Lecture 4: Message Sequence Graphs
Theoretical Foundations of the UML
Lecture 3+4: Message Sequence Graphs

Joost-Pieter Katoen

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

moves.rwth-aachen.de/teaching/ss-20/fuml/

April 28, 2020
A Message Sequence Chart is a visual partial order:

- between send and receive events
- totally ordered per process
- receive events happen after their send events
- respecting the FIFO property

**Race:** in practice, the order of receive events cannot be guaranteed.

**Causal order:** send events should happen before their matching receive events.

The ordering of events with regards to sends on the same process is respected.

**AMSC Race:** if causal order = visual order. Checking whether an MSC has a race can be done in quadratic time (in number of events) using an optimized version of Warshall's algorithm.

Vertical ordering: receive events can only happen after their send events.

Horizontal ordering: receive events happen after their send events, respecting the FIFO property.

Non-FIFO: receive events can happen before their send events.
A Message Sequence Chart is a **visual** partial order
- between send and receive events
- totally ordered per process
- receive events happen after their send events
- respecting the FIFO property

**Race:** in practice, the order of receive events cannot be guaranteed

![Diagram of Message Sequence Chart with race](image)
Summary of Lecture #3

1. A Message Sequence Chart is a visual partial order \( \leq^* \) between send and receive events:
   - totally ordered per process
   - receive events happen after their send events (vertical ordering)
   - respecting the FIFO property (horizontal ordering)

2. Race: in practice, the order of receive events cannot be guaranteed.

3. Causal order \( \ll^* \):
   - send events should happen before their matching receive events
   - the ordering of events wrt. sends on same process is respected
   - receive events on a process sent from the same process are ordered as their sends

\[ \begin{array}{c}
n \end{array} \]
A Message Sequence Chart is a visual partial order:
- between send and receive events
- totally ordered per process
- receive events happen after their send events
- respecting the FIFO property

Race: in practice, the order of receive events cannot be guaranteed.

Causal order:
- send events should happen before their matching receive events
- the ordering of events wrt. sends on same process is respected
- receive events on a process sent from the same process are ordered as their sends

A MSC has a race if causal order $\neq$ visual order.
Summary of Lecture #3

1. A Message Sequence Chart is a **visual** partial order
   - between send and receive events
   - totally ordered per process
   - receive events happen after their send events
   - respecting the FIFO property

2. **Race:** in practice, the order of receive events cannot be guaranteed

3. **Causal order**
   - send events should happen before their matching receive events
   - the ordering of events wrt. sends on same process is respected
   - receive events on a process sent from the same process are ordered as their sends

4. A MSC has a **race** if causal order $\neq$ visual order
   - checking whether an MSC has a race can be done in quadratic time
     - (in number of events)
   - using an optimized version of Warshall’s algorithm

\[ O(E_1^3) \]
The need for composing MSCs

- An MSC describes a possible **single** scenario
- Typically: a **set** of scenarios

![Diagram of a MSC with a loop and a sequence of scenarios](image)

\[
\begin{align*}
\{M_1, \ldots, M_k\} & \quad k \in \mathbb{N} \\
\text{finite} & \\
M & \text{scenario} \\
\text{arbitrarily many times} & \\
M^* & \\
\text{MS Graph}
\end{align*}
\]
The need for composing MSCs

- An MSC describes a possible single scenario
- Typically: a set of scenarios
- and dependencies between these scenarios:
  - after scenario 1, scenario 2 occurs
  - after scenario 1, scenario 2 or 3 occurs
  - scenario 1 occurs repeatedly

This yields Message Sequence Graphs

Alternatives: ensembles of MSCs, high-level MSCs (MSC'96)
The need for composing MSCs

- An MSC describes a possible single scenario
- Typically: a set of scenarios
- and dependencies between these scenarios:
  - after scenario 1, scenario 2 occurs
  - after scenario 1, scenario 2 or 3 occurs
  - scenario 1 occurs repeatedly
- Need for: sequential composition (= concatenation), alternative composition, and iteration of MSCs
The need for composing MSCs

- An MSC describes a possible **single** scenario
- Typically: a **set** of scenarios
- and dependencies between these scenarios:
  - after scenario 1, scenario 2 occurs
  - after scenario 1, scenario 2 or 3 occurs
  - scenario 1 occurs **repeatedly**
- Need for: **sequential composition** (= concatenation), **alternative composition**, and **iteration** of MSCs

⇒ This yields **Message Sequence Graphs**

- Alternatives: ensembles of MSCs, high-level MSCs (MSC'96)

aka: hierarchical MSCs or message sequence chat automata
Message Sequence Graphs

initial vertex

MSG vertex

MSG edge

final vertex

Joost-Pieter Katoen  Theoretical Foundations of the UML
Let $\mathbb{M}$ be the set of MSCs (up to isomorphism, i.e., event identities).

**Definition**

A **Message Sequence Graph** (MSG) $G = (V, \rightarrow, v_0, F, \lambda)$ with:

- $(V, \rightarrow)$ is a digraph with finite set $V$ of vertices and $\rightarrow \subseteq V \times V$ a set of edges.
- $v_0 \in V$ is the starting (or: initial) vertex.
- $F \subseteq V$ is a set of final vertices.
- $\lambda : V \rightarrow \mathbb{M}$ associates to each vertex $v \in V$, an MSC $\lambda(v)$.

Note: An MSG can be considered as a non-deterministic finite-state automaton without input alphabet where states are MSCs. Obviously, every MSC is an MSG.
Let $\mathbb{M}$ be the set of MSCs (up to isomorphism, i.e., event identities).

**Definition**

A Message Sequence Graph (MSG) $G = (V, \rightarrow, v_0, F, \lambda)$ with:

- $(V, \rightarrow)$ is a digraph with finite set $V$ of vertices and $\rightarrow \subseteq V \times V$ a set of edges
- $v_0 \in V$ is the starting (or: initial) vertex
- $F \subseteq V$ is a set of final vertices
- $\lambda : V \rightarrow \mathbb{M}$ associates to each vertex $v \in V$, an MSC $\lambda(v)$

**Note:**

An MSG can be considered as a non-deterministic finite-state automaton without input alphabet where states are MSCs. Obviously, every MSC is an MSG.
Example

\[ g = (V, \rightarrow, v_0, F, \lambda) \]

\[ V = \{ u_0, \ldots, u_3 \} \]

\[ \rightarrow = \{ (u_0, u_1), (u_0, u_2), (u_2, u_0), (u_2, u_3) \} \]

\[ v_0 = u_0 \]

\[ F = \{ u_1, u_3 \} \]

\[ \lambda(u_0) = M_0 = \begin{pmatrix} c & o & n \end{pmatrix} \]

\[ \lambda(u_1) = \begin{pmatrix} g & r & a & t \end{pmatrix} \]

\[ \lambda(u_2) = \begin{pmatrix} g & r & a & t \end{pmatrix} \]

\[ \lambda(u_3) = \begin{pmatrix} a & c & k \end{pmatrix} \]
Let $M_i = (P_i, E_i, C_i, l_i, m_i, \preceq_i)$ with $i \in \{1, 2\}

be two MSCs with $E_1 \cap E_2 = \emptyset$
Concatenation of MSCs: definition

Let $M_i = (\mathcal{P}_i, E_i, C_i, l_i, m_i, \preceq_i)$ with $i \in \{1, 2\}$ be two MSCs with $E_1 \cap E_2 = \emptyset$

The concatenation of $M_1$ and $M_2$ is the MSC $M_1 \cdot M_2 = (\mathcal{P}, E, C, l, m, \preceq)$ with:

- $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$
- $E = E_1 \cup E_2$
- $C = C_1 \cup C_2$

(with $E_? = E_1,? \cup E_2,?$ etc.)

- $l(e) = \begin{cases} l_1(e) & \text{if } e \in E_1 \\ l_2(e) & \text{if } e \in E_2 \end{cases}$
- $m(e) = \begin{cases} m_1(e) & \text{if } e \in E_1 \\ m_2(e) & \text{if } e \in E_2 \end{cases}$

Joost-Pieter Katoen
Theoretical Foundations of the UML
Concatenation of MSCs: definition

Let $M_i = (\mathcal{P}_i, E_i, C_i, l_i, m_i, \preceq_i)$ with $i \in \{1, 2\}$ be two MSCs with $E_1 \cap E_2 = \emptyset$.

The concatenation of $M_1$ and $M_2$ is the MSC $M_1 \cdot M_2 = (\mathcal{P}, E, C, l, m, \preceq)$ with:

\begin{align*}
\mathcal{P} &= \mathcal{P}_1 \cup \mathcal{P}_2 \\
E &= E_1 \cup E_2 \\
C &= C_1 \cup C_2
\end{align*}

(with $E? = E_1,? \cup E_2,?$ etc.)

\begin{align*}
l(e) &= \begin{cases} 
l_1(e) & \text{if } e \in E_1 \\
l_2(e) & \text{if } e \in E_2 \end{cases} \\
m(e) &= \begin{cases} 
m_1(e) & \text{if } e \in E_1 \\
m_2(e) & \text{if } e \in E_2 \end{cases}
\end{align*}

\begin{align*}
\preceq &= (\preceq_1 \cup \preceq_2 \cup \{(e, e') | \exists p \in \mathcal{P}. e \in E_1 \cap E_p, e' \in E_2 \cap E_p\}^*)
\end{align*}
Concatenation of MSCs: observations

Ordering

\[ \preceq = (\preceq_1 \cup \preceq_2 \cup \{(e, e') \mid \exists p \in \mathcal{P}. e \in E_1 \cap E_p, e' \in E_2 \cap E_p\})^* \]

\[ \begin{array}{c}
M_1 \cdot M_2 \\
M_1 \\
M_2
\end{array} \]
Concatenation of MSCs: observations

Ordering

\[ \preceq = (\preceq_1 \cup \preceq_2 \cup \{(e, e') \mid \exists p \in \mathcal{P}. e \in E_1 \cap E_p, e' \in E_2 \cap E_p\})^* \]

Observations

- events are ordered per process:
  
  every event at \( p \) in MSC \( M_1 \) precedes every event at \( p \) in MSC \( M_2 \)
Concatenation of MSCs: observations

**Ordering**

\[ \preceq = (\preceq_1 \cup \preceq_2 \cup \{(e, e') \mid \exists p \in \mathcal{P}. e \in E_1 \cap E_p, e' \in E_2 \cap E_p\})^* \]

**Observations**

- Events are ordered per process:
  - Every event at \( p \) in MSC \( M_1 \) precedes every event at \( p \) in MSC \( M_2 \).
- Events at distinct processes in \( M_1 \) and \( M_2 \) can be incomparable.
Concatenation of MSCs: observations

**Ordering**

\[ \preceq = (\preceq_1 \cup \preceq_2 \cup \{(e, e') \mid \exists p \in P. e \in E_1 \cap E_p, e' \in E_2 \cap E_p\})^* \]

**Observations**

- events are ordered per process:
  every event at \( p \) in MSC \( M_1 \) precedes every event at \( p \) in MSC \( M_2 \)
- events at distinct processes in \( M_1 \) and \( M_2 \) can be incomparable
- thus: a process may start with \( M_2 \) before other processes do pause
- this differs from: first complete \( M_1 \), then start with \( M_2 \)

\[ \preceq \neq \text{execute all events in } M_1 \]

\[ \text{have finished } M_1 \]
Example (1)

\[ M_1: \]

\[ M_2: \]

\[ M_1 \circ M_2 \]
Example (1)

\[ M_1 \cdot M_2 \]

\[ \preceq_1 : e_1 \rightarrow e_2 \]

\[ \preceq_2 : e'_4 \rightarrow e'_3 \]

\[ M_1 : e_1 \rightarrow e_2 \]

\[ M_2 : e'_1 \rightarrow e'_2 \]

\[ M_1 \cdot M_2 \]
Example (2)

\[ \preceq_1: e_1 \rightarrow e_2 \]
\[ e'_2 \xleftarrow{\cdot} e'_1 \]
\[ \preceq_2: e'_4 \xleftarrow{\cdot} e'_3 \]

\[ e_1 \rightarrow e_2 \]
\[ e'_2 \xleftarrow{\cdot} e'_1 \]
\[ e'_4 \xleftarrow{\cdot} e'_3 \]

\[ M_1 \cdot M_2 \]
Example (2)

Note:
Events $e_1$ and $e'_1$ are not ordered in $M_1 \cdot M_2$

Example linearizations:

\[
\begin{align*}
e_1 & \quad e_2 & \quad e'_1 & \quad e'_2 & \ldots & \in Lin(M_1 \cdot M_2) \\
e'_1 & \quad e_1 & \quad e_2 & \quad e'_2 & \ldots & \in Lin(M_1 \cdot M_2)
\end{align*}
\]
Properties of concatenation

1 Concatenation is associative:

\[(M_1 \bullet M_2) \bullet M_3 = M_1 \bullet (M_2 \bullet M_3)\]

\[M_1 \bullet M_2 \bullet \ldots \bullet M_k\]
Properties of concatenation

1. Concatenation is **associative**:

\[(M_1 \cdot M_2) \cdot M_3 = M_1 \cdot (M_2 \cdot M_3)\]

2. Concatenation preserves the **FIFO** property:

\[(M_1 \text{ is FIFO } \land M_2 \text{ is FIFO }) \implies M_1 \cdot M_2 \text{ is FIFO}\]

3. **Race-freeness**, however, is not preserved

\[(M_1 \text{ is race-free } \land M_2 \text{ is race-free }) \not\Rightarrow M_1 \cdot M_2 \text{ is race-free}\]
Example

Race Freedom

\[ M_1: \]

\[ \leq_1: e_1 \rightarrow e_2 \]

\[ e'_2 \leftarrow e'_1 \]

\[ \leq_2: e'_4 \leftarrow e'_3 \]

\[ M_2: \]

\[ \leq: e'_2 \leftarrow e'_1 \]

\[ e'_4 \leftarrow e'_3 \]

\[ M_1 \cdot M_2 \]
Let $G = (V, \rightarrow, v_0, F, \lambda)$ be an MSG.

A path through MSG $G$ is a finite traversal through the graph $G$.

**Definition**

A path $\pi$ in MSG $G$ is a finite sequence

$$\pi = u_0 \ u_1 \ldots \ u_n$$

with $u_i \in V$ ($0 \leq i \leq n$) and $u_i \rightarrow u_{i+1}$ ($0 \leq i < n$)

$(u_i, u_{i+1}) \in \rightarrow$
Let $G = (V, \rightarrow, v_0, F, \lambda)$ be an MSG.

A path through MSG $G$ is a finite traversal through the graph $G$.

**Definition**

A path $\pi$ in MSG $G$ is a finite sequence

$$\pi = u_0 \ u_1 \ldots \ u_n$$

with $u_i \in V$ ($0 \leq i \leq n$) and $u_i \rightarrow u_{i+1}$ ($0 \leq i < n$)

An accepting path through MSG $G$ is a path starting in the initial vertex and ending in a final vertex.

**Definition**

Path $\pi = u_0 \ldots u_n$ is accepting if: $u_0 = v_0$ and $u_n \in F$. 
Paths in an MSG represent MSCs

Let $G = (V, \rightarrow, v_0, F, \lambda)$ be an MSG.

**Definition**

The **MSC** of a path $\pi = u_0 \ldots u_n$ through MSG $G$ is defined by:

$$M(\pi) = \lambda(u_0) \cdot \lambda(u_1) \cdot \ldots \cdot \lambda(u_n)$$

- MSC of $u_0$
- MSC of $u_1$
- MSC of $u_n$
Example paths

Accepting: \( u_0 u_1 \)  
\( u_0 u_2 u_0 u_3 \)

Non-accepting: \( u_0 u_2 u_0 u_2 u_3 \) etc.

\( M(u_0 u_2 u_0 u_3) \)
Language of an MSG

- set of MSCs that are accepted by the MSG

- MSG $G$ as a descriptor a set of MSCs

finite

Infinite
The language of an MSG, i.e., the set of MSCs it represents, is the set of MSCs of its accepting paths.

**Definition**

The MSC language of MSG $G$ is defined by:

$$L(G) = \{ M(\pi) \mid \pi \text{ is an accepting path of } G \}.$$
The language of an MSG, i.e., the set of MSCs it represents, is the set of MSCs of its accepting paths.

**Definition**

The **MSC language** of MSG $G$ is defined by:

$$L(G) = \{ M(\pi) \mid \pi \text{ is an accepting path of } G \}.$$

**Definition**

The **word language** of MSG $G$ is defined by $Lin(L(G))$ where

$$Lin(\{M_1, \ldots, M_k\}) = \bigcup_{i=1}^{k} Lin(M_i).$$
Example

\[ L(G) = \{ M(u_0(u_2u_0)^*u_1), M(u_0(u_2u_0)^*u_2u_3) \} \]

\[ |L(G)| = \infty \]
Recall: MSC $M$ has a race if $\leq \not\ll^*$
Races in MSGs

Recall: MSC $M$ has a race if $\leq \not\subset \ll^*$

or, equivalently $\text{Lin}(M, \leq) \not\subset \text{Lin}(M, \ll^*)$

Linearisations wrt $\leq$ $\neq$ linearisations wrt $\ll^*$
Races in MSGs

Recall: MSC $M$ has a race if $\leq \subsetneq \ll^*$

or, equivalently $\text{Lin}(M, \leq) \subsetneq \text{Lin}(M, \ll^*)$

or, equivalently $\text{Lin}(M, \ll^*) \subset \text{Lin}(M, \leq)$

**Definition**

MSG $G$ has a race if $\text{Lin}(G, \ll^*) \subset \text{Lin}(G, \leq)$

MSG $g$ has race if some MSC $M \in L(G)$ has a race.
Example

**Definition**

MSG $G$ has a race if $\text{Lin}(G, \ll^*) \subset \text{Lin}(G, \leq)$

MSG $G$ has a race.
Deciding whether an MSG has a race is undecidable

\( \text{Theorem} \) [Muscholl & Peled, 1999]

The decision problem "does MSG \( G \) have a race?" is undecidable.

**Proof.** By a reduction from the universality of semi-trace languages. Requires some Mazurkiewicz' trace theory. Omitted here. We will see other reduction proofs later on.

No undecidable problem can ever be solved by a computer or computer program of any kind.

Does MSC \( M \) have a race? \( \rightarrow \) Warshall algorithm

\[ O(1E1^2) \]

\( \forall \text{ events } m \cdot M \)

Does MSG \( G \) have a race?

- if \( L(G) = \{ M_1, \ldots, M_k \} \) \( k \in \mathbb{N} \).

  run Warshall's on each \( M_i \)

  \[ O( k \cdot 1E1^2) \]

- if \( L(G) = \{ M \} \)

\( \rightarrow \) Worsham's algorithm.
Deciding whether an MSG has a race is undecidable

**Theorem**

[Muscholl & Peled, 1999]

The decision problem “does MSG $G$ have a race?” is undecidable.

**Proof.**

By a reduction from the universality of semi-trace languages. Requires some Mazurkiewicz’ trace theory. Omitted here. We will see other reduction proofs later on.

$L = \Sigma^*$ ? universality problem
Deciding whether an MSG has a race is undecidable

Theorem [Muscholl & Peled, 1999]

The decision problem “does MSG $G$ have a race?” is undecidable.

Proof.

By a reduction from the universality of semi-trace languages. Requires some Mazurkiewicz’ trace theory. Omitted here. We will see other reduction proofs later on.

No undecidable problem can ever be solved by a computer or computer program of any kind.
Do MSGs have an MSC in common?

Theorem: undecidability of empty intersection

The decision problem:

for MSGs $G_1$ and $G_2$, do we have $L(G_1) \cap L(G_2) = \emptyset$?

is undecidable.

Do MSGs $G_1$ and $G_2$ describe at least one common MSC?
Do MSGs have an MSC in common?

Theorem: undecidability of empty intersection

The decision problem:

for MSGs $G_1$ and $G_2$, do we have $L(G_1) \cap L(G_2) = \emptyset$?

is undecidable.

Proof: Reduction from Post’s Correspondence Problem (PCP)

\[ \text{... black board ...} \]

next lecture