MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Time Extension of Message Sequence Chart

BACHELOR THESIS

Ľuboš Korenčiak

Brno, Spring 2009

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

In Brno, May 22, 2009
Ľuboš Korenčiak

Advisor: RNDr. Vojtěch Řehák, Ph.D.

Acknowledgement

I would like to express my deepest gratitude to my advisor RNDr. Vojtěch Řehák, Ph.D., for his helpful comments, suggestions and patience. Next I would like to thank my family for its support through my life. Finally I thank Martin Chmelík, Martin Křivánek and Ondřej Kocian for fruitful discussions.

Abstract

Message sequence charts (MSC) is a graphical and textual formalism suitable for specifying distributed communication. It consists of the MSC and a High-level MSC (HMSC). The formalism incorporates the possibility of specifying time in designed systems. There arise severe problems regarding timing constraints in MSC, such as computation of minimal network. First attempts to solve the problems using various approaches have been presented.

In this thesis, we analyze different approaches to time extension of MSC. As most suitable appears to be the approach using labeled partially ordered sets. An extension to this approach is provided to handle computation of minimal network in MSC formalism with coregions and constraints in HMSC. Restriction to proper constraint specification is given, which rules out ambiguous and erroneous constraint specifications. Algorithm pseudocodes for checking proper time constraints are provided.

Keywords

Message Sequence Charts, Message Sequence Graph, timing consistency, tightening, minimal network

Contents

Chapter 1

Introduction

Message Sequence Chart (MSC) is an appealing formalism for specifying distributed communication, especially network protocols. MSCs are formally defined in ITU standard Z.120 [9]. They consist of equivalent graphical and textual notation. MSC is used in the early phases of protocol design process. Many works and approaches how to detect serious design faults in distributed systems specified in MSC formalism have been presented . These faults include deadlock and livelock.

There are already developed tools which support automated detection of these faults, such as *MSCan* [16], *uBet* [20] or *SCStudio* [18]. The programs differ by various approaches to the properties of MSCs and by ability to design more complex systems. The SCStuio is program which is being developed at Faculty of Informatics, Masaryk University. So far it can be used to verify some important MSC properties such as livelock, deadlock and trace race conditions. But it lacks the option of specifying time in designed systems, which is important in many protocols.

In standard Z.120 there are specified options how to include time in MSC. After specifying time there rise severe problems concerning time in MSC. There have been presented severe approaches how to address the problem of extension of time to MSC and problems regarding time in MSC in the literature . This thesis aims to analyze the different approaches and find the answer to problems regarding time, especially in the context of SCStudio.

The thesis is structured as follows. We first formally define MSCs, then specify criteria for finding ideal approach. Then we analyze already presented papers to handle time in MSC and pick the most suitable approach. Restriction to proper time constraints is given, which rules out ambiguous and erroneous timing constraints specifications. Finally extensions to the approach are provided and suggestions for next research are given.

Chapter 2

Preliminaries

According to ITU recommendation Z.120 standard [9], an MSC formalism consists of simple MSC and high–level Message Sequence Charts (HMSC). Both, MSC and HMSC, have textual and graphical notation. Instead of HMSC we use semantically equivalent MSG.

Through this thesis $\mathbb{N}$ will denote natural numbers with zero, $\mathbb{R}$ real numbers and $\mathbb{R}^+$ nonnegative real numbers.

2.1 Message Sequence Chart

An MSC consists of set of processes depicted as vertical lines, set of events on process lines and asynchronous communication depicted as arrows connecting two events. Event belongs to exactly one process and exactly one arrow. Set of events is partitioned into *send* and *receive* events. Arrow leaves from a send event and aims to a receive event. Send event is always ordered before its corresponding receive event and all process lines depict total order on their events except events in coregion. Coregions are sections on the process lines where the events are not ordered. Coregion is depicted as a rectangle on process line and events in the coregion are events on the rectangle. If one wants to enforce additional ordering of events in coregion, connections can be used. Connection a is dashed arrow leaving from the preceding event and aiming to the subsequent event.

The next two definitions are used from [8, 19].

Definition 1. *An* MSC *(with coregions and connections) is defined as a tuple* $(E, <, \mathcal{P}, \tau, P, \mathcal{M}, \mathcal{C}, \mathcal{G})$ *where*

- E *is a set of* events*;*
- $<$ *is a partial ordering on* E *called* visual order*;*
- $\mathcal{P}$ *is a finite set of* processes*;*
- $\tau : E \to \{s, r\}$ *is a labeling function dividing events into* send, receive*;*
- $P : E \to \mathcal{P}$ *is a mapping that associates each event with a process;*
- $\mathcal{M} \subseteq (\tau^{-1}(s) \times \tau^{-1}(r))$ *is a bijective mapping, relating every send with a unique receive, such that for any* $(e, f) \in \mathcal{M}$ *we have* $P(e) \neq P(f)$ *– a process cannot send messages to itself;*

- $\mathcal{C}$ *is a set of pairwise disjoint* coregions *where a coregion (say* $C \in \mathcal{C}$*) is defined as a subset of events on some process, i.e.* $C \subseteq P^{-1}(p)$ *where* $p \in \mathcal{P}$*;*
- $\mathcal{G} \subseteq \bigcup_{C \in \mathcal{C}} C \times C$ *is a partial ordering on events within coregions and is called* connections.

Visual order $<$ *is defined as the reflexive and transitive closure of*

$$\mathcal{M} \cup \mathcal{G} \cup \bigcup_{p \in \mathcal{P}} <_p \text{ where } <_p = (\prec_p \setminus \{(e_x, e_y) \mid e_x, e_y \in C \ \wedge \ C \in \mathcal{C}\})$$

where $\prec_p$ *is a total order on* $P^{-1}(p)$ *such that each coregion associated with* p *is a set of consecutive events wrt* $\prec_p$*. In other words,* $<_p$ *is a total ordering of events outside of the coregions and of the coregions as whole units; each two events within the coregions are pairwise unordered.*

Definition 2. *Given two MSCs* $M_1 = (E_1, <_1, \mathcal{P}, \tau_1, P_1, \mathcal{M}_1, \mathcal{C}_1, \mathcal{G}_1)$ *and* $M_2 = (E_2, <_2, \mathcal{P}, \tau_2, P_2, \mathcal{M}_2, \mathcal{C}_2, \mathcal{G}_2)$ *over a common process set* $\mathcal{P}$ *and with* $E_1 \cap E_2 = \emptyset$*, let the concatenation of* M_1 *and* M_2 *be the MSC* $M_1 \cdot M_2 = (E_1 \cup E_2, <, \mathcal{P}, \tau_1 \cup \tau_2, P_1 \cup P_2, \mathcal{M}_1 \cup \mathcal{M}_2, \mathcal{C}_1 \cup \mathcal{C}_2, \mathcal{G}_1 \cup \mathcal{G}_2)$ *where* $<$ *is a reflexive and transitive closure of* $<_1 \cup <_2 \cup \bigcup_{p \in \mathcal{P}} (P_1^{-1}(p) \times P_2^{-1}(p))$*.*

The concatenation of two MSCs simply glues two process lines corresponding to the same process, one after the other in the correct order. Please observe that some events from the second MSC can be executed before events in the first MSC.

2.2 High-level Message Sequence Chart

Definition is the same as in [4]

Definition 3. *An* HMSC *is defined as a tuple* $H = (N, B, v^I, v^T, \mu, E)$

- N *is a finite set of nodes.*
- B *is a finite set of supernodes.*
- $v^I \in N \cup B$ *is an initial node or supernode.*
- $v^T \in N \cup B$ *is a terminal node or supernode.*
- μ *is a labeling function* $\mu : N \cup B \rightarrow$ *MSC* $\cup$ *HMSC, labeling each node* $n \in N$ *by an MSC and each supernode* $m \in B$ *by an HMSC.*
- $E \subseteq N \cup B$ *is an edge relation on the nodes and supernodes.*

2.3 Message Sequence Graph

According to [2,17] instead of HMSC we will use semantically equivalent Message Sequence Graph (MSG). Next two definitions are from [19, 4].

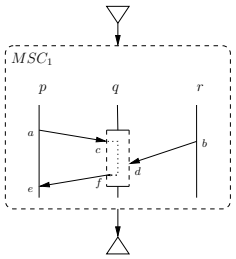

Figure 2.1: Example of an MSG

Definition 4. *An* MSC-graph *is a tuple* $G = (S, \rightarrow, s_0, s_f, L, \mathcal{L})$ *that consists of*

- *a finite set of states* S*,*
- *a transition relation* $\rightarrow \subseteq S \times S$*,*
- *an initial state* $s_0 \in S$*,*
- *a final state* $s_f \in S$*,*
- *a finite set of MSCs* $\mathcal{L}$*, and*
- *a mapping* L*, assigning to state* $s \in S \setminus \{s_0, s_f\}$ *an MSC* $L(s)$ *over set of MSCs* $\mathcal{L}$*.*

Definition 5. *Let* G *be an MSG,* $G = (S, \rightarrow, s_0, s_f, L, \mathcal{L})$*. A sequence of states* $s_1 s_2 \dots s_k$ *is a* path*, if* $(s_i, s_{i+1}) \in \rightarrow$ *for every* $1 \leq i < k$*. A path is a* run*, if* $s_1 = s_0$ *and* $s_k = s_f$*.*

We will call the states of the MSG also vertices and nodes. In addition to MSC, MSG allow us to specify infinite and alternative communication. MSG is simply a directed graph with specified initial and terminal vertex. Each state which is not initial or final represents one MSC. All states have to be reachable from the initial state, and the final state has to be reachable from all states. MSG represents a set of MSCs which are made by concatenating MSCs along runs or infinite paths from the initial state. That is why if we have two alternative paths between initial and final vertex they represent alternative executions. Due to possibility of specifying cycles, the MSG can represent infinite set of MSCs.

In Figure 2.1 we have an example of MSG. It contains just three states: initial, final and state representing MSC MSC_1. This MSG represents only one MSC. MSC MSC_1 contains three processes p, q, r and six events $a, \dots, f$. Events a, e belong to process p and there is total order specified on these events. Event on the top precedes events bellow, i.e. $a < e$. A coregion is specified on the process q, which contains events c, d, f. A connection between events c and f is specified. Thus events c, f are ordered, $c < f$, but there is no ordering between c, d and d, f.

Chapter 3

Introduction to the Problematics

There have been many papers published using various approaches how to address the problem of extension of MSC by time. We would like to find the ideal approach. In the beginning it is important to choose the right criteria for finding the ideal approaches from published papers. First we have to know what we want to include into the time extension to MSCs, and what problems we want to be able to decide. We will see that most of the suggestions follow directly from Z.120 standard.

When coming to the time extension we first have to be able to include time in the system specifications in MSC. *Timers* and *time intervals* are introduced for this purpose in Z.120 [9].

3.1 Timers

In the MSC specification timer is incorporated by 4 types of special events: *start, restart, stop* and *timeout*. The timer can be set to a value and started by *start*, stopped by *stop*, restarted by *restart* and observed for *timeout*. Timers are local to a single process and their events can be spread through various nodes in MSG. For more detailed description please refer to the standard [9].

In Figure 3.1 the drawing rules for the timers are specified.

3.2 Time Intervals

We first provide the definition of *interval set*.

Definition 6. Interval set $\mathcal{I}$ *is a union of all possible nonnegative intervals of the form*

$$(a, b), \langle a, b\rangle, (a, b\rangle, \langle a, b),$$

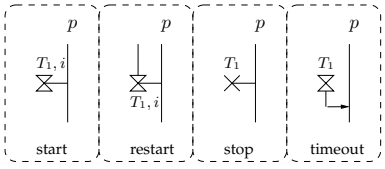

Figure 3.1: Timer events

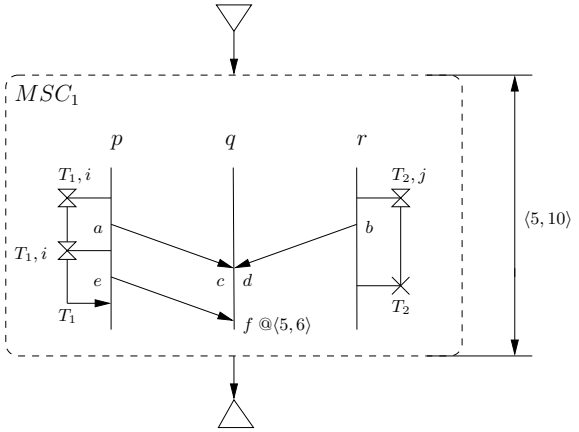

Figure 3.2: Timing constraints

where $a, b \in \mathbb{N} \wedge a \leq b$ i.e. open, closed and half-open nonnegative intervals.

Time *interval constraints* can be described at the MSC and MSG level. They incorporate one or two *points* and a union of intervals from $\mathcal{I}$. Point for the MSC is an event, for MSG it is the begining or the end of a state from the state set. If the interval constraint is specified on a begining of the vertex it denotes constraint on the first executed event from the MSC, that the vertex represents. If the interval constraint is specified on the end of a vertex it denotes constraint on the last executed event from the MSC, that the vertex represents.

There are two types of intervals: *absolute* and *relative*. Absolute intervals denote constraint on the time difference between the start of the system and an execution of point. Relative intervals represent time constraint on the difference of time between execution of two points. In the relative interval the order of points is not explicitly given, but the first point is derived dynamically from the structure of the MSC/MSG. In constraint we allow only finite number of intervals with nonnegative values i.e. the lowest lower border among all intervals in the interval constraint is greater or equal to zero.

We denote *timing constraints* as interval constraints and timers. Through this thesis all timing constraints are meant as restrictions for the design.

Imagine MSC MSC_1 in Figure 3.2. There are two timers T_1, T_2 specified. Firstly the timer T_1 is started, then it waits for execution of event a. After a is executed timer T_1 is restarted and timeouts after i time units. Timer T_2, after it is started, waits for an execution of event b. After execution of event b the timer stops. Event b has also absolute time constraint denoted by prefix @. It constraints the execution of event b to be within $\langle 5, 6 \rangle$ time units after the start of the system. Relative interval constraint, with no prefix, is specified between the begining and the end of MSC_1. That is why it specifies the constraint on difference of time between execution of the last and the first event from MSC_1 to be in $\langle 5, 10 \rangle$.

3.3 Definition of MSC and MSG with Timing Constraints

In this work we introduce various restrictions on specifications of intervals. For the sake of simplicity we forbid absolute intervals. In the MSC we can easily substitute them using relative time intervals, if we make an unique start event of the MSC. In the HMSC the semantics of the time interval constraints is not fully defined. There arise severe meanings and ambiguities when coming to specification of the absolute time interval constrains in an MSG. From now on we will denote relative intervals constraints as the *interval constraints* and the interval constraints and timers as *timing constraints*.

Definition 7. *An* MSC *(with timing constraints) is defined as a tuple* $(E, <, \mathcal{P}, \tau, P, \mathcal{M}, \mathcal{C}, \mathcal{G}, \mathcal{K}, \mathcal{T})$ *where all symbols in the tuple except* τ, $\mathcal{K}$ *and* $\mathcal{T}$ *denote the same as for general MSC and*

- $\mathcal{T}$ *is set of timer labels*
- $\tau : E \to \{s, r, start(t,i), stop(t), restart(t,i), timeout(t)\}$ *is a labeling function dividing the events into* send, receive *and the timer events where* $t \in \mathcal{T}$ *and* $i \in \mathbb{N}$
- $\mathcal{K} \subseteq \{(\{a,b\}, d) \mid a, b \in E, a < b, d \in 2^{\mathcal{I}}\} \cup \{(c, d) \mid c \in \mathcal{C}, d \in 2^{\mathcal{I}}\}$

We allow both time interval constraints and timers, because the timer has a slightly different semantics than the interval constraint. We also restrict the specification of interval constraints in the MSC only to pairs of events ordered via visual order. We need to be able to decide the order of the events in MSC because the order of events should not be specified by the constraint, according to the standard Z.120. We also allow the specification of interval constraint on the length of the coregion which constraints the difference of time between the execution of the first and the last event from the coregion to be in the specified interval.

We redefine concatenation of MSCs with timing constraints.

Definition 8. *Given two MSCs* $M_1 = (E_1, <_1, \mathcal{P}, \tau_1, P_1, \mathcal{M}_1, \mathcal{C}_1, \mathcal{G}_1, \mathcal{K}_1, \mathcal{T}_1)$ *and* $M_2 = (E_2, <_2, \mathcal{P}, \tau_2, P_2, \mathcal{M}_2, \mathcal{C}_2, \mathcal{G}_2, \mathcal{K}_2, \mathcal{T}_2)$ *over a common process set* $\mathcal{P}$ *and with* $E_1 \cap E_2 = \emptyset$*, let the concatenation of* M_1 *and* M_2 *be the MSC* $M_1 \cdot M_2 = (E_1 \cup E_2, <, \mathcal{P}, \tau_1 \cup \tau_2, P_1 \cup P_2, \mathcal{M}_1 \cup \mathcal{M}_2, \mathcal{C}_1 \cup \mathcal{C}_2, \mathcal{G}_1 \cup \mathcal{G}_2, \mathcal{K}_1 \cup \mathcal{K}_2, \mathcal{T}_1 \cup \mathcal{T}_2)$*, where* $<$ *is a reflexive and transitive closure of* $<_1 \cup <_2 \cup \bigcup_{p \in \mathcal{P}} (P_1^{-1}(p) \times P_2^{-1}(p))$.

Definition 9. *An* MSC-graph *(with timing constraints) is a tuple* $G = (S, \to, s_0, s_f, L, \mathcal{L}, \mathcal{K}_{MSG}, \mathcal{T}_{MSG})$ *where all symbols in the tuple except* $\mathcal{K}_{MSG}$ *and* $\mathcal{T}_{MSG}$ *denote the same as for general MSG and*

- *a finite set* $\mathcal{K}_{MSG} \subseteq \{((a, \overline{\uparrow}), (c, \underline{\downarrow}), d) \mid a, c \in S, d \in 2^{\mathcal{I}}\} \cup \{((a, \underline{\downarrow}), (c, \overline{\uparrow}), d) \mid a, c \in S \,.\, a \to c, d \in 2^{\mathcal{I}}\}$
- *a finite set* $\mathcal{T}_{MSG}$ *consists of*

$$\mathcal{T}_{MSG} = \bigcup_{M \in \mathcal{L}} \{\mathcal{T} \mid M = (E, <, \mathcal{P}, \tau, P, \mathcal{M}, \mathcal{C}, \mathcal{G}, \mathcal{K}, \mathcal{T})\}$$

$\overline{\uparrow}$ denotes the begining of a vertex in MSG and $\underline{\downarrow}$ denotes the end of a vertex in MSG. The formal semantics of the time interval constraint for the HMSC is not fully defined in Z.120. We restricted the options of defining time intervals to be able to decide which point from the constraint is executed first.

We allow the specification of the interval constraints in MSG only between the beginning of one vertex and the end of another while the vertex with constraint on the beginning of the vertex has to be executed first. We also allow the specification of the interval constraint between the end of a vertex and the beginning of another one, while the second vertex must be an immediate successor of the first one i.e. there has to be an edge from the vertex with constraint on the end to the vertex with the constraint on the beginning. This type of constraint rules out the interleaving of two successive MSCs in the MSG. We believe these restrictions do not significantly decrease the expressive power of the time constraints. There is still a possibility to use the timers on places where we disallowed the interval constraints.

3.4 Problems on MSCs with Timing Constraints

Given the MSC with timing constraints, it is natural to ask whether there exists at least one execution of the system which satisfies all the intervals and timers in the MSC. We will call this problem *timing consistency*.

Next definition is inspired by [3].

Definition 10. *We define a* timing assignment ι *for an MSC* $M = (E, <, \mathcal{P}, \tau, P, \mathcal{M}, \mathcal{C}, \mathcal{G}, \mathcal{K}, \mathcal{T})$ *to be a function* $\iota : E \to \mathbb{R}^+$ *that assigns each event* $e \in E$*, a time stamp* $\iota(e)$ *so that it satisfies every constraint, i.e. for every interval constraint* $(\{a, b\}, d) \in \mathcal{K}$. $a < b$, $\iota(b) - \iota(a) \in d$ *and for every timer* $t \in \mathcal{T}$ *and two events* $a, b \in \{start(t, i), restart(t, i), timeout(t), stop(t)\}, i \in \mathbb{N}$ *either*

- $\iota(b) - \iota(a) = i$*, where* $\tau(b) = timeout(t)$, $\tau(a) \in \{start(t, i), restart(t, i)\}$ *and between* a *and* b *are no* $start(t, i)$ *or* $restart(t, i)$ *timer* t *events, or*
- $\iota(b) - \iota(a) \leq i$*, where* $\tau(b) \in \{stop(t), restart(t, i)\}$ *and* $\tau(a) \in \{start(t, i), restart(t, i)\}$ *and between* a *and* b *are no* $start(t, i)$ *and* $restart(t, i)$ *timer* t *events.*

Definition 11. *A* Timing assignment ι_{MSG} *for a run* $s_0 \dots s_f$ *or infinite path* $s_0.r_1 \dots$ *in MSG* $G = (S, \to, s_0, s_f, L, \mathcal{L}, \mathcal{K}_{MSG}, \mathcal{T}_{MSG})$ *where MSC* $M = (E, <, \mathcal{P}, \tau, P, \mathcal{M}, \mathcal{C}, \mathcal{G}, \mathcal{K}, \mathcal{T})$ *and* $M = L(s_0) \dots L(s_f)$ *or* $M = L(s_0).L(r_1) \dots$ *is a function* $\iota_{MSG} : E \to \mathbb{R}^+$ *which is timing assignment for* M *and for every constraint* $k \in \mathcal{K}_{MSG}$

- $\iota_{MSG}(e') - \iota_{MSG}(e) \in d$*, where* $k = ((a, \overline{\uparrow}), (c, \underline{\downarrow}), d)$*,* e *is the first event of* $L(a)$ *according to* ι_{MSG} *and* e' *is the last event of* $L(c)$ *according to* ι_{MSG}
- $\iota_{MSG}(e') - \iota_{MSG}(e) \in d$*, where* $k = ((a, \underline{\downarrow}), (c, \overline{\uparrow}), d)$*,* e *is the last event of* $L(a)$ *according to* ι_{MSG} *and* e' *is the first event of* $L(c)$ *according to* ι_{MSG}*.*

This way the timing assignment gives the possible times in which events may occur. Another definition from [3] follows.

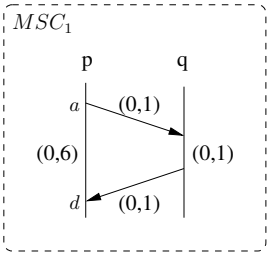

Figure 3.3: Tightening

Definition 12. *MSC M is* timing consistent *if there exists at least one timing assignment ι for M.*

Next definition was changed from [5].

Definition 13. *MSG G is* timing consistent *if for every path $p = s_0 \dots s_f$ or $p = s_0 \dots$, i.e. runs or infinite paths from initial vertex, has a timing assignment ι_{MSG}.*

A motivation for another problem is as follows: Given the timed MSC/MSG with timing intervals, is there a tighter description of the same system such that we accurately tighten the intervals to get stricter constraints? Imagine an MSC depicted by Figure 3.3. There are four time intervals. It is easy to see that events a and d cannot have the difference between execution times more than 3, but the interval $\langle 0, 6\rangle$ allows the difference to be 6. So we can tighten the upper bound of the interval $\langle 0, 6\rangle$ to 3 and we do not change the behavior of system. Through literature this problem is widely referred to as computing of *minimal network*. We will also refer to it as tightening and computing minimal MSC/MSG.

For a the formal definition of the tightening we have to define a *solution set*. The remaining definitions from this chapter were inspired by [7].

Definition 14. *A* Solution set *for an MSC M is a set, which consists of all timing assignments for M.*

Definition 15. *A* Solution set *for an MSG G is a set, which consists of all pairs (r, t) where r is a run or infinite path from initial node in G and t is possible timing assignment for r in G.*

In the following definition we define the partial order on intervals, constraints and MSCs.

Definition 16. *Interval $I \preceq J$ for $I, J \in \mathcal{I}$ iff*

$$\forall x \in I \Rightarrow x \in J.$$

A partial order among interval constraints can be defined as follows. Given the interval constraints T, S, T is tighter than S, denoted $T \preceq S$, if for every interval $I \in T$ there exists interval $J \in S$ such that $I \preceq J$. Let M, N be an MSC/MSG that differ only by values of interval constraints. M is tighter than N, denoted $M \preceq N$, if the partial order is satisfied for all the corresponding interval constraints; namely for every interval constraint S from M there is a corresponding interval

constraint N such that $T \preceq S$. Two MSC/MSG are equivalent *if they represent the same solution set.*

Definition 17. Minimal MSC/MSG *is a minimal from all equivalent MSC/MSG according to $\preceq$.*

Definition 18. *The* tightening *or finding* minimal network *of MSC/MSG is a problem of finding minimal interval constraints such that for interval J in constraint A. $J \in A$, we can tighten the interval to interval I, where $I \preceq J$ such that we keep the same solution set of MSC/MSG.*

Another problem is computing measurements. Measurements in Z.120 standard are variables of type *time* which could take position of interval in a constraint. The problem is to find most restrictive value of the variable such that we do not restrict the system any more, i.e. keep the same solution set. The motivation is to get possible difference between the execution times of the events.

Many works dealing with problematics of *race conditions* have been published. Informally two events in the MSC are in race if one precedes another according to the visual order, but in fact these events can occur in the opposite order. Most of the works aim on detecting race conditions and avoiding them in the system design in the MSC formalism using the *coregions*. The works try to find events that are in race and put these events in coregion so that the specification is correct. The extension of time in MSCs can reduce the number of event pairs in race. So the problem would be to find the valid race conditions in timed MSC/MSG.

By research through already presented works including time and MSCs, we were able to find only the problems listed above. There are also some minor problems, which are interesting, but they are subproblems of the above–mentioned problems. So we can use the above–mentioned problems the criteria for finding the ideal approaches to time extension of MSCs.

Actually we can reduce the problem of measurements to a problem of computing minimal network. For each measurement we can introduce an interval constraint $\langle 0, \infty)$ and find the minimal network. It is easy to see that the final interval constraint is the measurement value.

Finding race conditions is also reducible to tightening the network to minimal network. Imagine we have just two events e, e' which are in race. We can find them using the algorithm for finding trace race conditions proposed in [19]. We can then run the tightening algorithm two times: for original MSC extended with most generous time interval constraint $(\{e, e'\}, \langle 0, \infty))$ for $e < e'$ and for $e' < e$. If we gain consistent network for both MSCs we know that these events remain in race. The extension of previous approach to multiple events in race is not hard but space and time consuming thus we omit the construction.

That is why our criteria for finding the ideal approaches are:

- capability to decide timing consistency,
- capability to compute minimal network.

Chapter 4

Research Through Presented Works

There have been published many papers using various approaches to our problems. The main approaches are based on automata [15, 6, 1], Petri nets [14], lposets [4, 5, 21] and linear programming [13].

4.1 Automata Approach

Most of the works using this approach [15, 6] translate MSC/MSG into automata, mostly *timed automata*. Then they try to solve problems using known techniques on automata and programs such as UPPAAL [12]. The main drawback of this approach is that we restrict the language of MSC to regular MSC languages. These languages correspond to class of *bounded MSC or MSG* see [4, 10].

In [15] authors are interested in problem of checking that implementation behaves according to the specification. They use Timed sequence diagrams specification, which is similar to MSC. Timing marks are used which are equivalent to relative and absolute time intervals. They convert specification in *timed sequence diagrams* to timed automata and control it against specification, which is already given as timed automata. Authors simulate concurrent run through both automata models and synchronize them every time the state of channels is changed. For every wrong behavior transitions to sink states are made, one for timing errors, one for other errors. They can check the model for *optional* and *mandatory* behavior via different formulae which UPPAAL can solve.

Useful features are idea of specifying optional and mandatory cases for specification and options of specifying constraints in loops. They introduce a flag which specifies in which iteration the constraint is valid (next, first, last). They also have constraints on number of iterations of loop. Authors assume only MSCs.

Another work using automata approach is [6]. Authors use both MSC and MSG. MSCs are with relative time intervals on causal order which is subset of visual order. They do not use time intervals between nodes in an MSG. *Timed Message passing automaton* (TMPA) is defined, which is similar to timed automata. They can convert timed MSG to TMPA and they define two properties which they want to check: *scenario matching* and *universality*.

Definition 19. *For timed HMSC H and TMPA A* scenario matching, *is deciding whether*

$$\exists\, timed\, MSC\; B \in H\; .\; L(B) \cap L(A) \neq \emptyset.$$

Definition 20. *For timed HMSC H and TMPA A* universality*, is deciding whether*

$$\forall\, timed\, MSC\, B \in H\, .\, L(B) \cap L(A) \neq \emptyset.$$

They can check scenario matching in UPPAAL but they can check universality only by restriction to finite set of runs. There was no contribution to our problems.

Authors in [1] give constructive proof that universality from above [6] is decidable for bounded MSG. They also provide extension of the proof to arbitrary timed MSG but in every time they restrict themselves to only one node in MSG, what is easier problem. They allow relative time interval constraints between two events in MSC and for each process on an edge in MSG.

The automata approach papers try to solve different problem. They look at the system specified in MSC formalism as a specification which should be used to test the implementation. The main problem is to detect whether both implementation and specification are doing the same. However we look at the system specification only as a design. We would like to solve the problems regarding only the design, i.e. the correctness of MSC/MSG. SC-Studio is design and verification tool. So far it cannot control implementation against specification. The papers do not give us answers to our problems and the extension of their work could be hard. Also if we would like to use this approach we would probably have to restrict ourselves only to regular subclass of MSC languages [4, 10].

4.2 Petri Net Approach

Petri net approach was presented in [14]. Authors use MSC and define time constraints which are stronger than interval constraints in standard Z.120. They define *Timed Petri Nets* (TPN), *satisfaction problem* on TPN and give algorithm to solve it. Scenario matching is a property, that a given TPN N whenever scenario described by a given MSC D appears in N, the corresponding segment of a run must satisfy the time constraints enforced by D. They use timed automata to control scenario matching.

The authors also address the problem of testing the implementation in Petri nets, against the specification in MSC. This paper does not give answers to our problems. The extension to our problems would be rather problematic. Authors use only MSC with more expressive timing constraints. But we do not need more expressive timing constraints since we want to follow the standard. The more expressive constraints could be also drawback in MSG concept.

4.3 Lposet Approach

Lposet stands for labeled partially ordered set. This structure is most closely related to MSC formalism and most of the authors use it. Using this approach authors use lposets to describe the behavior of MSC.

In [3] authors give one of the earliest and most straightforward approaches. They define timed MSC with a single relative interval per constraint. The constraints are relative intervals or timers. They define *timing consistency* for MSC and provide procedure how to check in polynomial time whether the network is timing consistent. They convert the problem of solving timing consistency to finding negative cost cycles in a weighted digraph what is solvable in $O(n^3)$ time. But there is no extension given for MSG.

In [5] authors describe various options of time constraint definition. They decide to use relative time intervals and timers. They use MSG structure and have special events for the beginning and the end of the process in every MSC. Authors provide examples of ambiguous timer specifications and restrict the timer specification to unambiguous. They define *timing consistency* for MSC and MSG and give an algorithm to test *timing consistency* in *MSC specification* which is similar to MSG. The main idea is to finitely encode the infinite paths in MSG from initial state using observation that we do not need more than one unfolding of a cycle. For every such path and every run they concatenate MSCs along the path and execute consistency algorithm from [3] on created MSCs.

Despite this approach is very nice, we are loosing most of the restrictions on the system, which we need to appropriately tighten the intervals and find measurements. The approach also works only on restricted version of the problem because authors are using single intervals instead of unions of the intervals.

In [21] authors provide similar approach as provided in [5]. They use MSC and MSG with relative and absolute time interval constraints with only one interval per constraint. Relative time constraints in MSC are specified only between casually ordered events. Authors provide algorithms in pseudocodes which can decide timing consistency and weak timing consistency for MSG. Weak timing consistency of MSG is deciding whether there is a run or infinite path in MSG which is timing consistent. They do not have constraints between nodes of MSG and that is why their algorithms work correctly. When trying to add constraints algorithms, do not work. They use special procedures from [7] to speed up the process of checking consistency for MSC.

In [7] there are presented algorithms to efficiently tighten interval constraints in many classes of problems. In this work there are structures similar to ours – digraphs labeled by unions of relative time constraints. They call these structures *networks*. They address problems of finding minimal networks and deciding timing consistency.

Approach of lposets seems to be most advanced in the research of time in MSC and MSG. The authors mostly try to solve the problem of *timing consistency*. Some authors can check the property quite successfully, but not for such generous problem as we would like. But the extension does not seem to be hard. Listed approaches differ mostly by restrictions to specifications of timing constrains. Big amount of work regarding tightening is already done in [7]. We would just have to extend their approaches to the setting of MSC with coregions and MSG.

4.4 Linear Programming

In [13] authors use timing marks which have more expressive power than interval constraints. The order of events is explicitly given. There are no constraints between vertices in MSG except of special events for starting and terminating the processes in MSG. They want to find finite representation of behaviors of MSC specification, which is similar to MSG. For this purpose they use normalized regular expressions to specify set of paths through MSC specification. They provide algorithm for deciding timing consistency and inconsistency which works similar to [5,21]. This approach could be very effective in computing measurements and deciding timing consistency. However, linear programming is a CSP (constraint satisfaction problem), as well as the problem of computing minimal networks see [7]. So that the problems are very similar, but computing minimal networks in [7] is much closer to MSC formalism and to answers to our problems. It is also very hard to predict how difficult the tightening of unions of the intervals would be.

4.5 Choosing the Approach

From presented works we see that most of the work regarding to our problematics was done in lposet approach. But the presented works do not give full solutions to our problems. There are missing constructs for tightening MSG and MSC with coregions. However there are already good foundations presented [21,7] on which we can build the extensions. Other approaches are not as advanced in deciding timing consistency and finding minimal network. That is why we suggest for the next research the approach of lposets.

Chapter 5

Proper Time Constraints

In this chapter we first formally define timing constraints which are cleaned of ambiguous and erroneous constraints. We will call them *proper*. Then we will provide algorithms to check whether MSC and MSG contains only proper constraints.

So far we can specify such constraints, for which it is hard to say, what they mean. For instance suppose MSG in Figure 5.1.

The time interval t is ambiguous, because we do not now whether it is specified between $B.1$ and $B.2_i$ where $i \in \mathbb{N}$ is iteration number of the cycle. The constraint t could mean the constraint between $B.1$ and the first iteration $B.2$ or last iteration of $B.2$ or any other iteration. Timeout event specification in node $B.3$ is erroneous because we observe timer T for timeout but it was not set.

There have been introduced definitions of ambiguity in [5,11]. However their definitions do not affect case of MSG in Figure 5.1. That is why we provide our definitions of *proper* time constraints, which should rule out erroneous and ambiguous time constraints. Our definitions are also in conformance with Z.120 standard. We provide some checking algorithms which detect interval constraints which are not proper.

5.1 Definitions

Definition 21. *Let $G = (S, \rightarrow, s_0, s_f, L, \mathcal{L}, \mathcal{K}_{MSG}, \mathcal{T}_{MSG})$ be MSG and $f = ((a, \overline{\uparrow}), (c, \underline{\downarrow}), i) \in \mathcal{K}_{MSG}$ be an interval constraint where $a, c \in S$ and $i \in 2^I$. We say that constraint f is proper iff*

- *there is no path r through G such $r = s_0.r_1.r_2...r_n.c$, where $r_i \neq a$ for any $i \in 1, 2, ..., n$*
- *in any path $r = s_0.r_1.r_2...$ in G if $r_i = r_j = a$ for $i, j \in \mathbb{N}$ and there is no $r_k = a$ for $i < k < j$ there must be some l. $i \leq l < j$ such that $r_l = c$.*
- *in any path $r = s_0.r_1.r_2...$ in G if $r_i = r_j = b$ for $i, j \in \mathbb{N}$ and there is no $r_k = b$ for $i < k < j$ there must be some l. $i < l \leq j$ such that $r_l = c$.*
- *there is no paths r through G such $r = s_0.r_1.r_2...r_n.s_f$, where $r_i = a$ and there is no j . $i \leq j$, $i, j \in 1, 2, ..., n$ such $r_j = b$.*

Any constraint $((a, \underline{\downarrow}), (c, \overline{\uparrow}), i) \in \mathcal{K}_{MSG}$ with $(a, c) \in \rightarrow$ is proper.

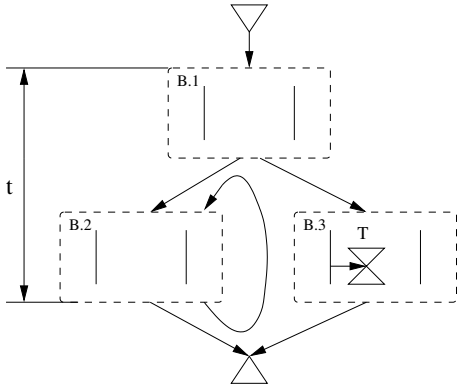

Figure 5.1: Ambiguity

The first and the last condition disallow incomplete intervals. The second condition disallows two a states in a row without no b state between them. The third condition similarly disallows two b states in a row without no a state between them.

Definition 22. *Let $G = (S, \rightarrow, s_0, s_f, L, \mathcal{L}, \mathcal{K}_{MSG}, \mathcal{T}_{MSG})$ be MSG and $t \in \mathcal{T}$ be a timer. We say that timer t is proper iff*

- *in every run r timer $start(t, i)$ event must be executed first among timer t events for some $i \in \mathbb{N}$*
- *after $stop(t)$ or $timeout(t)$ event only $start(t, i)$ event can occur from timer t events for some $i \in \mathbb{N}$*
- *after $start(t, i)$ only $stop(t)$, $restart(t, j)$ or $timeout(t)$ can be executed from timer t events for some $i, j \in \mathbb{N}$*
- *the $start(t, i)$ or $restart(t, i)$ event cannot be executed last among timer t events in any run in G, for some $i \in \mathbb{N}$.*

The first condition controls that in any run in MSG the timer has to be first started before stopping, resetting or observing for timeout. The second and the third condition check the appropriate order of timer events in any run through MSG. The third condition enforces that timer has to be stopped or timed out before the end of the run.

5.2 Checking Algorithms

We will provide two algorithms for checking whether constrains in MSG are proper. Both are similar, one is for interval constraints and one is for timers. Both algorithms are just to illustrate the idea of checking the conditions. They can be optimized easily, but they would not be as intuitive.

It is easy to see that both algorithms terminate. We iterate through the interval set which is finite and use DFS extended by finite commands, which is also known of terminating. Both algorithms simply traverse the graph and in every node they check whether the constraint is active or not. If the constraint is active, it cannot be activated again and if the constraint is not active it cannot be terminated. This corresponds to the second and the third condition in both definitions of proper constrains. The algorithms store the information in the variable on. For each vertex we also keep the information about the value of on when finding the vertex. We store it to be able to check that values of on from two different paths to the vertex conform. If they do not conform we know that the constraint is not proper, because we found the same node with different paths and if the first path was correct, the second one will be trivially wrong. So that when we find the already found vertex v (it is not white), we check whether $v.on = on$. When finding the node v for the first time we assign the value of $v.on$ to the current value of on. We also update on according to the current node, where we check that when $on = 0$ constraint can be only activated and when $on = 1$ constraint can be only terminated. Both algorithms use slightly different approaches in updating on. Updating for the Algorithm 1 is trivial. For the Algorithm 2 $updateOn(t, v, on)$ simply traverses events in MSC $L(v)$ according to its visual order $<$ and while processing event behaves according to the Table Update On 5.1. If some of the timer events in MSC are not ordered via the visual order we update and check on for each traverse of MSC via the updated visual order. We create updated visual order for each permutation of unordered timer events and $<$. If the MSC is correct in the end we should get the same value of on or we throw exception. For simplicity we have not included catching the exception in the main Algorithm 2.

In the beginning $on = 0$ and that forces the constraint to be activated first. This is exactly what both definitions for proper constraints in first conditions require. Similarly we check whether $on = 0$ when reaching final state. This enforces the constraint to be terminated in the end of run. Again this is exactly what fourth conditions in both definitions require.

For simplicity, we have not provided an extension of Algorithm 1 for constraints from end of the node to the beginning of the node, i.e. $((a, \underline{\downarrow}), (c, \overline{\uparrow}), i)$ for some $a, c \in S$. The idea is straightforward: we just check whether a and c are in $a \rightarrow c$, which can be done in linear time.

We can speed up the algorithms. Instead of running DFS for every constraint we run single DFS for all constraints. We will have to have an array of boolean variables on for each vertex in MSG and one for the algorithm. All of the arrays will be of length n, where n is the number of constraints.

Algorithm 1 Check whether Interval Constraints in MSG $G = (S, \rightarrow, s_0, s_f, L, \mathcal{L}, \mathcal{K}_{MSG}, \mathcal{T}_{MSG})$ are Proper

for all $f = ((a, \overline{\uparrow}), (c, \underline{\downarrow}), i) \in \mathcal{K}_{MSG}$ **do**
  $on \longleftarrow 0$;
  run DFS in G from s_0 such that the main procedure while processing node v will be extended by following code:
  if $v = s_f \wedge on = 1$ **then**
    {interval constraint started but did not end before reaching s_f}
    return $currentDFSpath$
  else if v is white **then**
    $v.on \longleftarrow on$
    if $a = v$ **then**
      if $on = 1$ **then**
        {interval constraint f started two times in a row}
        return $currentDFSpath$
      else
        $on \longleftarrow 1$
      end if
    else if $c = v$ **then**
      if $on = 0$ **then**
        {interval constraint f ended two times in a row}
        return $currentDFSpath$
      else
        $on \longleftarrow 0$
      end if
    end if
  else if $v.on \neq on$ **then**
    { interval constraint f is not proper }
    return $currentDFSpath$
  end if
end for

Algorithm 2 Check whether Timers in MSG $G = (S, \rightarrow, s_0, s_f, L, \mathcal{L}, \mathcal{K}_{MSG}, \mathcal{T}_{MSG})$ are Proper

for all $t \in \mathcal{T}$ **do**
  $on \longleftarrow 0$;
  run DFS in G from s_0 such that the main procedure while processing node v will be extended by following code:
  if $v = s_f \wedge on = 1$ **then**
    {timer was set and reached s_f}
    return $currentDFSpath$
  else if v is white **then**
    $v.on \longleftarrow on$
    $on \longleftarrow updateOn(t, v, on)$
  else if $v.on \neq on$ **then**
    { timer t is not proper }
    return $currentDFSpath$
  end if
end for

Table 5.1: Update On

event	on	action
start	0	$on \longleftarrow 1$
start	1	throw exception
stop	0	throw exception
stop	1	$on \longleftarrow 0$
restart	0	throw exception
restart	1	$on \longleftarrow 1$
timeout	0	throw exception
timeout	1	$on \longleftarrow 0$

Chapter 6

Tightening

In this chapter we describe our approach to solve tightening. It will be based on computing minimal network in *temporal constraint satisfaction problem* (TCSP) from [7]. For the sake of simplicity we use only closed intervals. The extension to open intervals is not hard.

Since we have more constructs in MSCs and MSGs which have specific meaning e.g. coregions, we need to extend the approach of tightening of TCSP from [7]. The approach is based on three operators: *union* $\cup$, *intersection* $\cap$ and *plus* $\oplus$ (in [7] was intersection marked as $\oplus$ and plus as $\otimes$, however we changed it to $\cap$ and $\oplus$, because we think it is more intuitive to use $\cap$ as intersection and $\oplus$ as plus).

6.1 Old Operators

All operators are binary and work on unions of time intervals:

$$\cup, \cap, \oplus \subseteq 2^{\mathcal{I}} \times 2^{\mathcal{I}} \to 2^{\mathcal{I}}.$$

Next definition from [7] is used.

Definition 23. *Let $T = \{t_1, \ldots, t_m\}$ and $H = \{h_1, \ldots, h_n\}$ be constraints, $T, H \in 2^{\mathcal{I}}$.*

- *The* union *of T and H, denoted by $T \cup H$ admits only values that are allowed by either one of them, namely,*
$$T \cup H = \{t_1, \ldots, t_m, h_1, \ldots, h_n\}.$$
- *The* intersection *of T and H, denoted by $T \cap H$, admits only values that are allowed by both of them, namely,*
$$T \cap H = \{k_1, \ldots, k_p\},$$
where $k_q = t_i \cap h_j$ for some $i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\},\ q \in \{1, \ldots, p\}$.
- *The* composition *of T and H, denoted by $T \oplus H$, admits only values r for which there exist $t \in T$ and $h \in H$, such that $t + h = r$, namely,*
$$T \oplus H = \{k_1, \ldots, k_p\},$$
where $k_q = \langle a + c, b + d \rangle$ for some $t_i = \langle a, b \rangle, h_j = \langle c, d \rangle, i \in \{1, \ldots, m\},\ j \in \{1, \ldots, n\}$, $q \in \{1, \ldots, p\}$. Note that $p \leq n \times m$.

6.2 New Operators

We need to handle coregions and time intervals restricting whole MSC from its beginning to its end. For this purpose we define operators *max* and *co-max*.

In Z.120 standard the time interval constraint from the beginning to the end of MSC is a constraint the time difference between execution of the first event and execution of the last event. That is why if we need to restrict the time interval constraint on the whole MSC we need to compute the maximal time difference between the possible first events to possible last events. This will be provided by the *max* operator.

The *co-max* operator will provide us the tightening interval for interval constraint between any two points in MSC according to constraint on the whole MSC and other constraints in the MSC.

Definition 24. *Let $T = \{t_1, \ldots, t_m\}$ and $H = \{h_1, \ldots, h_n\}$ be constraints, $T, H \in 2^{\mathcal{I}}$.*

- *The* max *of T and H, denoted by $T\ max\ H = max(T, H)$*

$$max(T, H) = \{k_1, \ldots, k_p\},$$

where $k_q = \langle max\{a, c\}, max\{b, d\}\rangle$ for some $i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}, q \in \{1, \ldots, p\}$, $t_i = \langle a, b\rangle,\ h_j = \langle c, d\rangle$.

- *The* co-max *of T and H, denoted by $T\ co-max\ H = co-max(T, H)$*

$$co-max(T, H) = \{k_1, \ldots, k_p\},$$

where

$$k_q = \begin{cases} t_i & \text{if } t_i \cap h_j = \emptyset \\ \langle 0, b\rangle & \text{otherwise} \end{cases}$$

for some $i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}, q \in \{1, \ldots, p\}, t_i = \langle a, b\rangle$.

Following definition was used from [10].

Definition 25. *Communication graph G of MSC $M = (E, <, \mathcal{P}, \tau, P, \mathcal{M}, \mathcal{C}, \mathcal{G}, \mathcal{K}, \mathcal{T})$ is a tuple $(\mathcal{P}, \longmapsto)$, where*

- *$\mathcal{P}$ is set of processes of M*
- *$\longmapsto \subseteq \mathcal{P} \times \mathcal{P}$ where $a, b \in \mathcal{P}\ .\ a \longmapsto b$ iff $\exists e, e' \in E\ .\ \mathcal{M}(e) = e' \wedge P(e) = a \wedge P(e') = b$.*

We say that the communication graph is weakly com-connected *if it contains one connected component and isolated vertices.*

Next definition will be useful in remaining part of thesis. It was inspired by [19].

Definition 26. *Let $M = (E, <, \mathcal{P}, \tau, P, \mathcal{M}, \mathcal{C}, \mathcal{G}, \mathcal{K}, \mathcal{T})$ be an MSC. We set:*

- $Min(M) = \{e \in E \mid \forall e' \in E \,.\, e' \not< e\}$
- $Max(M) = \{e \in E \mid \forall e' \in E \,.\, e \not< e'\}$.

We call $Min(M)$ minimal events of MSC M and $Max(M)$ maximal events of MSC M. $e \in Min(M)$ is called minimal event and $e' \in Max(M)$ is called maximal event.

We will now formally state that, when used correctly, *max* computes the restriction for constraint on duration of MSC. Thus it should compute the difference of time between execution of the first and the last event of an MSC. The intuition for applying the operator is as follows: we find paths through MSC from minimal events to maximal events. For each path we compute the relative constraint on its duration using operator $\oplus$. If we have just two paths we apply the operator on path constraints and we get the final tightening constraint.

Theorem 1. *Let M be an MSC with only two paths from minimal to maximal events. Let $J = \{j_1, \cdots, j_p\}$ be an interval constraint on duration of* M*. Let $T = \{t_1, \ldots, t_m\}$ and $H = \{h_1, \ldots, h_n\}$ be interval constraints on two paths from minimal events to maximal events, $J, T, H \subseteq 2^{\mathcal{I}}$. Then*

$$J \leftarrow J \cap \begin{cases} max(H,T) & \text{if communication graph of M is weakly com-connected} \\ \langle x, \infty) & \text{otherwise,} \end{cases}$$

where x is the lowest lower bound from all intervals in $max(H,T) = (x,y) \cup \ldots$ (if the intervals are ordered according to lower bounds), computes correct tightening on J.

Proof. (sketch) We first prove the case for weakly connected communication graph of M. Imagine we have only simple interval constraints, i.e. $J, H, T \in \mathcal{I}$. Since the communication graph is weakly com-connected we can have just two minimal events and one maximal event or just one maximal event and two minimal events. Thus the paths from minimal to maximal events have to have a common vertex, moreover they share minimal or maximal event. In other words, they are synchronized.

We want to compute interval J' which is correct interval for the time difference between execution of first and last event of MSC. We will show that $J' = max(H,T)$. Let $J = \langle J_l, J_u\rangle, H = \langle H_l, H_u\rangle, T = \langle T_l, T_u\rangle$. The lower bound is equal to $J'_l = max\{H_l, T_l\}$ because both enforce minimal time for duration of their execution. We have to take the longer time in order to get the minimal time of execution of whole MSC. For upper bound we similarly have to take longer from the two upper bounds in order to get the upper bound for the duration of MSC. That is why the resulting upper bound is $J'_u = max\{H_u, T_u\}$. Altogether $J' = \langle J'_l, J'_u\rangle = (max\{H_l, T_l\}, max\{H_u, T_u\}) = max(H,T)$.

If we extend this to multiple intervals per constraint we get

$$max(H,T) = \bigcup_{i=1}^{n} \bigcup_{j=1}^{m} max(H_i, T_j)$$

what is exactly the case for weakly com-connected communication graph of *M*.

For the case of a communication graph of M which is not weakly com-connected we have to have exactly two minimal and two maximal events, i.e. minimal and maximal for each connected component in communication graph which is not trivial. That is why we have two paths from minimal to maximal events which are event disjoint. This follows from the fact that we have only two paths from minimal to maximal events. With simple intervals per constraint the lower bound of J' is the same as for case with weakly connected communication graph with simple intervals. However we cannot assume the same upper bound. The paths do not have any communication between them which would force them to synchronize. The paths can in general start at any time. One can start and finish, while the second one can start long time after the first one finished. That is why the upper bound has to be ∞.

For multiple intervals per constraint we just show that

$$J' = \bigcup_{i=1}^{n} \bigcup_{j=1}^{m} \langle max\{h_{il}, t_{jl}\}, \infty \rangle$$

equals to $\langle x, \infty)$, where x is the lowest lower bound from all intervals in the union and $\langle h_{il}, h_{iu}\rangle \in H \wedge \langle t_{il}, t_{iu}\rangle \in T$. The above union just computes J' according to the approach for simple intervals for every choice of interval from H and interval from T. We get many intervals with some values for lower bound, but ∞ for upper bound. Since all the intervals are included in the interval $\langle x, \infty)$, where x is the lowest lower bound from all intervals in the union. We can simply write $\langle x, \infty)$. □

For more complex MSCs (with more minimal and maximal events) we just need to apply previous approach in the correct way. We have to find all undirected shortest paths from minimal to maximal events and iteratively apply *max* operator to compute just lower bound or all intervals depending whether the MSC is weakly com-connected.

Now we prove that when used correctly *co-max* computes correct tightening on the interval constraint in MSC according to the outside interval constraint on duration of MSC and other interval constraints in MSC.

Theorem 2. *Let* M *be an MSC with only two paths from minimal to maximal events. Let* $J = \{j_1, \cdots, j_p\}$ *be an interval constraint on length of* M*, which was already tightened by* max*. Let* $T = \{t_1, \ldots, t_m\}$ *and* $H = \{h_1, \ldots, h_n\}$ *be constraints on two paths from minimal events to maximal events,* $J, T, H \subseteq 2^{\mathcal{I}}$*. Then*

$$T \leftarrow T \cap co - max(J, H)$$

computes correct tightening on T*.*

Proof. Imagine easier case with only simple intervals. We get three options how the intervals J and H are oriented on the time line (see Figure 6.1). Interval J cannot begin earlier than interval H, because it was already tightened by max.

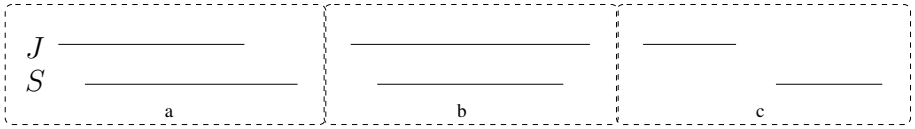

Figure 6.1: Co-max

We would like to tighten upper and lower bounds of intervals appropriately by J. However max is not a symmetric operation. We can bound both upper bounds of intervals H, T by upper bound of J, because none of the two intervals H, T shall contain larger values than interval J. The problem is the tightening of lower bounds. It is sufficient to tighten only one of the intervals H, T. But we do not know which one.

Assume we have situation depicted in Figure 6.1 (c). We see that $H_u < J_l$, where $H = \langle H_l, H_u \rangle$ and $J = \langle J_l, J_u \rangle$. During tightening we can only make intervals more restrictive. We cannot increase H_u and we cannot decrease J_l since we always use intersection when tightening. That is why it is impossible for interval H to be tightened from below, i.e. to meet the restriction that $J_l = H_l$ since $H_l \leq H_u < J_l$. That is why we can tighten the second interval from the both sides, i.e. by an interval J: $T \leftarrow T \cap J$, when $H_u < J_l$. But since $H_l \leq H_u < J_l \leq J_u \implies H \cap J = \emptyset$, which corresponds to the first condition from the definition of $co - max$ for $m = n = 1$. That is why we can tighten by $T \leftarrow T \cap J$.

For case $H \cap J \neq \emptyset$, depicted in Figure 6.1 (a) and (b), we do not know whether we can tighten the lower bound of T. Thus we give least restrictive lower bound: zero. The interval T can be then tightened by $\langle 0, J_u \rangle$ what is exactly as the first case of definition of *co-max* for $m = n = 1$.

For case of the unions of intervals it remains to show that every interval $i \in co{-}max(J, S)$ is correct interval for tightening. We can divide the problem into $p.n$ subproblems, where p is the number of simple intervals in J and n in H. We compute tightening interval for each choice of simple intervals from J and H. Since every simple interval we have got, is correct tightening interval for some choice of intervals we need to include it in whole tightening constraint. This is because we can not restrict too much the original interval constraint. We want to tighten safely. Thus if there is any valid value for constraints we are restricting, the value has to appear in tightening interval. Therefore we union final tightening intervals into the tightening constraint. It is easy to see that this construction computes same thing as $co - max$ by its definition. Altogether we get that operator $co - max$ computes correct tightening. □

6.3 Tightening of MSC

Now we have the operators that we need to tighten the MSCs. First we have to translate the MSC to the setting of TCSP. The TCSP is defined on a digraph with unions of the time interval constraints on the edges of the graph. So that we have to make such graph from

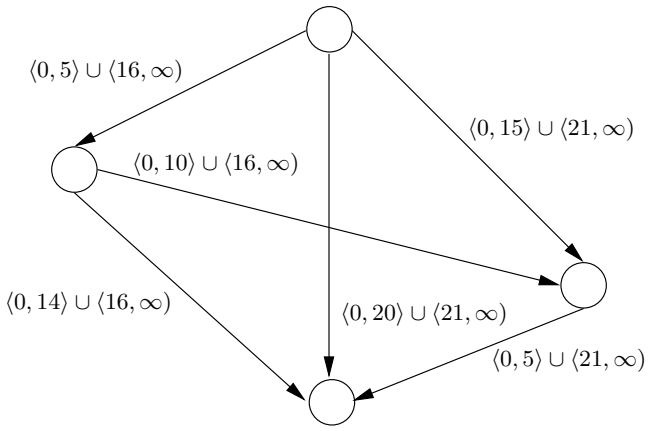

Figure 6.2: Counterargument to PC

MSC. We can view events as the nodes of the graph and we make an edge between the nodes only if the events representing the nodes are ordered via visual order. We also give the constraints on the edges according to a constraint set. New edges are made between start or restart of the timer and corresponding restart, start or timeout. We give constraints on these edges in a following way: If there is a timer t of value $x \in \mathbb{N}$ between start and timeout or restart and timeout (and no other timer t event is between these events) we put on the edge a simple interval $\langle x, x\rangle$. If there is a timer t of value $x \in \mathbb{N}$ between start or restart and restart or stop (and no other timer t event between these events) we put on the edge a simple interval $\langle 0, x\rangle$. If we do not assume coregions and outside constraints on the duration of whole MSC we have correct TCSP, i.e. digraph with unions of the intervals as in [7]. So that we can run any of the known algorithms for solving TCSP. However there are many different algorithms for solving TCSP. Thus we first give an answer, which algorithm is most suitable for our case before making any extensions to solving coregions and outside constraints on the duration of whole MSC.

The problem is that solving general case TCSP is *NP-hard* [7]. For general case TCSP an exponential time complexity algorithm is known. It is called *backtrack algorithm*. However there are some classes of networks for which polynomial algorithms for solving TCSP work correctly. Two of these algorithms are *path consistency* and *directional path consistency* and they work correctly for *series parallel networks*. There are also other algorithms, but they are either similar to the path consistency algorithms or are too complicated for our setting. It is easy to see that our network is not necessarily series parallel. However we still have less difficult problem, because we have always nonnegative values in intervals and we have acyclic graph. Acyclicity of MSCs follows from the partial order. We would like to know whether *path consistency* runs correctly in our type of network. However we were able to find counterargument depicted in Figure 6.2. We provide the path consistency algorithm pseudocode from [7] in 3.

Algorithm 3 Path consistency

```
repeat
  S ⟵ T
  for k := 1 to n do
    for i, j := 1 to n do
      T_ij ⟵ T_ij ∩ (T_ik ⊕ T_kj)
      if T_ij = ∅ then
        {the network is inconsistent}
        exit
      end if
    end for
  end for
until S = T
```

Path consistency algorithm fails to tighten constraint $\langle 0, 20\rangle \cup \langle 21, \infty)$, despite the fact that the value 20 does not have any timing assignment.

Since most of the other algorithms, which we were able to find for general TCSP, just make the backtrack algorithm run faster, we aim our attention to this algorithm. In spite of, there are several improvements to this algorithm the worst case still remains exponential. The main idea of the algorithm is to pick from every constraint only one interval and solving tightening for this restricted problem. Network with only simple intervals is in [7] referred as *simple temporal problem* (STP). The tightening in STP can be done using *Floyd-Warshall algorithm* in $O(n^3)$. So that the algorithm solves simple problem for every choice of the intervals. After it solves the STPs, it unions all tightened STP networks and gain tightened original network. When inconsistent network is reached, the algorithm backtracks as soon as possible. There have been published many papers with improvements to original algorithm which can make it run faster. These approaches use ideas of preprocessing, searching whether the network does belong to a class for which faster techniques are available (especially series parallel networks), improving backtrack search (making trees of the most restrictive intervals, etc.), and improving tightening of STP.

To sum up, we have to be able to solve STP with coregions and outside constraints on the duration of whole MSC. With our operators *max* and *co-max* it is easy to do so. It remains to make final algorithm. The first idea is to iteratively run these steps: solve STP, then constraints on whole MSC and coregions if present. We will iterate until fixed point or inconsistency is reached. We believe that this straightforward approach can be improved significantly, probably using some of the techniques for solving backtrack algorithm faster.

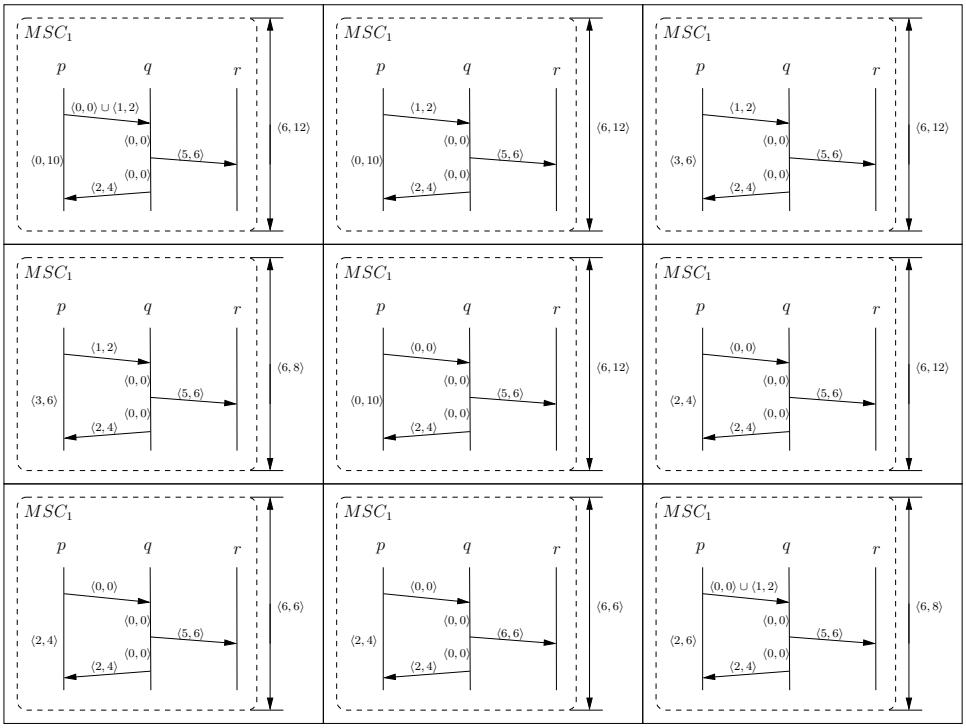

Figure 6.3: Example of tightening of MSC

Chapter 7

Tightening of MSG

When coming to tightening of MSGs we run into many difficulties. One of them is the tightening of the loops. In MSCs many problems are easy, because MSCs represent finitely many behaviors. However this is not valid for MSG. MSG, thanks to the loops, can represent infinitely many scenarios. That is why we would have great difficulties when using previous algorithms. They do not even have to terminate. But it is still possible to do some work regarding to tightening. We will suggest the approach for the tightening of MSG.

7.1 Main Approach

The main idea is that whole MSG , when we look at it as graph, has different semantics as single MSC. Imagine we have multiple paths between two points in MSC and MSG. In MSC the paths are evolving simultaneously and that is why the time constrains on these paths have to conform. Thus we can tighten the constraints according to these parallel paths. However in MSG the paths represent alternative executions. Literally if we get to the same node via different paths, we are in a totally different situation. To the concrete path we can apply only timing constraints which are present along the path. From this reasoning follows our aim to use the path consistency algorithms.

We can view the parallel paths in MSC as concurrent execution. That is why we use intersection operator. On the other hand parallel paths are alternative paths in MSG. Thus we use union operator. In Figure 7.1 we give an example of parallel paths in MSG.

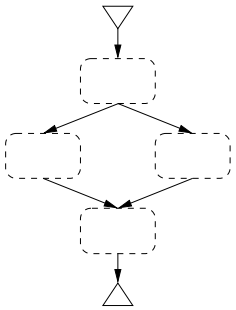

Figure 7.1: Example of parallel paths in MSG

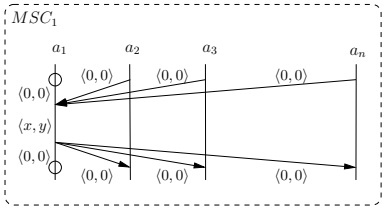

Figure 7.2: Synchronization

Our suggestion, to deal with tightening, is to concatenate MSCs represented by neighboring nodes between which there is no boundary for any MSG constraint. For constraints that are between successive nodes in MSG from the end of one node to the beginning of the second node we make an MSC depicted in Figure 7.2, i.e. we synchronize the all processes by messages to one process, we enforce the time constraint from MSG and then we distribute the delay to all processes. This construction follows directly from Z.120 standard. The constraint restricts the duration of time between execution of the last event in first MSC to execution of the first event in following MSC. So these constraints show us pairs of nodes in MSG (i.e. MSCs) which cannot interleave. Thus we are getting a graph with interval constraints and timers where the exact nodes are those which we have just made i.e. one exact node for the beginning and one for the end of each synchronization MSC made. In this graph we can run (altered directional) path consistency tightening. After we have tightened MSG we then tighten each constraint from beginning of the node to the end of the other node. We first concatenate these two nodes and all intermediate nodes according to the transition function to form an MSC. Then we tighten the constraint according to this MSC using operator max, we can tighten the MSC by the outside constraint using operator $co-max$ and we can tighten the MSC with its inner constraints. We do this for all such constraints from the beginning to the end of the node in MSG. Then we can apply the tightening of MSG again and the inside tightening of MSCs until we reach the fixed point or gain inconsistency.

The advantage of the path consistency algorithms from [7] is that they run in the polynomial time. However we cannot use the algorithms straight away, because of many problems. We first have to adjust them to MSG setting, i.e. due to alternative paths we can tighten outside constraint only if all the inside paths allow it to be tightened. This follows from the union operator. There are also problems with different semantics of cycles, tightening of branching nodes and interleaving of MSCs. So far we cannot solve all the problems, but we have already made first steps. One of them is an observation that we cannot tighten internal interval constraints of loop by the outside interval constraint.

7.2 Loops

Because the loop represents infinitely many runs we have to be able to tighten all of the runs, if we want to tighten the loop. Imagine a loop depicted in Figure 7.3.

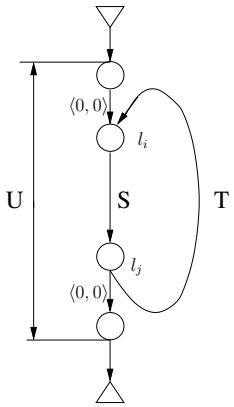

Figure 7.3: Loop

Theorem 3. *Given an MSG $G = (S, \rightarrow, s_0, s_f, L, \mathcal{L}, \mathcal{K}_{MSG}, \mathcal{T}_{MSG})$ and a loop $l = l_0.l_1 \dots l_n.l_0$, where $n \in \mathbb{N}$. We cannot tighten the interval constraints in the loop by an outside interval constraint for duration of a loop.*

Proof. Let l_i be a vertex which is the input vertex for a cycle from the initial the vertex, i.e. there is a path $s_0.k_0 \dots k_p.l_i$ where $p \in \mathbb{N}, \forall r, s \ . \ r \in \{0, \dots, p\}, s \in \{0, \dots, n\}, k_r \neq l_s$. And let l_j be vertex which is the output vertex from the loop to the terminal vertex, i.e. there is a path $l_j.k_0 \dots k_p.s_f$ where $p \in \mathbb{N}, \forall r, s \ . \ r \in \{0, \dots, p\}, s \in \{0, \dots, n\}, k_r \neq l_s$. Let S, T be the constraint for execution of shortest path along cycle from l_i to l_j and l_j to l_i respectively and U an interval constraint for an execution of the cycle, see Figure 7.3. We prove that we cannot tighten the constraints S, T in the loop by constraint U.

We first prove the case for simple intervals. Let $S = \langle c, d \rangle$, $T = \langle a, b \rangle$ and $U = \langle x, \infty \rangle$. The last constraint has to be infinity. If it was not be infinity, the MSG G would be trivially inconsistent.

For $b, d = 0$ we cannot tighten the loop trivially, since $0 \leq a \leq b$ and $0 \leq c \leq d$ implies $a, b, c, d = 0$. Since we can only make the intervals to be smaller, only consistency check is sufficient.

For $d \neq 0 \vee b \neq 0$. Assume that $b \neq 0$ (for $c \neq 0$ the proof is similar). We choose run which is made by $y \in \mathbb{N}$ iterations of loop such that y satisfies the inequality

$$x - yb < 0 \Longrightarrow y > \frac{x}{b}.$$

If we want to tighten any interval $\langle k_z, r_z \rangle$ where $z \in \{1, 2, \dots, 2y + 3\}$ we get

$$\langle k_z, r_z \rangle \longleftarrow \langle k_z, r_z \rangle \cap \langle \quad x - r_1 - r_2 - \dots - r_{z-1} - r_{z+1} - \dots - r_{2y-3},$$
$$\infty - k_1 - k_2 - \dots - k_{z-1} - k_{z+1} - \dots - k_{2y-3} \rangle.$$

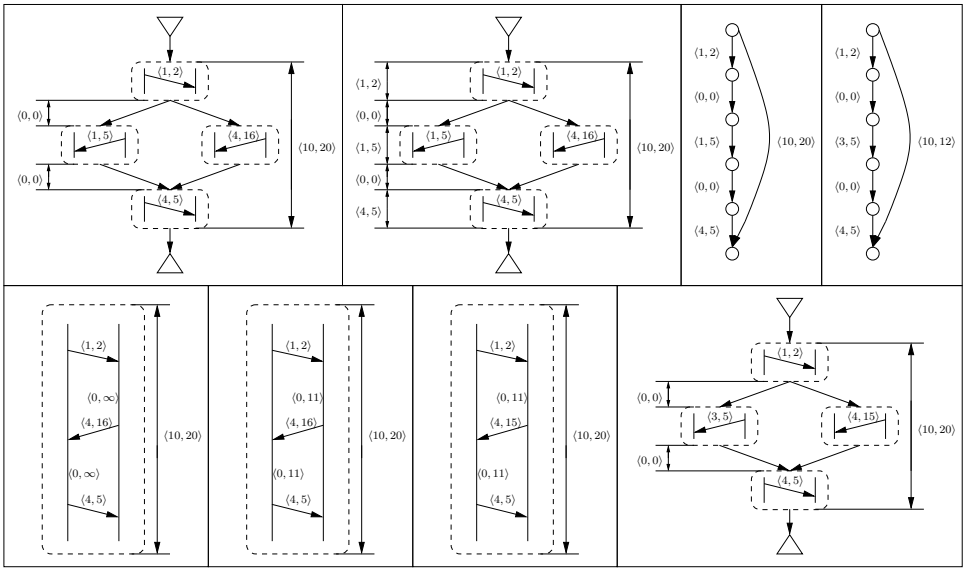

Figure 7.4: Example of tightening of an MSG

Let

$$k'_z = x - r_1 - r_2 - \cdots - r_{z-1} - r_{z+1} - \cdots - r_{2y-3} = x - d - b - d - b - d - \cdots - d.$$

Since we have $y + 1$ or y values of b

$$k'_z \leq x - b.(y + 1) \leq x - by < 0$$

and we get

$$\langle k_z, s_z \rangle \cap \langle k'_z, \infty) = \langle k_z, s_z \rangle,$$

because $k'_z < 0$. We have chosen arbitrary interval in a run, that is why we cannot tighten any interval in run. Since we cannot tighten one run, we cannot tighten loop.

For unions of the intervals we just do the above for every choice of simple intervals S, T and the interval $\langle x, \infty) \in U$. From the above argument it follows that we cannot tighten any choice of the simple intervals from S, T. When we union all the results from tightening, all values for S, T will appear in new S, T. Thus we do not have to do any work regarding tightening for every choice of simple intervals from S, T and $U \setminus \langle x, \infty)$. From that follows that we cannot tighten constraints S, T by U. □

Chapter 8

Conclusion

In this thesis we have addressed the formal background of the time extension of MSC. We have defined MSC and MSG with timing constraints and problems regarding time and MSC, such as timing consistency and finding minimal networks. We have studied published results to this problematics without finding a complete solution. Lposet approach has been chosen as the closest approach to our setting. Then the time constraints in MSC to proper time constraints, i.e. constraint specifications cleaned of erroneous and ambiguous constraints, have been restricted. We have also introduced algorithms and algorithm outlines for checking whether all constraints in MSG are proper. Then we extended the lposet approach to be strong enough to compute solutions to all previously specified problems. We have also provided outlines of algorithms for achieving this. It remains to finish the algorithms and implement them into our developed tool *SCStudio*.

Bibliography

[1] S. Akshay, M. Mukund, and K.N. Kumar. Checking Coverage for Infinite Collections of Timed Scenarios. *Lecture Notes in Computer Science*, 4703:181, 2007.

[2] R. Alur, K. Etessami, and M. Yannakakis. Realizability and verification of MSC graphs. *Theoretical Computer Science*, 331(1):97–114, 2005.

[3] R. Alur, G.J. Holzmann, and D. Peled. An Analyzer for Message Sequence Charts. In *TACAS'96*, LNCS, pages 35–48. Springer, 1996.

[4] R. Alur and M. Yannakakis. Model checking of message sequence charts. *Lecture Notes in Computer Science*, pages 114–129, 1999.

[5] H. Ben-Abdallah and S. Leue. Expressing and analyzing timing constraints in message sequence chart specifications. *Department of ElectricalComputer Engineering, University of Waterloo*, 1997.

[6] P. Chandrasekaran and M. Mukund. Adding Time to Scenarios. *Next Generation Design and Verification Methodologies for Distributed Embedded Control Systems.*, page 83, 2007.

[7] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. In *Knowledge Representation*, volume 49, pages 61–95. MIT Press, 1992.

[8] E. Elkind, B. Genest, and D. Peled. Detecting Races in Ensembles of Message Sequence Charts. In *TACAS'07*, volume 4424 of *LNCS*, pages 420–434. Springer, 2007.

[9] ITU Telecommunication Standardization Sector Study group 17. ITU recommandation Z.120, Message Sequence Charts (MSC), 2004.

[10] J.G. Henriksen, M. Mukund, K.N. Kumar, and PS Thiagarajan. *Towards a theory of regular MSC languages*. BRICS, Computer Science Department, University of Aarhus, 1999.

[11] T.H. Kim and S.D. Cha. Timed High-Level Message Sequence Charts for Real-Time System Design. *Lecture Notes in Computer Science*, 4320:82, 2006.

[12] K.G. Larsen, P. Pettersson, and W. Yi. Uppaal in a Nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):134–152, 1997.

[13] X. Li and J. Lilius. *Timing analysis of message sequence charts*. Turku Centre for Computer Science, 1999.

[14] Xuandong Li, Bu Lei, Jun Hu, Jianhua Zhao, Tao Zhang, and Guoliang Zheng. Scenario-based timing consistency checking for time petri nets. In Elie Najm, Jean-François Pradat-Peyre, and Véronique Donzeau-Gouge, editors, *FORTE*, volume 4229 of *Lecture Notes in Computer Science*, pages 388–403. Springer, 2006.

[15] P. Lucas. Timed semantics of message sequence charts based on timed automata. *Electronic Notes in Theoretical Computer Science*, 65(6):160–179, 2002.

[16] MSCan – Message Sequence Charts analyzer. `http://aprove.informatik.rwth-aachen.de/~kern`. [Online; accessed 18-April-2009].

[17] A. Muscholl and D. Peled. Message Sequence Graphs and Decision Problems on Mazurkiewicz Traces. In *MFCS'99*, volume 1672 of *LNCS*, pages 81–91. Springer, 1999.

[18] SCStudio – Sequence Chart Studio. `http://scstudio.sourceforge.net/`, 2009. [Online; accessed 18-April-2009].

[19] P. Slovák. Decidable Race Conditions in High-Level Message Sequence Charts. *Bachelor thesis*, Faculty of Informatics, Masaryk University, Brno, 2008.

[20] UBET (formaly named MSC/POGA.). `http://cm.bell-labs.com/cm/cs/what/ubet/`. [Online; accessed 18-April-2009].

[21] T. Zheng and F. Khendek. Time consistency of MSC-2000 specifications. *Computer Networks*, 42(3):303–322, 2003.