \end{aligned}$$

$$d_{12max} = d_{13max} = d_{23max} = 2$$

$$\begin{aligned} Node_1 \;&=\; a[x=t]; s_{12}(m,x); \mathbf{exit} \\ Node_2 \;&=\; r_{12}(m,x); b[t \le x+5 \wedge y=t]; \\ &\qquad s_{23}(m',x,y); \mathbf{exit} \\ Node_3 \;&=\; r_{23}(m',x,y); c[t \le x+5 \wedge t \le y+5]; \mathbf{exit} \end{aligned}$$ □

Here we can remove some redundancies in inserting synchronization messages when time is considered. Specifically, if there's no executable time of a succeeding action that is earlier than or equal to some executable time of the preceding action, and there's no values to propagate to succeeding nodes, the synchronization message at this place is of no need to guarantee actions' order, i.e., time implicitly guarantees the order (recall Assumption 3 in Section 4.2.1). For example, let $S = a^1[P(t,\bar{x})]$; $b^2[Q(t,\bar{y})]$; **exit** and suppose $\forall t, t', \bar{x}, \bar{y}[[P(t,\bar{x}) \wedge Q(t',\bar{y})] \Rightarrow t < t']$ is true. Then from the time constraints, a is always executed before b, so even if we simply execute a and b at different places, the order is still preserved. Therefore, we can remove the synchronization message from node 1 to node 2 in this case. This can be formalized as follows. For a behaviour expression $B = a^p[P(t,\bar{x})]; B'$, let $TopTC(B', t, \bar{y})$ denote a logical disjunction of time constraints of all starting actions of B'. We call the time constraint of the starting action a^p of B is *temporally non-overlapping* for B', if $\forall t, t', \bar{x}, \bar{y}[[P(t,\bar{x}) \wedge TopTC(B', t', \bar{y})] \Rightarrow t < t']$ holds.

Example 4.2 If the input is the following:

$$S = a^1[1 \leq t \leq 3]; b^2[4 \leq t \leq 5]; \textbf{exit}$$

$$d_{12max} = 4,$$

we will simply derive:

$$\begin{aligned} Node_1 &= a[1 \leq t \leq 3]; \textbf{exit} \\ Node_2 &= b[4 \leq t \leq 5]; \textbf{exit} \end{aligned}$$

because the time constraint a^1 is temporally non-overlapping for $b[4 \leq t \leq 5]; \textbf{exit}$, i.e. $\forall t, t'[[(1 \leq t \leq 3) \wedge (4 \leq t' \leq 5)] \Rightarrow (t < t')]$ holds. □

From now, we consider the case where communication delay affects the simulation. For action prefix $a^p[P(t,\bar{x})]; B$, we will derive a specification $Restr(S)$ whose time constraint of a^p is restricted so that there exists a time to execute the succeeding actions in B no matter how late the messages from the node p reach the nodes in $SP(B)$. Because we describe time constraints in Presburger formulas, we can easily restrict time constraints by logical conjunction.

Example 4.3 If the input is:

$$\begin{aligned} S = {} & a^1[1 \leq t \leq 3]; b^2[4 \leq t \leq 7]; \\ & c^3[5 \leq t \leq 10]; d^2[6 \leq t \leq 12]; \textbf{exit} \end{aligned}$$

$$d_{12max} = 4\ ,\ d_{23max} = 4,\ d_{32max} = 3,$$

we restrict the time constraint of each action as follows:

d^2: $6 \leq t \leq 12$ (unmodified)

c^3: $5 \le t \le 10 \wedge\ \exists t'(t' \ge t + d_{32max} \wedge 6 \le t' \le 12)$ $(\equiv 5 \le t \le 9)$
b^2: $4 \le t \le 7 \wedge\ \exists t'(t' \ge t + d_{23max} \wedge\ 5 \le t' \le 9\ )$ $(\equiv 4 \le t \le 5)$
a^1: $1 \le t \le 3$ (unmodified (by Example 4.2))

So the derived protocol entity specification will be the followings:

$$\begin{aligned} Node_1 &= a^1[1 \le t \le 3]; \mathbf{exit} \\ Node_2 &= b^2[4 \le t \le \underline{5}]; s_{23}(m1); \\ &\qquad r_{32}(m2); d^2[6 \le t \le 12]; \mathbf{exit} \\ Node_3 &= r_{23}(m1); c^3[5 \le t \le \underline{9}]; s_{32}(m2); \mathbf{exit} \end{aligned}$$

□

Now we can define $Restr(S)$ formally as follows.

Definition 4.4 If $S = a[Q(t,x)]; B$, then $Restr(S)$ is defined inductively as follows:

$$Restr(S) \overset{def}{=} Restr(S, \emptyset)$$

$$Restr(S,V) \overset{def}{=} \begin{cases} S & \text{if } B = \mathbf{exit}, B = \mathbf{stop} \\ & \text{or } B = P \text{ (Process invocation)}, \\ a[Q(t,\bar{x}) \wedge Q'(t)]; Restr(B, V \cup \bar{x}) & \\ & \text{otherwise.} \end{cases}$$

where, if $\{b_k[Q_k(t,y_k)] \mid k \in K\}$ is the set of starting actions of $Restr(B, V \cup \bar{x})$ with their time constraints,

$$Q'(t) \overset{def}{=} \begin{cases} \bigwedge_{k \in K}\{\exists t' \exists y_k[(t' \ge t + \\ \quad d_{place(a),place(b_k)_{max}}) \wedge Q_k(t', y_k)]\} \\ \quad \text{if } \forall t, t', x, y[[Q(t,x) \wedge \\ \quad TopTC(B, t', y)] \Rightarrow t < t'] \text{ is false,} \\ \quad \text{or } B \text{ contains some variables} \\ \quad \text{of } V \cup \bar{x}, \\ true \qquad \text{otherwise.} \end{cases}$$

□

To summarize this section, our derivation takes 3 steps:

Step 1 determine at what position the synchronization messages are needed.
Step 2 according to the results of Step 1 and $d_{ij_{max}}$, construct $Restr(S)$.
Step 3 decompose $Restr(S)$ into each node by the similar method to [KHB96, YHT94], already described above.

4.2.2.2 Choice

To simulate choice expressions, we must solve the problem about distributed choice and empty alternatives[KHB96]. A choice expression $B_1[]B_2$ is called *distributed* *choice* if the starting actions of B_1 and B_2 may be executed at different nodes. And we say that a node p has an *empty alternative* w.r.t. $B_1[]B_2$ if some actions in B_i may be executed at node p, whereas no actions in $B_{(i \bmod 2)+1}$ are executed at node p. Distributed choice may cause simultaneous execution of the starting actions of both B_1 and B_2. Empty alternatives on node p may cause unconditional execution of B_i even if $B_{(i \bmod 2)+1}$ is chosen. As for distributed choice, we avoid it by putting the same restriction (Restriction 6) as [KHB96]. We have proposed a method for solving the empty alternative problem for the untimed case in [KHB96]. But, here, we will use a slightly modified method. Unlike [KHB96], the node where choice was made should immediately sends messages to the nodes that have empty alternatives in order not to violate time constraints of succeeding processes. Moreover, to make sure each B_i would not terminate before the messages sent to the nodes which has empty alternatives reach the destinations, the ending nodes of the chosen expression will receive acknowledgments from the nodes with empty alternatives before executing the ending actions (note that if the starting action of B_i coincides the ending action of it, i.e., the maximum length of B_i's action sequences is 1, this simulation method may not be applicable). Furthermore, to simulate a choice expression $B_1[]B_2$ in the above way successfully, we must not remove redundant synchronization messages in both B_1 and B_2, discussed in Section 4.2.2.1 (Example 4.2), otherwise the intermediate actions of each B_i may be executed independently, no matter which alternative is chosen.

To make it possible to simulate choice in the way above, we must guarantee that all the messages reach the destinations in time by restricting the time constraints of some actions. For a choice expression $B_1[]B_2$, if the messages sent from the node choice was made wouldn't have reached the destinations, or the acknowledgments wouldn't return, before the chosen behaviour B_i have been done, extra time would be spent waiting for the messages. So we will restrict the time constraints of the starting actions of B_1 and B_2 so that the messages can reach in time.

Example 4.4 Consider the following input:

$$\begin{aligned} S \;=\; & a^1[2 \le t \le 5 \wedge x = t]; b^2[t \le x + 3]; c^3[t \le x + 6]; \mathbf{exit} \\ & []d^1[3 \le t \le 9]; e^3[t \le 10]; \mathbf{exit} \end{aligned}$$

$$d_{12max} = 2,\; d_{23max} = 4,\; d_{13max} = 3$$

We must restrict the time constraint of d^1 in order to make the message from node 1 to 2 and the acknowledgment from node 2 to 3 reach by time 10.

$$\begin{aligned} Restr(S) \;=\; & a^1[2 \le t \le 5 \wedge x = t]; b^2[t \le x + 2]; \\ & c^3[t \le x + 6]; \mathbf{exit} \\ & []d^1[\underline{3 \le t \le 4}]; e^3[t \le 10]; \mathbf{exit} \end{aligned}$$

Then, the specification of each node will be derived as follows:

$$\begin{aligned}
Node_1 &= a^1[2 \leq t \leq 5 \wedge x = t]; s_{12}(m1, x); s_{13}(m3, x); \\
&\quad \textbf{exit} \\
&\quad [] d^1[3 \leq t \leq 4]; (s_{13}(m2); \textbf{exit} ||| s_{12}(m4); \textbf{exit}) \\
Node_2 &= r_{12}(m1, x); b^2[t \leq x + 2]; s_{23}(m5); \textbf{exit} \\
&\quad [] r_{12}(m4); s_{23}(m4); \textbf{exit} \\
Node_3 &= r_{13}(m3, x); r_{23}(m4); c^3[t \leq x + 6]; \textbf{exit} \\
&\quad [](r_{13}(m2); \textbf{exit} ||| r_{23}(m4); \textbf{exit}) >> e^3[t \leq 10]; \textbf{exit} \qquad \square
\end{aligned}$$

For defining $Restr(S)$, we need the following auxiliary function $Restr'(S)$, which is the same as $Restr(S)$ except that no removal of redundant messages is considered.

Definition 4.5 $Restr'(S)$ is defined inductively as follows:
If $S = a[Q(t, x)]; B$, then,

$$Restr'(S) \overset{def}{=} \begin{cases} S & \text{if } B = \textbf{exit}, B = \textbf{stop} \\ & \text{or } B = P \text{ (Process invocation)}, \\ a[Q(t, x) \wedge Q''(t)]; Restr'(B) & \\ & \text{otherwise.} \end{cases}$$

where, if $\{b_k[Q_k(t, y_k)] \mid k \in K\}$ is the set of starting actions of $Restr(B)$ with their time constraints,

$$Q''(t) \overset{def}{=} \bigwedge_{k \in K} \{\exists t' \exists y_k [t' \geq t + d_{place(a), place(b_k)_{max}} \wedge Q_k(t', y_k)]\}$$

Otherwise,

$$Restr'(S) \overset{def}{=} Restr(S) \qquad \square$$

Now we can define $Restr(S)$ as follows:

Definition 4.6 If $S = B_1[]B_2$, then $Restr(S)$ is defined inductively as follows:

$$Restr(S) \overset{def}{=} Restr'(f(B_1))[]Restr'(f(B_2))$$

where, we assume that $\{b_k[Q_k(t, y_k)] \mid k \in K\}$ is the set of the starting actions of B_i with their time constraints, and that $f(B_i)$ is an expression B_i whose time constraint of each starting action $Q_k(t, y_k)$ is replaced with $Q_k(t, y_k) \wedge R'_k(t)$. Here $R'_k(t)$ is a Presburger formula defined as follows.

$$R'_k(t) \overset{def}{=} \bigwedge_{\substack{q \in AP(B_i) \setminus AP(B_{(i \bmod 2)+1}) \\ l \in L, r \in EP(B_i)}} \{\exists t' \exists z_l [t' \geq t + d_{pq_{max}} + d_{qr_{max}} \wedge R_l(t', z_l)]\}$$

where $\{R_l(t, z_l) | l \in L\}$ denotes the time constraints of $EP(B_i)$ and $SP(B_i) = \{p\}$.$\square$

4.2.2.3 Asynchronous Parallel

For any asynchronous parallel expression $B_1|||B_2$, B_1 and B_2 are executed independently. So any synchronization messages are necessary between B_1 and B_2. Thus, $Restr(S)$ is defined as follows:

Definition 4.7 If $S = B_1|||B_2$, then $Restr(S) \stackrel{def}{=} Restr(B_1)\ |||\ Restr(B_2)$ □

4.2.2.4 Enabling

For enabling expression $B_1 >> B_2$, we can apply essentially the same idea as action prefix. The difference is that we must add time constraints between processes B_1 and B_2, not between actions. We will achieve this by adding time constraints between each action in $EP(B_1)$ and each action in $SP(B_2)$.

However, in each enabling expression we sequentially connect two *processes*. So in general it is possible that the preceding process is an infinite process and it may execute actions indefinite times before the succeeding process is invoked. That is very problematic. Take a look at the following example.

Example 4.5 Consider the following behaviour expression:

$$a[t=2]; P >> b[3 \le t \le 4]; \textbf{exit}$$

If the process P is defined as $P := c[t=1]; P[]d[t=1]; \textbf{exit}$, the action b cannot be executed before time 4 when c is executed more than 3 times (because at least 3 units of time must have passed since P is invoked). □

Thus, in this case, whether the succeeding actions can be executed depends on the number of the recursion. Because it is difficult to analyze, we add Restriction 3 for simplicity. From Restriction 4, B_1 in each enabling expression $B_1 >> B_2$ must be a finite process. So we can avoid the problem described above.

Example 4.6 Consider the following input:

$$\begin{aligned} S \ = \ & (a^1[1 \le t \le 3]; \textbf{exit}|||b^2[2 \le t \le 4]; \textbf{exit}) \\ & >> (c^3[2 \le t \le 7]; \textbf{exit}|||d^4[3 \le t \le 8]; \textbf{exit}) \end{aligned}$$

$$d_{13max} = 5\ ,\ d_{14max} = 7\ ,\ d_{23max} = 4\ ,\ d_{24max} = 3$$

The synchronization messages will be sent from node 1 to both nodes 3 and 4 to guarantee a^1 is executed before c^3 and d^4. Thus, the time constraint of a^1 must be restricted to make the message reach both node 3 by time 7 and node 4 by time 8. A similar restriction must be made for b^2.

$$\begin{aligned} Restr(S) \ = \ & (a^1[\underline{1 \le t \le 1}]; \textbf{exit}|||b^2[\underline{2 \le t \le 3}]; \textbf{exit}) \\ & >> (c^3[2 \le t \le 7]; \textbf{exit}|||d^4[3 \le t \le 8]; \textbf{exit}) \end{aligned}$$

So, the protocol entity specification of each node will be derived as follows:

$$\begin{aligned}
Node_1 &= a^1[1 \leq t \leq 1]; \mathbf{exit} \\
&\quad >> (s_{13}(m1); \mathbf{exit} ||| s_{14}(m1); \mathbf{exit}) >> \mathbf{exit} \\
Node_2 &= b^2[2 \leq t \leq 3]; \mathbf{exit} \\
&\quad >> (s_{23}(m2); \mathbf{exit} ||| s_{24}(m2); \mathbf{exit}) >> \mathbf{exit} \\
Node_3 &= \mathbf{exit} >> (r_{13}(m1); \mathbf{exit} ||| r_{23}(m2); \mathbf{exit}) \\
&\quad >> c^3[2 \leq t \leq 7]; \mathbf{exit} \\
Node_4 &= \mathbf{exit} >> (r_{14}(m1); \mathbf{exit} ||| r_{24}(m2); \mathbf{exit}) \\
&\quad >> d^3[3 \leq t \leq 8]; \mathbf{exit}
\end{aligned}$$

□

The formal definition of $Restr(S)$ follows:

Definition 4.8 If $S = B_1 >> B_2$, then $Restr(S)$ is defined inductively as follows:

$$Restr(S) \overset{def}{=} Restr(g(B_1)) >> Restr(B_2)$$

where $g(B_1)$ is an expression obtained by replacing the time constraint $P_k(t, x_k)$ of each ending action $a_k[P_k(t, x_k)]$ in B_1 with $P_k(t, x_k) \land P'_k(t)$. Here, $P'_k(t)$ is a Presburger formula defined as follows:

$$P'_k(t) \overset{def}{=} \bigwedge_{l \in L} \{\exists t' \exists y_l [t' \geq t + d_{place(a_k), place(b_l)_{max}} \land Q_l(t', y_l)]\}$$

where $\{b_l[Q_l(t, y_l)] \mid l \in L\}$ is the set of starting actions of $Restr(B_2)$ with their time constraints. □

4.2.2.5 Disabling

For each disabling expression $B_1[> B_2$, we make a strong restriction, Restriction 2, for simplicity. That is, for some t_0, all actions in B_1 are not executable after time t_0, and all actions in B_2 are executable only after time t_0. From Restriction 2, there is no cases that actions in B_1 and B_2 are simultaneously enabled at different nodes. So $B_1[> B_2$ can be simulated by inserting messages to notify successful termination of B_1 to all nodes.

To make this simulation method work,the messages notifying B_1's termination have to reach before t_0.

Example 4.7 The input described below satisfies Restriction 2 ($t_0 = 11$) and Restriction 7:

$$\begin{aligned} S \; = \; & a^1[1 \le t \le 4]; b^2[3 \le t \le 8]; c^3[7 \le t \le 10]; \mathbf{exit} \\ & [> d^3[12 \le t]; \mathbf{exit} \end{aligned}$$

$$d_{12max} = 3,\ d_{23max} = 4,\ d_{31max} = 3,\ d_{32max} = 4,\ d_{ij_{max}} = 2 \text{ for other } i,j.$$

In order to guarantee that the notification of successful termination sent from node 3 to nodes 1 and 2 can reach before time $t_0 = 11$, the time constraint of c^3 must be restricted to $7 \le t \le 7$, because the notification from node 3 to node 2 may take $d_{32max} = 4$ units of time. Then, the restriction discussed in the previous section is applied for b^2 and a^1.

$$\begin{aligned} Restr(S) \; = \; & a^1[\underline{1 \le t \le 1}]; b^2[\underline{3 \le t \le 4}]; \\ & c^3[\underline{7 \le t \le 7}]; \mathbf{exit}[> d^3[12 \le t]; \mathbf{exit} \end{aligned}$$

From this, the protocol entity specification of each node will be derived as below:

$$\begin{aligned} Node_1 \; &= \; a^1[1 \le t \le 1]; r_{31}(m1); \mathbf{exit}[> i[t = 12]; \mathbf{exit} \\ Node_2 \; &= \; b^2[3 \le t \le 4]; r_{32}(m1); \mathbf{exit}[> i[t = 12]; \mathbf{exit} \\ Node_3 \; &= \; c^3[7 \le t \le 7]; (s_{31}(m1)|||s_{32}(m1)) >> \mathbf{exit} \\ & [> d^3[12 \le t]; \mathbf{exit} \end{aligned} \qquad \square$$

The definition of $Restr(S)$ is as follows:

Definition 4.9 If $S = B_1[> B_2$,

$$Restr(S) \overset{def}{=} Restr(g'(B_1))[> Restr(B_2),$$

where $g'(B_1)$ represents a transformation replacing the time constraint $P(t, \bar{x})$ of each ending action of B_1 with $P(t, \bar{x}) \wedge P'(t)$. Here

$$P'(t) \overset{def}{=} \bigwedge_{p \in EP(B_1), q \in ALL} \{t + d_{pq_{max}} \le t_0\}. \qquad \square$$

4.2.2.6 Process Invocation

In our specification language, time is reset to 0 at every moment processes are invoked, avoiding accumulation of time constraints. To simulate this in distributed environments, we make all nodes to pretend as if they invoke a process simultaneously. In order to do so,

1. Fix one node for a *responsible node*, which decides the time to invoke a process (the time just before invoking a process). In this chapter, from Restriction 5, the context of each process invocation must be the form of $a; B$ or $a[P(t, x)]; B$. So we fix the node $place(a)$ as the responsible node w.r.t. the process P.

2. The responsible node notifies the invocation time of the process to all nodes, and immediately invokes the process locally.
3. The other nodes except the responsible node receive the notification, and invoke the process whose time constraints are modified to make the invocation time be virtually equal to that of the responsible node. Recall that the actual local time is reset to 0 just after the process invocation of each node.

To implement 3., we modify each process P without parameters in service specifications to $P(e_P)$ with just one parameter e_P in protocol specifications (Restriction 4), and replace every occurrence of t in the right hand of the process definition of P with $t + e_P$. The parameter e_P represents the difference between the actual invocation time and virtual invocation time. For example, $P(3)$ means the process P with replacing its time constraint, for instance, $t \leq 5$, with $t + 3 \leq 5$. Corresponding to each process invocation of P in the service specification, we derive a protocol specification such that (1.)the responsible node sends the current time t_P to every other node just before invoking $P(0)$, and (2.)the other nodes invoke $P(t - t_P)$ after receiving t_P from the responsible node. The time $t - t_P$ corresponds to the actual communication delay from the responsible node.

Note that the process P may be called by another process Q. In such a case, the variable t in $P(t - t_P)$ should be adjusted to represent the virtual time at which Q had been invoked. So it should be modified to $P(t + e_Q - e_P)$ if this process invocation occurs in the right hand of the definition of process Q, where $t + e_Q$ represents the virtual invocation time of Q.

Example 4.8

$$
\begin{aligned}
P &:= a^1[2 \leq t \leq 4 \wedge x = t]; b^2[t \leq x + 5]; P \\
&\quad [] c^1[5 \leq t]; \mathbf{exit} \\
&\qquad d_{12max} = 4 \ , \ d_{21max} = 3 \\
Node_1 &= P(e_P) := a^1[2 \leq t + e_P \leq 4 \wedge x = t + e_P \wedge \\
&\quad \exists t'(t' \geq t + e_P + d_{12max} \wedge t' \leq x + 5)]; \\
&\qquad s_{12}(m1, x); r_{21}(m3, t_P); P(t + e_P - t_P) \\
&\qquad >> s_2(m4); \mathbf{exit} [] c^1[5 \leq t + e_P]; \mathbf{exit} \\
Node_2 &= P(e_P) := r_{12}(m1, x); b^2[t + e_P \leq x + 5]; \\
&\qquad s_{21}(m2, x); s_{21}(m3, t + e_P); P(0) \\
&\quad [] r_{12}(m4); \mathbf{exit}
\end{aligned}
$$
□

To make this simulation possible, we check whether the starting action of each process cannot be late if the notification from the responsible node would reach in maximum delay. For consistency, we include this checking into $Restr()$. If the checking is false, the time constraint of the starting action becomes "false."

Definition 4.10 If $S = P$ where $P := B$, $Restr(S)$ is defined inductively as follows:

$$Restr(S) \overset{def}{=} P \text{ where } P := h(Restr(B))$$

where $h(Restr(B))$ is an expression obtained by replacing the time constraint $Q_k(t, x_k)$ of each starting action a_k of $Restr(B)$ with $Q_k(t, x_k) \wedge Q'_k$. Here Q'_k is a Presburger formula defined as follows :

$$Q'_k \overset{def}{=} \bigwedge_{p=1,\dots,n} \{\exists t' \exists x_k [t' \geq 0 + d_{p,place(a_k)_{max}} \wedge Q(t', x_k)]\} \qquad \square$$

4.2.3 Synthesis Algorithm

The synthesis algorithm consists of two parts:

1. For a given service specification S, an assignment of each action to a node, and a maximum delay $d_{ij_{max}}$ for each pair of nodes, construct $S' = Restr(S)$.
2. If $S \sim_u S'$, i.e., S and S' are bisimulation equivalent when time is ignored, derive a protocol entity specification $Node_i$ of each node i from S'. Otherwise, do not derive and halt.

$\sim_u$ denotes *untimed strong bisimulation equivalence* defined by Definition 2.10 in Chapter 2.

The algorithm to derive each $Node_i$ from S' is defined as follows. In the following, we assume that for any subexpression B of S, $N(B)$ is a unique ID of B, that q is any node such that $p \neq q$, and that V represents a set of variables (corresponding to *defined variables* as defined in [NHT94]).

Definition 4.11 A transformation $T_p(B)$, which derives a protocol entity specification of node p from a behaviour expression B, is defined inductively as Fig. 4.1. The auxiliary functions used in the definition of $T_p(B)$ are defined as Fig. 4.2. $\square$

Remark: For $T_p(B, V, p, \rho, B', B'')$ in Fig. 4.1, the parameters B, V, p, ρ, B' B'' mean, a service specification, a set of defined variables, a responsible node, a boolean-valued flag indicating whether redundant synchronization messages can be removed, ,the behaviour expression of message exchanges necessary for simulating choice expressions, the behaviour expression also necessary for simulating choice expressions inserted just before **stop** or **exit**, respectively.

Remark2: The intuitive meanings of functions defined in Fig. 4.2, are the followings:

$send_p(P, N, V)$: node p sends to each node in P a message labelled by ID N, with values of the variables in V.

$receive_p(P, N, V)$: node p receives from each node in P a message labelled by ID N, with values of the variables in V.

$$\begin{aligned}
T_p(B) &= T_p(B, \emptyset, p_0, true, empty, empty)\\
T_p(\mathbf{stop}, V, p', \rho, B', B'') &= B'' >> \mathbf{stop}\\
T_p(\mathbf{exit}, V, p', \rho, B', B'') &= B'' >> \mathbf{exit}\\
T_p(a^p; B, V, p', \rho, B', B'') &= a^p; send_p(SP(B) \setminus \{p\}, N(a^p; B), V) >> B' >> T_p(B, V, p, \rho, empty, B'')\\
T_p(a^q; B, V, p', \rho, B', B'') &= \begin{cases} receive_p(\{q\}, N(a^q; B), V) >> B' >> T_p(B, V, q, \rho, empty, B'') \\ \qquad\qquad \text{if } p \in SP(B) \\ B' >> T_p(B, V, q, \rho, empty, B'') \\ \qquad\qquad \text{otherwise.} \end{cases}\\
T_p(a^p[P(t,\bar{x})]; B, V, p', \rho, B', B'') &= \begin{cases} a^p[P(t,\bar{x})]; send_p(SP(B) \setminus \{p\}, N(a^p[P(t,\bar{x})]; B), V \cup \bar{x}) \\ \qquad\qquad >> B' >> T_p(B, V \cup \bar{x}, p, \rho, empty, B'') \\ \qquad \text{if } \forall t, t', \bar{x}, \bar{y}[[P(t,\bar{x}) \wedge TopTC(B, t', \bar{y})] \Rightarrow (t < t')] \text{ is false,} \\ \qquad \text{or } \rho = false \\ a^p[P(t,\bar{x})]; B' >> T_p(B, V \cup \bar{x}, p, \rho, empty, B'') \qquad \text{otherwise.} \end{cases}\\
T_p(a^q[P(t,\bar{x})]; B, V, p', \rho, B', B'') &= \begin{cases} receive_p(\{q\}, N(a^q[P(t,\bar{x})]; B), V \cup \bar{x}) >> B' \\ \qquad\qquad >> T_p(B, V \cup \bar{x}, q, \rho, empty, B'') \\ \qquad \text{if } p \in SP(B) \text{ and} \\ \qquad\qquad \forall t, t', \bar{x}, [[P(t,\bar{x}) \wedge TopTC(B, t', \bar{y})] \Rightarrow (t < t')] \text{ is false} \\ \qquad \text{or } \rho = false, \\ B' >> T_p(B, V \cup \bar{x}, q, \rho, empty, B'') \qquad \text{otherwise.} \end{cases}\\
T_p(B_1[]B_2, V, p', \rho, B', B'') &= T_p(B_1, V, p', false, Alternative_p(B_1, B_2)|||B', Alternative2_p(B_1, B_2)|||B'')\\
&\quad []T_p(B_2, V, p', false, Alternative_p(B_2, B_1)|||B', Alternative2_p(B_2, B_1)|||B'')\\
T_p(B_1|||B_2, V, p', \rho, B', B'') &= T_p(B_1, V, p', \rho, B', B'')|||T_p(B_2, V, p', \rho, B', B'')\\
T_p(B_1[> B_2, V, p', \rho, B', B'') &= (T_p(B_1, V, p', false, B', B'') >> Rel_p(B_1))\\
&\quad [> T_p(B_2, V, p', B', false, Alternative_p(B_2, ALL)|||B',\\
&\qquad Alternative2_p(B_2, ALL)|||B'')\\
T_p(B_1 >> B_2, V, p', \rho, B', B'') &= T_p(B_1, V, p', \rho, B', B'') >> Synch\_Left_p(B_1, B_2)\\
&\quad >> Synch\_Right_p(B_1, B_2) >> T_p(B_2, V, \min EP(B_1), \rho, B', B'')\\
T_p(P[g_1, \ldots, g_k], V, p, \rho, B', B'') &= send_p(SP(B_P) - \{p\}, N(P[g_1, \ldots, g_k], \{t\}) >> P[g_1, \ldots, g_k](0)\\
T_q(P[g_1, \ldots, g_k], V, p, \rho, B', B'') &= receive_q(\{p\}, N(P[g_1, \ldots, g_k], \{t_P\}) >> P[g_1, \ldots, g_k](t_P - t)\\
&\quad \text{iff a process definition } P[g_1, \ldots, g_k] := B_P \text{ exists.}\\
T_p(S \text{ where } P[g_1, \ldots, g_k] := B_P) &= T_p(S) \text{ where } P[g_1, \ldots, g_k](e_P) := [t + e_P/t]T_p(B_P)
\end{aligned}$$

Figure 4.1: Definition of $T_p(B)$

$Synch\_Left_p(B_1, B_2)$: each node in $EP(B_1)$ sends a synchronization message to each node in $SP(B_2)$

$Synch\_Right_p(B_1, B_2)$: each node in $SP(B_2)$ receives a synchronization message from each node in $EP(B_1)$.

$Rel_p(B)$: each node in $EP(B)$ notifies B's successful termination to all other nodes.

$Alternative_p(B_1, B_2)$: (the node where choice was made) sends messages to the nodes participating to B_2 but not to B_1.

$Alternative2_p(B_1, B_2)$: the ending nodes of B_1 receive acknowledgments from the nodes participating to B_2 but not to B_1.

Then, we obtain our main theorem:

$$\begin{aligned}
\mathit{send}_p(P,N,V) &= \text{if } P=\emptyset \text{ then } \mathit{empty}\\
&\quad \text{if } P=\{i,j,\ldots,k\} \text{ then } (s_{pi}(N,V);\mathit{exit}(V)|||\ldots|||s_{pk}(N,V);\mathit{exit}(V))\\
\mathit{receive}_p(P,N,V) &= \text{if } P=\emptyset \text{ then } \mathit{empty}\\
&\quad \text{if } P=\{i,j,\ldots,k\} \text{ then } (r_{ip}(N,V);\mathit{exit}(V)|||\ldots|||r_{kp}(N,V);\mathit{exit}(V))\\
\mathit{Synch\_Left}_p(B_1,B_2) &= \begin{cases} \mathit{send}_p((SP(B_2)\setminus\{p\}),N(B_1),\emptyset) & \text{if } p\in EP(B_1)\\ \text{empty} & \text{otherwise.}\end{cases}\\
\mathit{Synch\_Right}_p(B_1,B_2) &= \begin{cases} \mathit{receive}_p((EP(B_1)\setminus\{p\}),N(B_1),\emptyset) & \text{if } p\in SP(B_2)\\ \text{empty} & \text{otherwise.}\end{cases}\\
\mathit{Rel}_p(B) &= \begin{cases} \mathit{send}_p((\mathit{Act}\setminus\{p\}),N(B),\emptyset)|||\mathit{receive}_p((EP(B)\setminus\{p\}),N(B),\emptyset) & \text{if } p\in EP(B)\\ \mathit{receive}_p(EP(B),N(B),\emptyset) & \text{otherwise.}\end{cases}\\
\mathit{Alternative}_p(B_1,B_2) &= \begin{cases} \mathit{send}_p(AP(B_2)\setminus AP(B_1),N(B_1)) & \text{if } p\in SP(B_1)\\ \mathit{receive}_p(SP(B_1),N(B_1)) >> \mathit{send}(EP(B_1),N(B_1)) & \text{if } p\in AP(B_2)\setminus AP(B_1)\\ \mathit{empty} & \text{otherwise.}\end{cases}\\
\mathit{Alternative2}_p(B_1,B_2) &= \begin{cases} \mathit{receive}_p(AP(B_2)\setminus AP(B_1),N(B_1)) & \text{if } p\in EP(B_1)\\ \mathit{empty} & \text{otherwise.}\end{cases}
\end{aligned}$$

Figure 4.2: The Auxiliary functions used in Fig. 4.1.

Theorem 4.1 For a given service specification S, let $S' = \mathit{Restr}(S)$. If $S \sim_u S'$, the protocol specification $\{T_i(S')\}_{i=1,2,\ldots,n}$ is $\sqsubseteq_t$-correct w.r.t. the service specification S. $\Box$

Note that similar to [KHB96], derived protocol entity specifications by the algorithm $T_p(B)$ may have some redundancy. Some of those can be optimized by the similar method to [GB90].

4.3 Discussions

In this section, we discuss both weakening time constraints of the restricted service specification and extending the applicable class of the service specifications for our method.

4.3.1 Weakening Restriction of Time Constraints

We wish to weaken $\mathit{Restr}(S)$ as much as possible in order to enlarge the class for which protocol specification is derivable by our algorithm, i.e., to make $S \sim_u \mathit{Restr}(S)$ hold for as many S's as possible. $\mathit{Restr}(S)$ defined in Section 4.2.2 satisfies the following property: each action in $\mathit{Restr}(S)$ has the weakest timing constraint which makes the subsequent actions executable in time when every synchronization messages are *not* omitted in simulating S in distributed environment (under our policy). However, actually more redundant messages (than Section 4.2.2) can also be omitted by modifying the overlapped time constraints to non-overlapping ones, which we refer to as *overlapping removal*. Thus, in this section we make the time constraints of $Restr(S)$ slightly more weaker.

First, for each service specification $S = a[P(t,x)];B$ ($a \in Act$), let $TSync(t)$ denote the predicate which is true iff t is less than any time at which some of the starting action in B is executable ($TSync(t)$ is easily represented by a Presburger formula). We modify Definition 4.4 by replacing $Q'(t)$ with $TSync(t)$ if $\forall t[Q'(t) \Rightarrow TSync(t)]$ ($TSync(t)$ is weaker than $Q'(t)$) holds and otherwise $Q'(t)$ is retained. After the modification above, if overlapping removal is less restrictive than using synchronization synchronization, the algorithm $Restr()$ chooses overlapping removal for the time constraints.

Finally, for obtained $S' = Restr(S)$, the simulation policy by distributed nodes is modified so as not to send synchronization messages at the actions which is overlapping-removed.

By the modified method described above, we can increase the number of service specifications from which protocol entity specifications are derivable.

4.3.2 Extending Class of Service Specifications

Many of the restrictions imposed on inputs of our algorithm are for simplicity. Many of them are not essential restriction which heavily dependent on our simulation policy. Our simulation policy is merely one of all possible simulation policies. So introducing some other (possibly more complex) simulation policy, many of the restrictions we introduced may be removed.

However, it is difficult to remove the restrictions imposed on synchronous parallel(rendezvous) and disabling subexpressions.

At first, consider the case of the rendezvous subexpression $B_1|[G]|B_2$. If every action $a \in G$ and every subsequent action whose executable time is dependent on a, has a time constraint of form $[e \leq t]$ (no *rendezvous deadline*), we can derive protocol specification which contains Rendezvous by applying the same algorithm as described so far. Further extension is difficult by the following reasons:

- If the executable time of synchronous action $a \in G$ has an upper bound, $[e_1 \leq t \leq e_2]$ for example, in the subexpression B_1 or B_2, the actual upper bound of the synchronized behaviour $B_1|[G]|B_2$ is generally smaller than locally specified (in B_1 or B_2). So, even if there is a time to execute $a \in G$ in $Restr(B_1)|[G]|Restr(B_2)$, the synchronization message may be late for the actual upper bound of the subsequent actions of a. Therefore, to compute $Restr(B_1|[G]|B_2)$ when rendezvous deadline exists, we must analyze the behaviour expression globally.
- In order to obtain the actual time constraint of synchronous action $a \in G$, we may decide the combinations of synchronizing actions and take a logical

conjunction of them as the actual time constraint. However, this is too costly in general.

- In some service specification, synchronizing group of actions may vary dynamically on the execution. In such case, it is very difficult to analyze due to the possible state explosion.

Even if rendezvous with deadline is difficult to handle in our structural protocol synthesis, we may derive a correct protocol specification from a service specification which contains rendezvous with deadline, if we can expand it into the specification which is in finite length, does not contain any rendezvous operators and satisfies other restrictions imposed by the algorithm.

Restriction 2 imposed on disabling subexpression $B_1[> B_2$ is rather strong. However, it is known that if some action of B_1 and a starting action of B_2 are simultaneously executable, bisimulation equivalence between the service specification and the protocol specification may not be preserved. So if we take bisimulation as one of correctness criteria, such a restriction is necessary. Under the restriction, many of timeout scenarios are still expressible. Thus we think the restriction is not too strong for practical purposes.

Some system do need the specification which allows B_2 to interrupt B_1 while executing, not by timeout. In such a case, an alternative restriction as follows may be more useful. Divide up the time axis into time slots (finite intervals of constant length). The service specification must be arranged so that the two kind of slots alternate (like Time Sharing System), where B_1 is (temporally) disabled and B_2 is interruptible in one slot, and interruption of B_2 is disabled and B_1 is runnable in the other slot. Under this restriction, we can similarly consider the alternative simulation policy which enables a correct simulation of the service specification by distributed nodes.

Still another solution is possible. If we weaken the correctness criteria so that execution of B_1 is possible for a while when interruption occurred, Restriction 2 is no longer necessary.

4.4 Concluding Remarks

In this chapter, we have proposed a method to synthesize protocol specifications from timed service specifications written in LOTOS/T. The proposed method enables us to synthesize protocol specifications from both timed and structured service specifications. In contrast to [KBD95], our method restricts the time constraints of service specifications, not of the communication media, because the delay of the media depends on the physical lines, so it is more difficult to change them than those of the specifications. Moreover, our correctness criteria guarantee that the control structure of the derived protocol specification is a full, not partial, implementation of that of the service specification. Using the same timing extension as ours, our result should easily apply to other process models such as CCS.

The future work is to extend the class of service specifications and to establish a framework for evaluating performance aspects of the derived protocol entity specifications.

Chapter 5

Conclusions

In this thesis, we have proposed the following methods for specification, verification and decomposition of real-time services for design and developing reliable real-time distributed systems:

1. a specification language LOTOS/T, which is capable for expressing time constraints among only interested (possibly non-adjacent) actions, as well as urgency,
2. a verification method, which is applicable for the new semantic model A-TSLTS of LOTOS/T, and whose cost is independent of time domain nor contents of time constraints, and
3. a decomposition method for real-time services described in LOTOS/T.

In summary, we have proposed a language LOTOS/T, and shown its tractability by presenting a verification method of both timed and untimed bisimulation equivalence for A-TSLTS and a transformation method from LOTOS/T into A-TSLTS. Moreover, a decomposition method for real-time services is presented. The presented decomposition method strongly depends on the expressive power of LOTOS/T. Without capability of describing time constraints which may contain quantifiers of 1st-order logic, the basic idea of our decomposition would not be so simple. The specifications with quantifiers can be still verified by our symbolic verification method on A-TSLTSs.

Some discussions and future directions for our work is as follows.

We have adopted timed/untimed bisimulation equivalence as the correctness criteria of refined/modified specifications w.r.t. a given real-time service specification. So our verification method is basically *equivalence checking*. Equivalence checking is good for ensuring the correctness of service modifications (e.g. changing the timing constraints), as mentioned in Chapter 2. However, it might be too strong in some cases for ensuring implementation (refinement) correctness, since generally refinement may add some more actions, or constrain some more moves (i.e. the moves becomes more deterministic, which reduces the possibility of all moves). Therefore, we may need to develop verification techniques for some refinement relations (which are typically defined as pre-orders of systems) for real-time services, in addition to timed/untimed bisimulation equivalence.

There is another approach for specification and verification of real-time systems — a *Temporal Logic* approach. In this approach, requirements (services) are specified by temporal logic formulas, whereas, its implementations are described by formal models. Temporal logic is good for prototyping services in the earlier phase of a system development process. It is inherently compositional, that is, if we need to add or remove some functions/requirements/services, we can simply add or remove the corresponding descriptions in logic formulas. The verification method is *model checking*, i.e., checking whether the given implementation satisfies its requirements described in logic formulas. It is one of the most important future works to develop a model checking method for some temporal logics and a symbolic model such as our A-TSLTS model. Moreover, a temporal-logic version of synthesis problem is interesting. It is known as satisfiability problem of temporal logic formulas, that is, finding a model which satisfies the given temporal logic formulas.

In our decomposition method, rendezvous of actions in some parallel modules cannot be handled. This is due to the following two reasons in summary. First, rendezvous essentially requires simultaneity, that is, an action in rendezvous can be only executed when all of the modules participating rendezvous can perform the action at the same time. This troubles us especially when real-time property is considered. By this fact, the locality of time constraints is destructed, i.e., we cannot tell when an action can be executed (or not) by only looking at the local module specification, if the action is participating rendezvous with other modules. Thus, this makes it difficult to use structural divide-and-conquer methods. Second, rendezvous induces global cause among actions. By rendezvous, a causal relation is introduced between two actions which belong to different parallel modules, even if they do not rendezvous. For example, in a LOTOS expression "$a; b; ...|[b]|b; c; ...$", the causal relation between a and c is introduced by the rendezvous of b. This is known as *global cause*. Thus, the executed time of a may affect the execution of c if we add time constraints to this expression. In order to cope with rendezvous in our decomposition method for real-time services, we must analyze global cause relation between actions, and restrict the time constraint properly so that the synchronization messages reach in time. Use of models for real-time systems which are based on causality such as [Kat96] might be necessary.

Bibliography

[ACH94] Rajeev Alur, Costas Courcoubetis, and Thomas A. Henzinger. The observational power of clocks. In *Proc. of CONCUR'94*, volume 836 of *Lecture Notes in Computer Science*, pages 162–177. Springer-Verlag, 1994.

[AD90] Rajeev Alur and David Dill. Automata for modelling real-time systems. In M. S. Paterson, editor, *Proc. of ICALP'90*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer-Verlag, 1990.

[AKH92] S. Arun-Kumar and M. Hennessy. An efficiency preorder for processes. *Acta Informatica*, 29:737–760, 1992.

[BB91] J. C. M. Baeten and J. A. Bergstra. Real time process algebra. *Journal of Formal Aspects of Computing Science*, 3(2):142–188, 1991.

[BG86] G. v. Bochmann and R. Gotzhein. Deriving protocol specification from service specifications. In *Proc. of the ACM SIGCOMM '86 Symposium*, pages 148–156, Vermont, USA, 1986.

[BK85] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoret. Comput. Sci.*, 37:77–121, 1985.

[BK90] J. C. M. Baeten and J. W. Klop, editors. *Proc. of CONCUR '90*, volume 458 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[BL92] Tommaso Bolognesi and Ferdinando Lucidi. LOTOS-like process algebras with urgent or timed interactions. In K. R. Parker and G. A. Rose, editors, *Formal Description Techniques, IV*, pages 249–264. IFIP, Elsevier Science Publishers B.V.(North-Holland), 1992.

[BLT90] Tommaso Bolognesi, Ferdinando Lucidi, and Sebastiano Trigila. From timed Petri nets to timed LOTOS. In Logrippo et al. [LPU90], pages 395–408.

[Ćer92] K. Ćerãns. Decidability of bisimulation equivalence for parallel timer processes. In *Proc. of 4th CAV*, volume 663 of *Lecture Notes in Computer Science*, pages 302–315. Springer-Verlag, 1992.

[Che92] Liang Chen. An interleaving model for real-time systems. In A. Nerode and M. Taitslin, editors, *Proc. of 2nd Int'l Symp. on Logical Foundations of Computer Science (LFCS'92)*, volume 620 of *Lecture Notes in Computer Science*, pages 81–92. Springer-Verlag, July 1992.

[CL88] P. M. Chu and M. T. Liu. Protocol synthesis in a state transition model. In *Proc. IEEE COMPSAC '88*, pages 505–512, 1988.

[FK95] W.J. Fokkink and A.S. Klusener. An effective axiomatization for real time ACP. Report CS-R9542, CWI, Amsterdam, June 1995. To appear in *Information and Computation.*

[GB90] R. Gotzhein and G. v. Bochmann. Deriving protocol specifications from service specifications including parameters. *ACM Trans. on Computer Systems*, 8(4):255–283, 1990.

[Gro90] Jan Friso Groote. Transition system specifications with negative premises. In Baeten and Klop [BK90], pages 332–341.

[Han91] Hans A. Hansson. *Time and Probability in Formal Design of Distributed Systems.* Ph.D thesis DoCS 91/27, Dept. of Computer Systems, Uppsala University, September 1991.

[HBL$^+$92] Teruo Higashino, Gregor von Bochmann, Xiangdong Li, Keiichi Yasumoto, and Kenichi Taniguchi. Test system for a restricted class of LOTOS expressions with data parameters. In *Proc. 5th IFIP Int'l Workshop on Protocol Test Systems*, pages 205–216. IFIP, North-Holland, September 1992.

[HKT92] Teruo Higashino, Junji Kitamiti, and Kenichi Taniguchi. Presburger arithmetic and its application to program developments. *Journal of Japan Society of Software Science and Technology*, 9(6):31–39, 1992. (In Japanese).

[HL95] M. Hennessy and H. Lin. Symbolic bisimulations. *Theoret. Comput. Sci.*, 138:353–389, 1995.

[HLW91] U. Holmer, K. Larsen, and Y. Wang. Deciding properties of regular timed processes. In *Proc. of 3rd CAV*, volume 575 of *Lecture Notes in Computer Science*, pages 443–453. Springer-Verlag, 1991.

[HM85] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, January 1985.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[HOIT93] T. Higashino, K. Okano, H. Imajo, and K. Taniguchi. Deriving protocol specifications from service specifications in extended FSM models. In *Proc. of the 13th IEEE Int'l Conf. on Distributed Computing Systems (ICDCS-13)*, pages 141–148, 1993.

[HR91] Matthew Hennessy and T. Regan. A temporal process algebra. In J. Quemada, J. Mañas, and E. Vázquez, editors, *Formal Description Techniques, III*, pages 33–48. IFIP, Elsevier Science Publishers B.V.(North-Holland), 1991.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[Hul95] Maria Hultström. Structural decomposition. In Vuong and Chanson [VC95], pages 201–216.

[ISO89] ISO. *Information Processing System, Open Systems Interconnection, LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour.* IS 8807, January 1989.

[JP89] Bengt Jonsson and Joachim Parrow. Deciding bisimulation equivalences for a class of non-finite-state programs. In B. Monien and Cori R., editors, *Proc. of STACS'89*, volume 349 of *Lecture Notes in Computer Science*, pages 421–433. Springer-Verlag, 1989.

[Kat96] Joost-Pieter Katoen. *Quantative and Qualitative Extensions of Event Structures.* CTIT Ph.D-thesis series No.96-09, Centre for Telematics and Information Technology, Enshede, The Netherlands, September 1996.

[KBD95] A. Khoumsi, Gregor v. Bochmann, and R. Dssouli. On specifying services and synthesizing protocols for real-time applications. In Vuong and Chanson [VC95], pages 185–200.

[KBK89] F. Khendek, G. v. Bochmann, and C. Kant. New results on deriving protocol specifications from services specifications. In *Proc. of the ACM SIGCOMM '89*, pages 136–145, 1989.

[KHB96] Christian Kant, Teruo Higashino, and Gregor v. Bochmann. Deriving protocol specifications from service specifications written in LOTOS. *Distributed Computing*, 10(1):29–47, 1996.

[KS90] Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.

[Lan90] R. Langerak. Decomposition of functionality; a correctness-preserving LOTOS transformation. In Logrippo et al. [LPU90], pages 229–242.

[Lin96] Huimin Lin. Symbolic transition graph with assignment. In *Proc. of CONCUR'96*, Lecture Notes in Computer Sciences. Springer-Verlag, August 1996. to appear.

[LL93] Guy Leduc and Luc Léonard. A timed LOTOS supporting a dense time domain and including new timed operators. In M. Diaz and R. Groz, editors, *Formal Description Techniques, V*, pages 87–102. IFIP, Elsevier Science Publishers B.V.(North-Holland), 1993.

[LPU90] L. Logrippo, R. L. Probert, and H. Ural, editors. *Protocol Specification, Testing and Verification, X.* IFIP, Elsevier Science Publishers B.V.(North-Holland), 1990.

[LW93] Kim G. Larsen and Yi Wang. Time abstracted bisimulation: Implicit specifications and decidability. In S. Brookes, M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Proc. of 9th Int'l Conf. on Mathematical Foundations of Programming Semantics (MFPS'93)*, volume 802 of *Lecture Notes in Computer Science*, pages 160–175. Springer-Verlag, April 1993.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[MT90] Faron Moller and Chris Tofts. A temporal calculus of communicating systems. In Baeten and Klop [BK90], pages 401–415.

[MT91] Faron Moller and Chris Tofts. Relating processes with respect to speed. In J. C. M. Baeten and J. F. Groote, editors, *Proc. of CONCUR '91*, volume 527 of *Lecture Notes in Computer Science*, pages 424–438. Springer-Verlag, 1991.

[NHT94] Akio Nakata, Teruo Higashino, and Kenichi Taniguchi. LOTOS enhancement to specify time constraints among nonadjacent actions using first order logic. In R. L. Tenney, P. D. Amer, and M. Ü. Uyar, editors, *Formal Description Techniques, VI (FORTE'93)*, pages 451–466. IFIP, Elsevier Science Publishers B.V. (North-Holland), 1994.

[Par81] D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proc. of 5th GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[PS91] Robert L. Probert and Kassem Saleh. Synthesis of communication protocols: Survey and assessment. *IEEE Trans. Comput.*, 40(4):468–475, April 1991.

[QAF90] J. Quemada, A. Azcorra, and D. Frutos. TIC: A timed calculus for LOTOS. In Vuong [Vuo90], pages 195–209.

[SKTN90] Norio Shiratori, Hiroaki Kaminaga, Kaoru Takahashi, and Shoichi Noguchi. A verification method for LOTOS specifications and its application. In E. Brinksma, G. Scollo, and C. A. Vissers, editors, *Protocol Specification, Testing, and Verification, IX*, pages 59–70. IFIP, Elsevier Science Publishers B.V.(North-Holland), 1990.

[VC95] Son T. Vuong and Samuel T. Chanson, editors. *Protocol Specification, Testing and Verification, XIV (PSTV-XIV)*. IFIP, Chapman & Hall, 1995.

[vHTZ90] Wilfried H. P. van Hulzen, Paul A. J. Tilanus, and Han Zuidweg. LOTOS extended with clocks. In Vuong [Vuo90], pages 179–194.

[Vuo90] S. T. Vuong, editor. *Formal Description Techniques, II*. IFIP, Elsevier Science Publishers B.V.(North-Holland), 1990.

[Wan91] Yi Wang. CCS + time = an interleaving model for real time systems. In J. Leach Albert, B. Monien, and M. Rodríguez Artalejo, editors, *Proc. of ICALP '91*, volume 510 of *Lecture Notes in Computer Science*, pages 217–228. Springer-Verlag, 1991.

[YHMT92] Keiichi Yasumoto, Teruo Higashino, Toshio Matsuura, and Kenichi Taniguchi. PROSPEX: A graphical LOTOS simulator for protocol specifications with N nodes. *IEICE Trans. on Communication*, E75-

B(10):1015–1023, 1992.

[YHT94] Keiichi Yasumoto, Teruo Higashino, and Kenichi Taniguchi. Software process description using LOTOS and its enaction. In *Proc. of 16th Int'l Conf. on Software Engineering (ICSE-16)*, pages 169–179, May 1994.

[YOHT95] Hirozumi Yamaguchi, Kozo Okano, Teruo Higashino, and Kenichi Taniguchi. Synthesis of protocol entities' specifications from service specifications in a Petri net model with registers. In *Proc. of 15th IEEE Int'l Conf. on Distributed Computing Systems (ICDCS-15)*, May 1995.

List of Figures

List of Tables