Deciding Non–local Choice in High–level Message Sequence Charts

Bachelor Thesis

Martin Chmelík

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 24, 2009
Martin Chmelík

Advisor: RNDr. Vojtěch Řehák, Ph.D.
Acknowledgement

I would like to thank my advisor RNDr. Vojtěch Řehák, Ph.D. for help and guidance throughout the work on this thesis.
Abstract

Non-local choice property in Message Sequence Chart formalism is causing unwanted and unspecified behavior in the implemented system.

This work focuses on detecting non-local choice in High-level Message Sequence Chart specifications. We propose an algorithm that detects non-local choice. We introduce a new approach to deal with non-local choice, by having further assumptions on the system.

The approach cannot be applied in general, but we can algorithmically decide, whether our approach is applicable. Moreover, when we cannot apply our approach, we may advise the designer, what is causing the inapplicableness.
Keywords

MSC, message sequence chart, MSG, message sequence graph, HMSC, high-level message sequence chart, hierarchical message sequence chart, NLC, non-local choice.
# Contents

1. **Foundations** ........................................... 5  
   1.1 Basic Message Sequence Charts .......................... 5  
   1.2 High-level Message Sequence Charts ......................... 6  
2. **Non-local Choice** ........................................ 9  
   2.1 Algorithm to detect Non-local choice nodes in HMSC ........ 11  
      2.1.1 Total correctness .................................... 11  
3. **Decidable Non-local Choice** ................................. 15  
   3.1 Checking paths for deciding events ........................ 18  
   3.2 Checking boundedness for deciding events ..................... 19  
      3.2.1 Characterizing MSGs containing unbounded deciding events 20  
      3.2.2 Recognizing self-deciding cycles ...................... 22  
4. **Algorithms** ................................................ 25  
   4.1 Listing elementary cycles ................................. 25  
      4.1.1 Generating all acyclic paths ....................... 25  
   4.2 MSC data structure ..................................... 27  
      4.2.1 Concatenating MSCs .................................. 27  
      4.2.2 Finding a deciding event in an MSC .................. 28  
   4.3 Algorithm for decidable non-local choice .................... 29  
      4.3.1 Highlighting deciding events ....................... 29  
      4.3.2 Total correctness ................................... 29  
      4.3.3 Complexity .......................................... 29  
5. **Conclusion** ............................................. 31
Introduction

The Message Sequence Chart formalism [6] is a widely used mechanism to design message passing systems, such as network protocols and inter-process communication. Because the formalism provides visual and textual representation, the design may be done in a very intuitive way without losing the possibility to automatically verify and test the design.

Verification of a complex system in the early stage of development may help to discover potential problems and is therefore of great interest to designers.

Many properties that can lead to difficulties have been studied, such as race conditions [1][4][5], boundedness [2], non-local choice [9], deadlock, livelock, cycles. Some of these properties were proved to be undecidable, but there are also positive results e.g. the trace race problem [13].

There are tools developed to perform verification on the designs, such as uBet [14], MSCan [10] and SCStudio [12] that is being developed at Faculty of Informatics, Masaryk University.

This work focuses on the non-local choice problem, that appears in a Message Sequence Chart specification, when multiple behaviors are possible, but the decision, which behavior will be executed, has to be performed by distributed processes. One possible solution is to perform an explicit synchronization. Otherwise, this behavior cannot be implemented without unwanted and unspecified behavior of the system.

In this work, we present a new approach dealing with non-local choices that is applicable on certain Message Sequence Chart specifications and is having reasonable demands on the underlying system.
Chapter 1

Foundations

The Message Sequence Charts formalism is a textual and graphical language for describing interaction among parallel components of a distributed system. The formalism was introduced in International Telecommunications Union (ITU–T) Recommendation Z.120 [6]. In this chapter we introduce and define needed terms.

1.1 Basic Message Sequence Charts

Every basic Message Sequence Chart (MSC) describes a finite execution of a message–passing system. In a visual MSC representation each process of the system is represented by a vertical line. Message exchange is denoted as an arrow from the source process to the destination process. Because we assume asynchronous communication, we need to partition the message exchange into two events – send and receive events. The ordering of the messages from the top of the diagram determines the desired sequence of messages.

There are many various definitions of basic Message Sequence Chart (MSC). We will use a definition from [5].

Definition 1. An MSC is defined as a tuple \((E, <, P, T, P, M)\) where

- \(E\) is a finite set of events;
- \(<\) is a partial ordering on \(E\) called visual order;
- \(P\) is a finite set of processes;
- \(T : E \rightarrow \{\text{send}, \text{receive}\}\) is a labeling function dividing events into two types — send and receive;
- \(P : E \rightarrow P\) is a mapping that associates each event with a process;
- \(M \subseteq (T^{-1}(\text{send}) \times T^{-1}(\text{receive}))\) is a bijective mapping, relating every send with a unique receive, such that for any \((e, f) \in M\) we have
1. Foundations

Figure 1.1: MSC concatenation

\[ P(e) \neq P(f) - \text{a process cannot send messages to itself;} \]

Visual order \(<\) is defined as the reflexive and transitive closure of

\[ \mathcal{M} \cup \bigcup_{p \in P} <_p \]

where \(<_p\) is a total order on \(P^{-1}(p)\).

1.2 High-level Message Sequence Charts

To design complex systems the expressivity of MSCs may not be sufficient. When designing a complex system, we may need to specify alternations and potentially infinite behavior. That is the reason for introducing High-level Message Sequence Charts (HMSC), which allow us to combine MSCs. An HMSC can be imagined as a directed graph with two types of nodes — MSCs and supernodes containing HMSCs. The recursion may be only finite, so it is always possible to unfold supernodes and obtain a semantically equivalent HMSC with nodes labeled only by MSCs as [2] shows. We will refer to MSC labeled HMSCs as Message Sequence Graphs (MSG).

Every MSG has to contain two special nodes — an initial node and a terminal node. Every path from the initial node to the terminal node uniquely identifies an MSC by sequential composition of MSCs on the path. To define everything more formally, we will use a definition from [2].

Definition 2. An MSG is defined as a tuple \(G = (S, \tau, s_0, s_f, L)\) where

- \(S\) is a finite set of states.
- \(\tau \subseteq S \times S\) is an edge relation.
1. FOUNDATIONS

- $s_0 \in S$ is an initial state.
- $s_f \in S$ is a terminal state.
- $L : S \rightarrow MSC$ is a labeling function;

A finite sequence of nodes starting with an initial node and ending with a terminal node that has been constructed according to a path in the graph is called a run.

**Definition 3.** Given an MSG $G = (S, \tau, s_0, s_f, L)$ a finite sequence of states $s_1, s_2, \ldots, s_k$, where $\forall 1 \leq i < k : (s_i, s_{i+1}) \in \tau$, is a path. A path, where $s_1 = s_0$ and $s_k = s_f$, is a run.

We need to show how an MSG identifies a set of possible MSCs. It will be done by assigning an MSC to every run. This is achieved by sequential composition of MSCs on the path. It is sufficient to show how to compose two MSCs.

**Definition 4.** Given two MSCs $M_1 = (E_1, <_1, P, T_1, P_1, M_1)$ and $M_2 = (E_2, <_2, P, T_2, P_2, M_2)$, let their sequential composition be an MSC $M_1 \cdot M_2 = ((E_1) \cup (E_2), <, P, T_1 \cup T_2, P_1 \cup P_2, M_1 \cup M_2)$, where $<$ is a transitive closure of $<_1 \cup _2 \cup \bigcup_{p \in P}(P_1^{-1}(p) \times P_2^{-1}(p))$.

Because we do not synchronize processes when composing, it is possible that some events from the latter MSC may be executed before the events from the first MSC. See Figure 1.1 sending event of message $e_2$ may be executed even before the receive event of message $v_1$.

**Definition 5.** Let $G = (S, \tau, s_0, s_f, L)$ be an MSG and $\sigma = s_1, s_2, \ldots, s_k$ be a path in $G$. Then MSC $M_\sigma = L(s_1) \cdot L(s_2) \ldots \cdot L(s_k)$.

The set of all MSCs specified by an MSG is defined as follows:

**Definition 6.** Given an MSG $G = (S, \tau, s_0, s_f, L)$, let $L(G)$ be a set of MSCs:

$L(G) = \{ M_\sigma | \sigma$ is a run in $G \}$. 


Chapter 2

Non–local Choice

Sometimes we may consider an MSG specification to be more than a sequence diagram. We may try to find patterns of constructions that are hard to implement, or can lead to potential errors when implementing.

Detecting such problematic scenarios in an early stage of development prevents the final product of containing errors and ensures smooth implementing of desired behavior.

Non–local choice is a problematic property, occurring when processes from set $\mathcal{P}$ do not share a common memory. Whenever there is a branching node in an MSG specification, we want exactly one branch to continue with executing the run. That may be hard to achieve when there are two different branches initiated by different processes from $\mathcal{P}$. These processes should somehow synchronize and agree on a branch that will be executed, but that is not possible without any extra communication, because they do not share any memory. The branching node is called a non–local choice node.

Running an MSG with a non–local choice leads to an unwanted and unspecified behavior. We call this behavior an implied behavior. It appears as a consequence of a non–local choice node, when all the processes in different branches start executing their branches. An implied behavior for a non–local choice node can be seen in Figure 2.1.

There have already been approaches how to detect and deal with non–local choice nodes. Unfortunately there have been also many different definitions of non–local choice.

There is an algorithm in [3] detecting non–local choice nodes in MSG specification that runs in time linear to the total number of messages exchanged in the MSG specification. The speed of algorithm is achieved by further assumption on every MSC in MSG. However the result looses generality, because it is assumed that every process in every MSC exchanges at least one message.

An algorithm to detect non–local choice nodes in a general MSG is presented in [11]. The authors aim is to detect implied scenarios, by constructing LTS and analyzing them. Unfortunately the authors were unable to
prove the correctness of the algorithm.

A way how to deal with detected non–local choice nodes is introduced in [8]. A fixed arbiter process from $\mathcal{P}$ is chosen. Unspecified implied behavior is allowed in every branch, but it is guaranteed that there exists a position in every branch, where arbiter process decides which branch was chosen and the rest of the communication from other branches is forgotten. This approach is not general — there are MSGs, where no such arbiter can be chosen. No algorithm detecting MSGs, where this approach can be applied, is presented.

There are more definitions of non–local choice, but we consider the one most describing what the problem is in [9].

To formally define non–local choice we need to introduce some new notions. We will denote by $\text{Min}_B(M)$ a subset of $\mathcal{P}$ that contains all processes that are possessing a minimal event in an $\text{MSC} M$. 

2. NON–LOCAL CHOICE

![Diagram](image1.png)

An implied behaviour:

![Diagram](image2.png)

Figure 2.1: Non–local choice node with an implied behavior

Figure 2.2: Non–local choice
Definition 7. Let $M = (E, <, P, T, P, M)$ be an MSC. Then let

$$\text{Min}_B(M) = \{ p \in P \mid (\exists e \in E), (\forall e' \in E) : \neg(e' \leq e) \land P(e) = p \}$$

The previous notion can be extended on MSGs as follows:

Definition 8. Let $G = (S, \tau, s_0, s_f, L)$ be an MSG. Then let

$$\text{Min}_H(G) = \{ p \in P \mid (\exists \text{MSC} H) (H \in \text{Lin}(G) \land p \in \text{Min}_B(H)) \}$$

Given an MSG $G = (S, \tau, s_0, s_f, L)$ and $u \in S$, let the successor set be $\text{Succ}(u) = \bigcup_{(u, v) \in \tau} \{ v \}$.

Finally we are ready to formally define non-local choice.

Definition 9. Given an MSG $G = (S, \tau, s_0, s_f, L)$ a state $q \in S$ is called a non-local choice node iff $|\text{Succ}(q)| \geq 2$ and one of the following holds:

- $\exists v_k, v_l \in \text{Succ}(q)$, such that $\text{Min}_H(G_k) \neq \text{Min}_H(G_l)$
- $\exists v_k \in \text{Succ}(q)$, such that $|\text{Min}_H(G_k)| \geq 2$.

where:

$$\text{MSG } G_k = (S, \tau, v_k, s_f, L)$$
$$\text{MSG } G_l = (S, \tau, v_l, s_f, L)$$

2.1 Algorithm to detect Non-local choice nodes in HMSC

As no suitable algorithm with correctness proof on general MSGs is known, we present our algorithm that detects non-local choice nodes.

The idea behind the algorithm is to compute for every node in MSG a subset of processes that initiate the behavior directly after this node. As we want this algorithm to work on general MSGs, we do not assume that every process in every MSC exchanges a message. Therefore looking only at the successor set may not be enough. Whenever there is an idle process (does not exchange a message) in an MSC, it is possible that the initiative in the communication is propagated from some MSC deeper in the graph.

This leads to our iterative Algorithm 1 that improves this set until it reaches a fixed-point. When we have this set for every node it is easy to detect non-local choice nodes.

2.1.1 Total correctness

In the following section will be shown, that the algorithm is totally correct. We need to show that the algorithm terminates and is partially correct.
Algorithm 1 Detect Non–local Choice nodes

1: {Initialization}
2: for all \( q \in S \) do
3: \( q_0.\text{MEP} \leftarrow \text{Min}_B(c(q)) \)
4: \( q.\text{IP} \leftarrow \text{Idle processes in } c(q) \)
5: \( i \leftarrow 0 \)
6: {Iteration}
7: repeat
8: \( i \leftarrow i + 1 \)
9: for all \( q \in S \) do
10: \( q_i.\text{MEP} \leftarrow (\bigcup_{v \in \text{Succ}(q)} v_{i-1}.\text{MEP}) \cap q.\text{IP} \cup q_{i-1}.\text{MEP} \)
11: until \( \exists q \in S : (q_i.\text{MEP} \neq q_{i-1}.\text{MEP}) \)
12: {Detect NLC nodes}
13: for all \( q \in S \) do
14: \( q.\text{NLC} \leftarrow \text{FALSE} \)
15: if \(|\text{Succ}(q)| \geq 2\) then
16: if \((\exists v \in \text{Succ}(q) : |v_i.\text{MEP}| \geq 2)\) then
17: \( q.\text{NLC} \leftarrow \text{TRUE} \)
18: else if \((\exists v, w \in \text{Succ}(q) : v_i.\text{MEP} \cap w_i.\text{MEP} = \emptyset)\) then
19: \( q.\text{NLC} \leftarrow \text{TRUE} \)

Termination The only part of the code that can prevent the algorithm from terminating is the iteration phase (lines 7–11). We assign to each iteration its state, which is defined as the value of variable \( q.\text{MEP} \) for all \( q \in S \) at line 11. More formally a state of the \( i \)-th iteration is defined:

\[
Q_i = \prod_{q \in S} q_i.\text{MEP}
\]

There is only a finite number of different states. So it remains to show, that the algorithm does not cycle on these states.

Lemma 1. Let \( 0 \leq i \leq j \), then the following holds:

\[
\forall q \in S : q_i.\text{MEP} \subseteq q_j.\text{MEP}
\]

Proof. Follows directly from the variable computation. \( \square \)

Suppose there is a sequence of indexes \( u \leq v \leq w \) during the run, such that \( Q_u \neq Q_v \) and \( Q_u = Q_w \). Then there exists a \( q \in S \), such that:

\[
(q_u.\text{MEP} \subset q_v.\text{MEP})
\]
It follows from Lemma 1 that during the computation, we cannot reach state $Q_w$. So we reached a contradiction with the assumption. Therefore the algorithm will terminate, when some state is repeated for the first time.

**Partial correctness** Suppose that $MSG G = (S, \tau, s_0, s_f, L)$ is the input of the algorithm. During the computation the following invariant holds:

$$\forall q \in S, \forall p \in P, \forall i \geq 0 : p \in q_i.MEP \Rightarrow p \in \text{MIN}_H(G_q).$$

Moreover, when $Q_i = Q_{i+1}$, then

$$\forall q \in S, \forall p \in P : p \in q_i.MEP \Leftrightarrow p \in \text{MIN}_H(G_q),$$

where $MSG G_q = (S, \tau, q, s_f, L)$.

**Proof.** We will prove the first statement by induction with respect to the number of iterations. Let $p \in P$ and $q \in S$ be arbitrary, but fixed.

**Base case** $p \in q_0.MEP \Rightarrow p \in \text{Min}_B(L(q)) \subseteq \text{Min}_H(G_q)$

**Induction step** $p \in q_{i+1}.MEP$

There are two possibilities to consider:

$p \in q_i.MEP \Rightarrow p \in \text{Min}_H(G_q)$ (induction hypothesis)

$p \notin q_i.MEP \Rightarrow p \in \bigcup_{v \in \text{Succ}(q)} (v_i.MEP \cap q.IP)$

$\Rightarrow \exists v \in \text{Succ}(q) : p \in v_i.MEP \cap q.IP$

$\Rightarrow$ (induction hypothesis) $p \in \text{Min}_H(G_v) \cap p \in q.IP$

$\Rightarrow p \in \text{Min}_H(G_q)$

where $MSG G_v = (S, \tau, v, s_f, L), MSG G_q = (S, \tau, q, s_f, L)$.

Now we will prove the second statement. The $\Rightarrow$ case follows from the first statement. The $\Leftarrow$ case will be proved by contradiction.

We will fix the $i$-th iteration and denote by $q_i.MEP$ the value of $q_i.MEP$. Assume $p \in \text{Min}_H(G_q) \land p \notin q_i.MEP$. Because $p \in \text{Min}_H(G_q)$ there exists a path $s_0, \ldots, s_n$, such that $s_0 = q, p \in s_n.MEP$ and for $0 \leq j \leq (n - 1) : p \notin s_j.MEP$ and $p \in s_j.IP$. Therefore the algorithm would add process $p$ to the set $s_{(n-1)}.MEP$ in the next iteration. But that is not possible, because $Q_i = Q_{i+1}$. So there is no such node and the length of the path is one. But that is a contradiction with the assumption, because $p \in s_n.MEP = s_1.MEP = q.MEP$. □
Chapter 3

Decidable Non–local Choice

In this chapter we will present a new approach to deal with non–local choice nodes. We will assume that we are able to add some additional bounded information to every message. And that each process is able to store some bounded information. The non–local choice will be resolved by adding additional data into preceding communication. These data will be distributed among all the processes, that are involved in initiating the communication directly after the non–local choice node.

This work describes resolving non–local choice only for one arbitrary but fixed non–local choice node $q$. To resolve the whole MSG specification the procedure has to be performed for every non–local choice node.

We can assign to the non–local choice node $q$ in $MSG G = (S, \tau, s_0, s_f, L)$ a set of processes, to which we need to distribute the information. We will call this set $Trigger$.

$$Trigger(q) = \bigcup_{v \in Succ(q)} Min_H(G_v)$$

where $MSG G_v = (S, \tau, v, s_f, L)$.

We will say, that a send event $e$ in $MSC L(w)$ can resolve a non–local choice node $q$ in $MSG G$, if there exists a run going through $w$ and $q$, such that the information added to message $(e, f) \in M$, can be distributed during the run to all processes from $Trigger(q)$, by adding the information to existing communication.

**Definition 10.** Let $G = (S, \tau, s_0, s_f, L)$ be an MSG, $q$ a non–local choice node, $\sigma = s_0, \ldots, w, \ldots, q, \ldots, s_f$ a run in $G$, an event $e$ from $MSC L(w)$ is a resolving event for $q$ on $\sigma$, iff $T(e) = send$ and $MSC_{w\ldots q} = (E, <, P, T, P, M)$ has the following property:

$$\forall p \in Trigger(q), \exists e_p \in P^{-1}(p) : e < e_p$$
The foregoing definition covers events that are able to distribute additional information among processes in set \( \text{Trigger}(q) \). We would like to detect events that resolve non–local choice as late as possible. We will name these events deciding events.

**Definition 11.** A resolving event \( e \) for \( q \) on \( \sigma \) is a deciding event iff for all other resolving events \( e' \) for \( q \) on \( \sigma \) condition \(-\neg(e < e')\) holds.

**Proposition 1.** Let \( \sigma \) be a path without a deciding event for \( \text{Trigger}(q) \) and \( \sigma' \) a subsequence of \( \sigma \). Then \( \sigma' \) does not possess a deciding event for \( \text{Trigger}(q) \).

Let \( \sigma' \) be a path with a deciding event for \( \text{Trigger}(q) \) and let \( \sigma' \) be a subsequence of \( \sigma \). Then \( \sigma \) possesses a deciding event for \( \text{Trigger}(q) \).

Suppose we will find deciding events on every possible path from the initial node. That would basically mean that it doesn’t matter how we reached a non–local choice node, because we would always be able to add additional information, which solves the non–local choice problem. Clearly not all possible MSGs are having the needed property. See Figure 2.1.

But even when all paths from the initial node to a non–local choice node are possessing a deciding event, we may get into trouble when implementing such behavior.

Consider Figure 3.3. Every path incoming to a non–local choice node \( q \) contains node \( p \) with a deciding event \( a \) for \( q \). The amount of information that needs to be added to message \( a \) may rise above any given bound,
because a single occurrence of the deciding message \( a \) needs to decide potentially infinite occurrences of the non–local choice node \( q \).

![Figure 3.3: Problematic non–local choice](image)

The previous requirements lead to the following definition.

**Definition 12.** A non–local choice node \( q \) is called decidable iff every path from initial node to node \( q \) possesses a deciding event and there exists a constant \( k \) for node \( q \), such that every occurrence of a deciding event has to decide less or equal than \( k \) occurrences of the non–local choice node \( q \).

We will refer to property that additional information is bounded as to boundedness of the deciding event.

We will show how to check whether a given MSG \( G \) with a non–local choice node \( q \) is decidable. We will have to check two properties:
1. Every path from initial node to node $q$ possesses a deciding event.

2. Every occurrence of a deciding event has to decide less or equal than $k$ occurrences of the non–local choice node.

### 3.1 Checking paths for deciding events

**Definition 13.** Let $q$ be a non–local choice node in MSG $G$, then let

$$Path(G, q) = \{ \sigma \mid \sigma \text{ is a path from initial node to } q\}$$

We need to check that every MSC $M_{\sigma}$ for every $\sigma \in Path(G, q)$ contains a deciding message for $q$. But all possible elements of $Path(G, q)$ cannot be generated, because this set may be potentially infinite.

**Definition 14.** A path $\sigma$ in the directed graph $G = (V, E)$ is called acyclic, if each vertex $v \in V$ has at most one occurrence in $\sigma$.

The following lemma shows that it suffices to check only a finite subset of all possible paths.

**Lemma 2.** There exists an MSC $M_{\sigma}$ for $\sigma \in Path(G, q)$ without a deciding event iff there exists an MSC $M_{\sigma'}$ without a deciding event for acyclic $\sigma' \in Path(G, q)$.

**Proof.**

$\Leftarrow$ Obvious.

$\Rightarrow$ We will show converse. Let all MSC $M_{\sigma'}$, with acyclic $\sigma' \in Path(G, q)$ contain a deciding event, then all MSC $M_{\sigma}$ with $\sigma \in Path(G, q)$ are possessing a deciding event.

Consider a directed graph $G = (V, E)$, where:

- $V = \{ v \mid v \text{ is a state from } \sigma \}$
- $E = \{ (u, v) \mid u, v \text{ are nodes from } \sigma \text{ and } uv \text{ is a substring of } \sigma \}$

As $\sigma$ is a path from the initial node to the non–local choice node $q$, there must exist a shortest path $\sigma'$ in $G$ from the initial node to $q$. This path is acyclic, therefore it possesses a deciding event. The path $\sigma'$ is a subsequence of path $\sigma$ and by Proposition 1 path $\sigma$ possesses a deciding event.

$\square$
3. Decidable Non–local Choice

To check the needed property it suffices to check all paths from the following set.

\[ \text{Acyclic–Path}(G, q) = \{ \sigma' \mid \sigma' \text{ is an acyclic path from initial node to } q \} \]

The upper bound for the set cardinality will be given in Chapter 4.

3.2 Checking boundedness for deciding events

The requirement to keep the data bounded is quite natural. Otherwise we would force the execution of the MSG to loop only for a limited number of iterations, or to leave the non–local choice after bounded decided iterations unsolved.

Whenever an MSG contains a loop with a non–local choice node on it, the execution may repeat this loop potentially infinitely many times. So the loop itself has to contain a deciding event for the non–local choice node. But as we will see later not necessarily in the first iteration.

All cycles containing a fixed non–local choice node \( q \) have to be checked. Therefore we may restrict the whole MSG only to the strongly connected component containing \( q \).

**Definition 15.** A directed \( G = (V, E) \) is called strongly connected iff for arbitrary two vertices \( u, v \in V \), there exists a path from \( u \) to \( v \) and from \( v \) to \( u \). Strongly connected component containing vertex \( q \) is a maximal strongly connected subgraph containing \( q \).

As the set of all cycles from some non–trivial strongly connected component is always infinite, we will use a variant of Lemma 2 and will check only distinct elementary cycles.

**Definition 16.** A cycle is elementary if no vertex appears twice. We will call two cycles distinct if one is not a cyclic permutation of the other.

However the situation may get more difficult. One might think that when there is an elementary cycle without a deciding event it is sufficient to have an unbounded deciding event.

This situation can be seen in Figure 3.4. Node \( q \) is a non–local choice node and \( qv \) is an elementary cycle without a deciding event, but cycle \( qvqv \) possesses a deciding event (Figure 3.5). And it is clear that if we will iterate the cycle, we will generate new deciding events for the non–local choice node \( q \) and each of these events will be bounded.
We may leave the decision of the first occurrences of non–local choices on the deciding events on incoming acyclic paths, and when the cycles begin to generate their own deciding events, it will be possible to decide on the cycles.

It remains to characterize MSGs containing unbounded deciding events. And decide whether a cycle will generate deciding events.

### 3.2.1 Characterizing MSGs containing unbounded deciding events

As the property of having an unbounded deciding event is quite difficult to handle, we will prove the following theorem that characterizes those MSGs that contain an unbounded deciding event. We will assume from the previous part, that the MSG is already checked and it contains a deciding event.
3. Decidable Non–Local Choice

on every path from the initial node to the non–local choice node.

Theorem 1. Given an MSG $G = (S, \tau, s_0, s_f, L)$, with a deciding event on every path from $s_0$ to the non–local choice node $q$, then there exists an unbounded deciding event iff there exists an elementary cycle $c$ in graph $G' = (S, \tau)$ containing node $q$ such that $\forall n \geq 0$, MSC $M_{cn}$ doesn’t possess a deciding event.

Proof.

$\Leftarrow$ As node $q$ is reachable, there must exist a path $\sigma$ from initial node to node $q$, with a deciding event $e$. There also exists an elementary cycle $c$ containing node $q$ such that arbitrary iteration of the cycle $c$ does not contain a deciding event. It means that deciding event $e$ has to decide all occurrences of node $q$ and the number of occurrences may be arbitrary large.

$\Rightarrow$ We have an unbounded deciding event $e$ for non–local choice node $q$. Let $M$ denote the number of elementary cycles containing node $q$. And let $k \geq 0$ be an arbitrary but fixed number. Because $e$ is unbounded, there exists a path $\sigma$ in graph $G' = (S, \tau)$:

$$\sigma = u q c_1 q c_2 \ldots q c_{M-k+1} q$$

We require subpath $u$ to contain all deciding events for node $q$. We denote by $c_i$ (for all $i$) an arbitrary cycle starting in $q$ and ending in $q$, but there is no occurrence of $q$ in $c_i$. That means we have $M \cdot k + 1$ occurrences of node $q$ in subpath $c_1 q c_2 \ldots q c_{M-k+1} q$. And there is no deciding event for $q$ in this sequence.

We may replace every cycle $c_i$ by its accordant elementary cycle $c'_i$. By Proposition 1, the new path $c'_1 q c'_2 \ldots q c'_{M-k+1} q$ has no deciding event for $q$.

There exists at least one elementary cycle $c$, which has at least $k + 1$ occurrences in $c'_1 q c'_2 \ldots q c'_{M-k+1} q$. Using again Proposition 1, we may strip all other elementary cycles and obtain a path, with $k + 1$ elementary cycles $c$, without a deciding event. Therefore $\forall n \geq 0$, MSC $M_{cn}$ doesn’t possess a deciding event.

The previous theorem simplifies detecting of an unbounded deciding event. We need to generate all elementary cycles for a given graph, and check whether they will generate a deciding event. If this is the case we do not have an unbounded deciding event.
3. Decidable Non–Local Choice

3.2.2 Recognizing self-deciding cycles

We will show how to recognize that an elementary cycle will generate deciding events. It is sufficient to focus only on the communication type and the order of the messages is not important. Because by repeating the elementary cycle we can always achieve desired ordering. To handle communication type we introduce the following definition:

**Definition 17.** A communication graph for MSC $M = (E, <, P, T, P, M)$ is a directed graph $G_M = (P, E_M)$, where $E_M = \{(p, q) \mid \exists e_1, e_2 \in E : (e_1, e_2) \in M \land P(e_1) = p \land P(e_2) = q\}$

An example of a communication graph of MSC $M_{qvqvq}$ from Figure 3.5 can be seen on Figure 3.6.

![Figure 3.6: Communication graph](image)

We do not need to iterate the original MSC because we can decide the property from a communication graph as Lemma 3 shows.

**Lemma 3.** Let $c$ be an elementary cycle containing non–local choice node $q$, then some iteration of the cycle contains deciding event iff there exists a subgraph of the communication graph $G_c$ satisfying the following properties:

- $G_c$ is a tree and all the edges are oriented away from the root
- all the processes from set $\text{Trigger}(q)$ are in the tree

**Proof.**

⇒ Suppose there exists a number $n \geq 1$ such that MSC $M_{cn}$ contains a deciding event $e$ on a process $p$. Then for every process $r \in \text{Trigger}(q)$, there exists a sequence of messages that distributes the information from $p$ to $r$. There is an acyclic subgraph of the communication graph containing all processes from $\text{Trigger}(q)$. Spanning tree of the graph rooted in $p$ is a subtree with the needed properties.

⇐ There is a sequence of messages from some root process $p$ to every process $r \in \text{Trigger}(q)$. We may use these messages to distribute information from $p$ to $r$. All events on process $p$ are resolving events. Let $n$ be the height of the tree, then we need at most $n$ iterations of the cycle.
We have shown how to effectively check whether a given MSG $G$ with a non–local choice node $q$ contains an unbounded deciding event. The only remaining thing that needs to be shown is that there exists a bound $k$ not only for elementary cycles but for arbitrary cycles.

Let $C$ denote all elementary cycles in the strongly connected component of the MSG $G$ with the non–local choice node $q$. And let $Height(c)$ denote the minimal height of all subtrees (satisfying properties stated in Lemma 3) of the communication graph for elementary cycle $c$. Then we need at most

$$\sum_{c \in C} Height(c)$$

occurrences of node $q$, before a deciding event is created. As for all elementary cycles $c : Height(c) < |\mathcal{P}|$, we may generalize the upper bound $k$ to:

$$|C| \cdot |\mathcal{P}|$$

This chapter has shown that we are able to decide some non–local choice nodes by adding additional information into preceding communication. We wanted every deciding event to decide only finite occurrences of the non–local choice node, so that we may easily implement this feature. We have shown that we can algorithmically decide all the needed properties. In Chapter 4, an algorithm and some algorithmic aspects of the problem will be given.
Chapter 4

Algorithms

4.1 Listing elementary cycles

In our approach we need to enumerate all elementary cycles in a given directed graph. We will use an algorithm from [7]. We can slightly modify the algorithm, because we do not need all elementary cycles, but only cycles containing the non-local choice node $q$.

The complexity of Algorithm 2 lies in $O((n + e) \cdot (c + 1))$. Where $n$ is the number of vertices, $e$ number of edges and $c$ number of elementary cycles.

There may be at most:

$$\sum_{i=1}^{n-1} \binom{n}{n-i+1} (n-i)!$$

elementary cycles on $n$ vertices.

The graph $G = (V, E)$ is represented by an adjacency list $A_G(v)$ for all $v \in V$. To ensure no vertex is used more than once boolean array $\text{blocked}(v)$ is used.

4.1.1 Generating all acyclic paths

We may use this algorithm also to generate all acyclic paths from initial node to non-local choice node $q$ in MSG $G = (S, \tau, s_0, e_f, L)$. We construct a different graph $G' = (V', E')$, where

- $V' = S \cup \{v\}$, such that $v \notin S$
- $E' = \tau \cup \{(q, v), (v, s_0)\}$.

We enumerate all elementary cycles of graph $G$ containing the newly added vertex $v$. Every elementary cycle of $G$ has to contain vertices $s_0, q$ and uniquely identifies an acyclic path from $s_0$ to $q$. We can obtain this path by removing vertex $v$ from the cycle. It is easy to see, that every acyclic path can be constructed in this way.
Algorithm 2 Circuit finding algorithm

1. **Global variables:** stack, blocked, s, B
2. stack ← ∅
3. s ← non–local choice node q
4. for all \( i \in V \) do
5.   blocked\( (i) \) ← FALSE
6.   B\( (i) \) ← ∅
7. CIRCUIT\( (s) \)
8. 
9. procedure CIRCUIT\( (v) \)
10. \( f \) ← FALSE
11. stack ← push\( (v) \)
12. blocked\( (v) \) ← TRUE
13. for all \( w \in A_k(v) \) do
14.   if \( w = s \) then
15.     output circuit composed of stack followed by s
16.     \( f \) ← TRUE
17.   else if \( \neg \text{blocked}(w) \) then
18.     if CIRCUIT\( (w) \) then
19.       \( f \) ← TRUE
20. if \( f \) then
21.   UNBLOCK\( (v) \)
22. else
23.   for all \( w \in A_k(v) \) do
24.     if \( v \notin B(w) \) then
25.       B\( (w) \) ← B\( (w) \) ∪ \{v\}
26.     stack ← pop\( (v) \)
27.   return \( f \)
28. end procedure CIRCUIT
29. 
30. procedure UNBLOCK\( (u) \)
31. blocked\( (u) \) ← FALSE
32. for all \( w \in B(u) \) do
33.   B\( (u) \) ← B\( (u) \) − \{w\}
34.   if blocked\( (w) \) then
35.     UNBLOCK\( (w) \)
36. end procedure UNBLOCK\( (u) \)
4.2 MSC data structure

4.2.1 Concatenating MSCs

Most of our operations during the algorithm are concatenating MSCs along some path in MSG. Therefore we want these operations to be fast. We propose a data structure that allows us to concatenate two MSCs in time $O(|P|)$.

Every event set $E$ in an MSC $M = (E, <, P, T, P, M)$ with ordering $<$ is a directed acyclic graph. Moreover every vertex in the graph has out-degree less or equal than 2.

Therefore we may store events from $E$ in a double linked list and order the events according to some topological ordering.

Definition 18. Topological ordering of a partial ordered set $(E, \prec)$ is a linear order $(E, <)$ such that:

$$\forall e_1, e_2 \in E : e_1 \prec e_2 \Rightarrow e_1 < e_2$$

We need to store the outgoing edges of the vertex. We will do so by assigning two more integers to every vertex. We will call them $out_0, out_1$. The value denotes the relative position of the edge target in the double linked list. If there are less than two edges we will use a $\perp$ value.

When concatenating two MSCs we need to add some order between the maximal events from the earlier MSC and the minimal events from the latter MSC. Because finding such event may take linear time with respect to the number of events, additional structure in every MSC is used.

For MSC $M = (E, <, P, T, P, M)$, let $Min(M)$ be an array of positions of minimal events in the list. We will also add pointers to these events, to be able to directly access them. Let the array be sorted according to some fixed ordering of $P$. $Max(M)$ denotes an array of positions of maximal events with pointers to them. If there are no minimal or maximal events on the process in $M$ we will use $\emptyset$ for the position of the event. The last thing that needs to be stored is the length of the list (cardinality of the event set $E$) in $length(M)$. An example can be seen on Figure 4.1.

Given two MSCs $M_1$ and $M_2$ in the described structure, it will be shown how to compute MSC $M_{1, 2}$ in time $O(|P|)$.

1. Append $M_2$ list at $M_1$ (in $O(1)$).

2. $\forall p \in P$ add edge between the maximal event on $p$ in $M_1$ and the minimal in $M_2$, if there is some. Because we have pointers to these events and know the length of the lists, this takes only $O(|P|)$ time.
3. $\text{Max}(M_1 \cdot M_2)$ is equal to $\text{Max}(M_2)$ on positions with nonempty values. On empty positions values from $\text{Max}(M_1)$ are used. In $O(|P|)$.

4. $\text{Min}(M_1 \cdot M_2)$ is equal to $\text{Min}(M_1)$ on positions with nonempty values. On empty positions values from $\text{Min}(M_2)$ are used. In $O(|P|)$.

5. $\text{length}(M_1 \cdot M_2) = \text{length}(M_1) + \text{length}(M_2)$ (in $O(1)$)

The sum of all operations is in $O(|P|)$, so a concatenation of a path of length $n$ takes at most $O(n|P|)$ time.

4.2.2 Finding a deciding event in an MSC

Having an MSC $M$ it needs to be checked whether it possesses a deciding event for some non–local choice $q$. If we look at the event set $E$ as a partially
ordered set \((E, <)\), we are trying to find a supremum for events in \(\text{Max}(M)\) that are on processes from \(\text{First}(q)\).

This can be achieved in time \(O(|E| \cdot |P|)\). We allocate an array of size \(\text{length}(M)\) that will represent the events in their topological order and with constant access to elements. A subset of processes is assigned to every event in the following way: We start assigning from the array end. Every event \(e\) has a \(P(e)\) in its event set and gains all processes from events that are targets of its outgoing edges.

Because events are topologically ordered one pass through the events is enough. Moreover the first event that contains all processes from \(\text{First}(q)\) is a deciding event for \(q\).

4.3 Algorithm for decidable non–local choice

4.3.1 Highlighting deciding events

The answer whether a non–local choice node is decidable may not be enough. A designer of an MSG may need some further information. The Algorithm 3 depending on the decidability highlights various information about the MSG.

If the node is decidable, we highlight deciding events for every acyclic path from initial node to non–local choice node and a resolving event for every elementary cycle containing node \(q\), to show the designer where he should plan to add deciding information into the communication.

If the non–local choice node is not decidable we highlight a path from initial node to non–local choice node without a deciding event or an elementary cycle containing non–local choice node that does not generate its own deciding events.

4.3.2 Total correctness

We need to show that Algorithm 3 terminates, but that is clear because we iterate only over sets of finite size. Partial correctness follows from results in Chapter 3.

4.3.3 Complexity

We will analyze the complexity of the algorithm. Given an MSG \(G = (S, \tau, s_0, s_f, L)\) over a common process set \(P\). We will denote the sum of all events of all MSCs in the MSG by \(e\), \(c\) is the number of all elementary cycles containing non–local choice node \(q\) and \(p\) is the number of all acyclic paths from
Algorithm 3 Decidable non–local choice
1: input non-local choice node $q$
2: $c \leftarrow$ all acyclic paths
3: for all path $\in c$ do
4: if path contains a deciding event $e$ then
5: highlight $e$
6: else
7: highlight path
8: return $q$ is not decidable
9: $c \leftarrow$ all elementary cycles
10: for all cycle $\in c$ do
11: if cycle generates deciding events $e$ then
12: highlight $e$
13: else
14: highlight cycle
15: return $q$ is not decidable
16: return $q$ is decidable

the initial node to node $p$. Both constants $p$ and $q$ are bounded by:

$$\sum_{i=1}^{n-1} \binom{n}{n-i+1}(n-i)!$$

Generating all acyclic paths takes $O((|S| + |\tau|) \cdot (p+1))$ time. Constructing a path and test whether it contains a deciding event takes $O(|S| \cdot |P| + |P| \cdot e)$. Lines 1 – 8 together are in $O((|S| + |\tau|) \cdot (p + 1) + p(|S| \cdot |P| + |P| \cdot e))$. After rearrangement we get $O(p(|S| \cdot |P| + |P| \cdot e))$.

Generating all distinct elementary cycles takes $O((|S| + |\tau|) \cdot (c+1))$ time. Deciding whether an elementary cycle generate its own deciding events can be done by running BFS [15] for every process. The complexity of BFS is in $O(V + E)$, but because no vertex has out–degree and in–degree higher than 2, BFS runs on a path with $e$ events in $O(e)$. Together lines 9 – 16 run in $O((|S| + |\tau|) \cdot (c + 1) + c(|S| \cdot |P| + |P| \cdot e))$. That can be rearranged to $O(c(|P|(|S| + e) + |\tau|))$.

The complexity of the algorithm is the sum of the two previous partial complexities:

$$O((p + c)(|P|(|S| + e) + |\tau|))$$.
Chapter 5

Conclusion

We have presented a new approach to deal with non-local choice in MSG specifications. We assumed that we can add additional information into the communication that will help us to resolve the problem. We required the information to be bounded, to be able to easily implement such behavior.

Not all MSGs can be resolved in this way, therefore we introduced an algorithm that decides whether our approach can be applied to a specific non-local choice node. Moreover whenever our approach can be applied the algorithm advises the designer, where he should add information. When an MSG cannot be resolved we are able to print a witness, to help the designer easily solve the problem.

The complexity of the algorithm may be worse than exponential with respect to the number of nodes of the specification, but we believe that in common cases the algorithm will run acceptably fast.
Bibliography


5. CONCLUSION


