Lecture 2: Races

- Phenomenon in MSCs that complicates their interpretation
- Formal definition
- Algorithm: Input: MSC, Output: MSC has a race?
A Message Sequence Chart is a **partial order**
- between send and receive events
- totally ordered per process
- receive events happen after their send events
- respecting the first-in first out (FIFO) property
Summary of Lecture #1

1. **Message Sequence Chart** is a partial order
   - between send and receive events
   - totally ordered per process
   - receive events happen after their send events
   - respecting the first-in first out (FIFO) property

2. **Linearizations** are totally ordered extensions of partial orders
   - all linearizations of an MSC are well-formed
     - every receive is preceded by a corresponding send
     - respects the FIFO ordering
     - no send events without corresponding receive
Summary of Lecture #1

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

2. **Linearizations** are totally ordered extensions of partial orders
   - all linearizations of an MSC are well-formed
     1. every receive is preceded by a corresponding send
     2. respects the FIFO ordering
     3. no send events without corresponding receive

3. **Every well-formed word can be transformed** into an MSC
   - two linearizations of the same MSC yield isomorphic MSCs
Summary of Lecture #1

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

2. **Linearizations** are totally ordered extensions of partial orders
   - all linearizations of an MSC are **well-formed**
     1. every receive is preceded by a corresponding send
     2. respects the FIFO ordering
     3. no send events without corresponding receive

3. Every well-formed word can be **transformed** into an MSC
   - two linearizations of the same MSC yield **isomorphic** MSCs

4. So: there is a **1-to-1 relation** between an MSC and its linearizations
Example

\[ \ell(e) = !(p_1, p_2, a) \]
\[ \ell(e') = ?(p_2, p_1, a) \]
These pictures are formalized using **partial orders**.
An MSC $M = (\mathcal{P}, E, C, l, m, \preceq)$ with:
An MSC $M = (\mathcal{P}, E, C, l, m, \preceq)$ with:

- $\mathcal{P}$, a finite set of processes \{p_1, p_2, \ldots, p_n\}

\begin{center}
\begin{tikzpicture}
\draw (0,0) -- (0,5);
\draw (1,0) -- (1,5);
\draw (2,0) -- (2,5);
\node at (0,5.5) {$p_1$};
\node at (1,5.5) {$p_2$};
\node at (2,5.5) {$p_n$};
\end{tikzpicture}
\end{center}
Message Sequence Chart (MSC) (1)

Definition

An MSC $M = (\mathcal{P}, E, C, l, m, \preceq)$ with:

- $\mathcal{P}$, a finite set of processes $\{p_1, p_2, \ldots, p_n\}$
- $E$, a finite set of events

$E = \bigcup_{p \in \mathcal{P}} E_p = E? \cup E!$

- vertically
- horizontally
Definition

An MSC $M = (\mathcal{P}, E, \mathcal{C}, l, m, \preceq)$ with:

- $\mathcal{P}$, a finite set of processes $\{p_1, p_2, \ldots, p_n\}$
- $E$, a finite set of events

$$E = \bigcup_{p \in \mathcal{P}} E_p = E? \cup E!$$

- $\mathcal{C}$, a finite set of message contents

\[ \text{a, b, c} \]

\[ \text{a} \]
Definition

An MSC $M = (P, E, C, l, m, \preceq)$ with:

- $P$, a finite set of processes $\{p_1, p_2, \ldots, p_n\}$
- $E$, a finite set of events

$$E = \bigcup_{p \in P} E_p = E? \cup E!$$

- $C$, a finite set of message contents
- $l : E \rightarrow Act$, a labelling function defined by:

$$l(e) = \begin{cases} !(p,q,a) & \text{if } e \in E_p \cap E! \\ ?(p,q,a) & \text{if } e \in E_p \cap E? \end{cases}, \text{ for } p \neq q \in P, a \in C$$
Message Sequence Chart (MSC) (2)

Definition

- \( m : E! \rightarrow E? \) a bijection ("matching function"), satisfying:

\[
    m(e) = e' \land l(e) = !(p, q, a) \text{ implies } l(e') = ?(q, p, a) \quad (p \neq q, \ a \in C)
\]

\[ \begin{array}{c}
\text{e} \quad \text{m} \quad \text{e}' \\
\end{array} \]

\[ \begin{array}{c}
\text{e} \quad \text{m}^{-1} \quad \text{e}' \\
\end{array} \]

\[
    m(e) = e'
\]
Definition

- $m : E! \rightarrow E?$ a bijection ("matching function"), satisfying:

$$m(e) = e' \land l(e) = !(p, q, a) \implies l(e') = ?(q, p, a) \ (p \neq q, \ a \in C)$$

- $\preceq \subseteq E \times E$ is a partial order ("visual order") defined by:

$$\preceq = \left( \bigcup_{p \in P} \prec_p \bigcup \left\{ (e, m(e)) \mid e \in E! \right\} \right)^*$$

$\prec_p$ is a total order = "top-to-bottom" order on process $p$

communication order $\prec_c$

where for relation $R$, $R^*$ denotes its reflexive and transitive closure.
Example

Hasse diagram

\[
\begin{align*}
\leq_p : & \quad e_0 \leq_p e_5 \\
\leq_a : & \quad e_1 \leq_a e_2 \\
\leq_r : & \quad e_3 \leq_r e_4 \\
\end{align*}
\]

\[
\begin{align*}
m(e_2) &= e_3 \\
m(e_5) &= e_4 \\
\end{align*}
\]
Visual order can be misleading
Visual order can be misleading

If message $b$ takes much shorter than message $a$, then $c$ might arrive at $p_1$ before $a$. 
Visual order can be misleading

If message $b$ takes much shorter than message $a$, then $c$ might arrive at $p_1$ before $a$.

In practice, $e_6$ might occur before $e_2$, but $e_2 <_{p_1} e_6$ and thus $e_2 \preceq e_6$. This is misleading and called a race.
A race condition asserts a particular order of events will occur because of the visual ordering (i.e., the partial order $\leq$) when, in practice, this order cannot be guaranteed to hold.
What is a race?

A race condition asserts a particular order of events will occur because of the visual ordering (i.e., the partial order $\preceq$) when, in practice, this order cannot be guaranteed to hold.

Q: When are race conditions possible and how to detect them?

formally define what is a race?

algorithm MSC $M$

input $i$

output:

$M$ has a race or not.
Causal order

| defined in a different way than |
| den | visual order | part of the MSC definition.
Causal order

Main principles:
1. Send events should happen before their matching receive events
2. The ordering of events with respect to sends on the same process is unaffected
3. Receive events on a process sent from the same process are ordered as their sends

Definition

For MSC $M = (P, E, C, l, m, \preceq)$, relation $\preceq \subseteq E \times E$ is defined by:

1. $e \preceq e'$ iff $e' = m(e)$

\[ \begin{array}{ccc}
\text{\small{e}} & \downarrow & \text{\small{e'}} \\
\downarrow & & \downarrow \\
\text{\small{e''}} & & \text{\small{e <\!\!\!< e'}}
\end{array} \]
Causal order

Main principles:
- Send events should happen before their matching receive events
- The ordering of events wrt. sends on same process is unaffected
- Receive events on a process sent from the same process are ordered as their sends

Definition

For MSC $M = (\mathcal{P}, E, C, l, m, \preceq)$, relation $\ll \subseteq E \times E$ is defined by:

$$e \ll e' \quad \text{iff} \quad e' = m(e)$$

or

$$e <_p e' \quad \text{and} \quad E! \cap \{e, e'\} \neq \emptyset$$
e \neq e' \text{ because there is no process } u \text{ such that } m^{-1}(e) \lessdot u m^{-1}(e')

as \( m^{-1}(e) \) and \( m^{-1}(e') \) occur at different processes
Causal order

Main principles:

- Send events should happen before their matching receive events
- The ordering of events wrt. sends on same process is unaffected
- Receive events on a process sent from the same process are ordered as their sends

Definition

For MSC $M = (\mathcal{P}, E, C, l, m, \preceq)$, relation $\preceq \subseteq E \times E$ is defined by:

\[ e \preceq e' \iff e' = m(e) \]

or $e <_p e'$ and $E! \cap \{e, e'\} \neq \emptyset$

or $e, e' \in E_p \cap E_q$ and $m^{-1}(e) <_q m^{-1}(e')$

both at $p$

both at $q$
Causal order

Main principles:

- Send events should happen before their matching receive events
- The ordering of events wrt. sends on same process is unaffected
- Receive events on a process sent from the same process are ordered as their sends

**Definition**

For MSC $M = (P, E, C, l, m, \preceq)$, relation $\ll \subseteq E \times E$ is defined by:

\[
e \ll e' \quad \text{iff} \quad e' = m(e)
\]

or $e <_p e'$ and $E! \cap \{e, e'\} \neq \emptyset$

or $e, e' \in E_p \cap E?$ and $m^{-1}(e) <_q m^{-1}(e')$

$\ll^*$ is a partial order (referred to as **causal order**) in which events at the same process are not necessarily ordered.
Causal order: example

**Definition**

For MSC $M = (\mathcal{P}, E, C, l, m, \preceq)$, relation $\ll \subseteq E \times E$ is defined by:

- $e \ll e'$ if $e' = m(e)$
- or $e <_p e'$ and $E! \cap \{e, e'\} \neq \emptyset$
- or $e, e' \in E_p \cap E?$ and $m^{-1}(e) <_q m^{-1}(e')$

Joost-Pieter Katoen  Theoretical Foundations of the UML
Causal order: example

Definition

For MSC $M = (\mathcal{P}, E, C, l, m, \preceq)$, relation $\ll \subseteq E \times E$ is defined by:

\begin{align*}
e \ll e' & \quad \text{iff} \quad e' = m(e) \\
or & \quad e <_p e' \quad \text{and} \quad E! \cap \{e, e'\} \neq \emptyset \\
or & \quad e, e' \in E_p \cap E? \quad \text{and} \quad m^{-1}(e) <_q m^{-1}(e')
\end{align*}
Causal order: example

Definition

For MSC $M = (\mathcal{P}, E, C, l, m, \preceq)$, relation $\ll \subseteq E \times E$ is defined by:

$$e \ll e' \text{  iff  } e' = m(e)$$

or

$$e <_p e' \text{ and } E! \cap \{e, e'\} \neq \emptyset$$

or

$$e, e' \in E_p \cap E? \text{ and } m^{-1}(e) <_q m^{-1}(e')$$

Example

Joost-Pieter Katoen

Theoretical Foundations of the UML

11/23
Causal order: example

**Definition**

For MSC $M = (\mathcal{P}, E, \mathcal{C}, l, m, \preceq)$, relation $\ll \subseteq E \times E$ is defined by:

$$e \ll e' \quad \text{iff} \quad e' = m(e) \quad \text{①}$$

or $e <_p e'$ and $E! \cap \{e, e'\} \neq \emptyset$

or $e, e' \in E_p \cap E?$ and $m^{-1}(e) <_q m^{-1}(e')$

**Example**

$e_1 \ll e_2$, $e_3 \ll e_4$, $e_5 \ll e_6$, ①

$m(e_1) = e_2$  \[e_1 \ll e_2\]
Causal order: example

**Definition**

For MSC $M = (\mathcal{P}, E, C, l, m, \preceq)$, relation $\ll \subseteq E \times E$ is defined by:

\[
e \ll e' \quad \text{iff} \quad e' = m(e)
\]

or \( e <_p e' \) and \( E_1 \cap \{e, e'\} \neq \emptyset \)

or \( e, e' \in E_p \cap E_? \) and \( m^{-1}(e) < q m^{-1}(e') \)

**Example**

$e_1 \ll e_2$, $e_3 \ll e_4$, $e_5 \ll e_6$, $e_1 \ll e_3$, $e_4 \ll e_5$

\(e_2\) \(a\) \(e_1\) \(e_3\) \(b\) \(e_4\) \(c\) \(e_5\)

$e_4 <_p e_5$ \(\{e_4, e_5\} \cap E_1 \neq \emptyset\) \(e_4 \ll e_5\)
Definition

For MSC $M = (\mathcal{P}, E, \mathcal{C}, l, m, \leq)$, relation $\ll \subseteq E \times E$ is defined by:

$e \ll e'$ iff

1. $e' = m(e)$
2. $e <_p e'$ and $E! \cap \{e, e'\} \neq \emptyset$
3. $e, e' \in E_p \cap E?$ and $m^{-1}(e) <_q m^{-1}(e')$

Example

$e_1 \ll e_2$, $e_3 \ll e_4$, $e_5 \ll e_6$, $e_1 \ll e_3$, $e_4 \ll e_5$, not $(e_2 \ll e_6)$
Races

Definition

MSC $M$ contains a race if for some $e, e' \in E$ and process $p$:

$$e <_p e' \text{ but not } (e \ll^* e')$$

where $\ll^* \subseteq E \times E$ is the reflexive and transitive closure of $\ll$.

As relation $\ll^*$ contains at most all orderings in $\preceq$, the MSC $M$ has a race whenever $\preceq \nsubseteq \ll^*$. 

Visual order

causal order
Race: example
Race: example

Visual order versus causal order

1. $e_1 \preceq e_2, \ e_3 \preceq e_4, \ e_5 \preceq e_6, \ e_1 \preceq e_3, \ e_4 \preceq e_5, \ e_2 \preceq e_6$

2. $e_1 \ll e_2, \ e_3 \ll e_4, \ e_5 \ll e_6, \ e_1 \ll e_3, \ e_4 \ll e_5, \ \text{not} \ (e_2 \ll e_6)$

As $\preceq \not\subseteq \ll^*$, this MSC contains a race.
Other examples

On the black board.
MSC has a race

\[ \ll : \ll + 2 \leq b \]

because

\[ 2 \ll_p b \]

not \[ 2 \ll_b 6 \]

MSC has no race.

\[ \ll : \ 1 \ll 2 , \ 3 \ll 4 , \ 5 \ll 6 , \ 2 \ll 5 , \ 1 \ll 3 , \ 3 \ll 6 \]

= \leq \ text{visual order}
Why are races problematic?

Recall: MSC $M$ has a race if $\preceq \not\subseteq \ll^*$ or equivalently:

$$\exists e, e' \in E? . (e <_p e' \text{ and } e \ll^* e')$$

Whenever $\preceq \not\subseteq \ll^*$, implementations based on $<_p$ may cause problems:

1. **unspecified message reception**
   - a process receives a message which by the MSC is not possible

2. **deadlocks**
   - a process blocking on receipt of an unexpected message may block others too

3. **message loss**
   - unexpectedly received messages may be ignored

4. **exploiting wrong message content**
Checking whether an MSC has a race for digraphs without negative cycles.

Joost-Pieter Katoen

Theoretical Foundations of the UML
Checking whether an MSC has a race

- MSC $M$ has a race if $\leq \subseteq \ll^*$

- How to check whether MSC $M$ has a race?
  - Compute $\ll^*$ and check whether $\leq \subseteq \ll^*$

- Transitive closure $\ll^*$ is computed using Floyd-Warshall’s algorithm
  - Algorithm for finding shortest paths in a weighted digraph with positive or negative edge weights\(^1\)
  - Easily adapted for computing the transitive closure of digraphs
  - Worst-case time complexity $O(|E|^3)$
  - By using some specifics of MSC, this is reduced to $O(|E|^2)$

- So: race checking can be done quadratically in the number of events

---

\(^1\) for digraphs without negative cycles.
Computing $\leq^*$: Warshall’s algorithm

**Algorithm**

compute $\leq^*$ and compare with $\leq$

Warshall’s algorithm:

**input:** $R \subseteq X \times X$ where $X$ is a set

**output:** $R^*$

$X = E$ for MsCs
Computing $\ll^*$: Warshall’s algorithm

Algorithm

- compute $\ll^*$ and compare with $\leq$
- Warshall’s algorithm

Warshall’s algorithm:

- input: $R \subseteq X \times X$ where $X$ is a set
- output: $R^*$

Idea:

Consider $R$ and $R^*$ as directed graphs

There is an edge $x \Rightarrow y$ in $R^*$ iff there is a (possibly empty) sequence

$x = x_0 \to x_1 \to x_2 \to \ldots \to x_n = y$ in $R$

(our setting: $X = E$, $R = \ll$, $R^* = \ll^*$)
Warshall’s algorithm: preliminaries
assume: graph vertices are numbered \( \{1, 2, \ldots, n\} \) where \( n = |E| = |X| \)

\[
\begin{align*}
R \quad X &= \{x_1, \ldots, x_n\} \\
R &= \{(x_1, x_2), (x_3, x_4), (x_2, x_4)\}
\end{align*}
\]

**Graph** \( (R) \):

\[
\begin{array}{c}
(x_1) \rightarrow (x_2) \\
(x_3) \rightarrow (x_4)
\end{array}
\]
Warshall’s algorithm: preliminaries

- assume: graph vertices are numbered \( \{1, 2, \ldots, n\} \) where \( n = |E| \)
- for \( j \in \{1, \ldots, n+1\} \) define relation \( \rightarrow j \) as follows:
  \( x \rightarrow j y \) iff \( \exists \) path in \( R \) from \( x \) to \( y \) such that all vertices on the path \( (\neq x, y) \) have a smaller number than \( j \)
Warshall’s algorithm: preliminaries

- assume: graph vertices are numbered \(\{1, 2, \ldots, n\}\) where \(n = |E|\)
- for \(j \in \{1, \ldots, n+1\}\) define relation \(\overset{j}{\rightarrow}\) as follows:
  \(x \overset{j}{\rightarrow} y\) iff \(\exists\) path in \(R\) from \(x\) to \(y\) such that all vertices on the path \((\neq x, y)\) have a smaller number than \(j\)

Then:
1. \(\boxed{x \overset{n+1}{\rightarrow} y}\) iff \(x \overset{n}{\rightarrow} y\)
2. \(x \overset{1}{\rightarrow} y\) iff \(x = y\) or \(x \ll y\)
3. \(x \overset{y+1}{\rightarrow} z\) iff \(x \overset{y}{\rightarrow} z\) or \(x \overset{y}{\rightarrow} y \overset{y}{\rightarrow} z\)

\(\overset{j}{\rightarrow}\) by induction and \(j = 1\) stat \(x \overset{j}{\rightarrow} y\)
Warshall’s algorithm: preliminaries

- Assume: graph vertices are numbered \{1, 2, \ldots, n\} where \( n = |E| \)

- For \( j \in \{1, \ldots, n+1\} \) define relation \( \rightarrow^j \) as follows:
  \( x \rightarrow^j y \) iff \( \exists \) path in \( R \) from \( x \) to \( y \) such that all vertices on the path (\( \neq x, y \)) have a smaller number than \( j \)

- Then: (1) \( x \rightarrow y \) iff \( x \nrightarrow^{n+1} y \)
  (2) \( x \rightarrow y \) iff \( x = y \) or \( x \ll y \)
  (3) \( x \rightarrow^{y+1} z \) iff \( x \rightarrow z \) or \( x \rightarrow y \rightarrow y \rightarrow z \)

- Algorithm: determine the relations \( \rightarrow^1, \ldots, \rightarrow^n, \rightarrow^{n+1} \) iteratively using properties (2) + (3);
Warshall’s algorithm: preliminaries

- assume: graph vertices are numbered \{1, 2, \ldots, n\} where \(n = |E|\)

- for \(j \in \{1, \ldots, n+1\}\) define relation \(\xrightarrow{j}\) as follows:
  \(x \xrightarrow{j} y\) iff \(\exists\) path in \(R\) from \(x\) to \(y\) such that all vertices on the path \((\neq x, y)\) have a smaller number than \(j\)

Then:

1. \(x \xrightarrow{n+1} y\) iff \(x \xrightarrow{n} y\)
2. \(x \xrightarrow{1} y\) iff \(x = y\) or \(x \ll y\)
3. \(x \xrightarrow{y+1} z\) iff \(x \xrightarrow{y} z\) or \(x \xrightarrow{y} y \xrightarrow{y} z\)

Algorithm: determine the relations \(\xrightarrow{1}, \ldots, \xrightarrow{n}, \xrightarrow{n+1}\) iteratively using properties (2) + (3); Result is then given by (1).

- Store \(\xrightarrow{i}\) in a boolean matrix \(C\) of cardinality \(|E| \times |E|\)
Warshall’s algorithm: preliminaries

- Assume: graph vertices are numbered \(\{1, 2, \ldots, n\}\) where \(n = |E|\)

- For \(j \in \{1, \ldots, n+1\}\) define relation \(\xrightarrow{j}\) as follows:
  \[x \xrightarrow{j} y \text{ iff } \exists \text{ path in } R \text{ from } x \text{ to } y \text{ such that all vertices on the path } (\neq x, y) \text{ have a smaller number than } j\]

- Then:
  \(\begin{align*}
  & \text{(1)} \quad x \xrightarrow{1} y \text{ iff } x \xrightarrow{n+1} y \\
  & \text{(2)} \quad x \xrightarrow{1} y \text{ iff } x = y \text{ or } x \ll y \\
  & \text{(3)} \quad x \xrightarrow{y+1} z \text{ iff } x \xrightarrow{y} z \text{ or } x \xrightarrow{y} y \xrightarrow{y} z
  \end{align*}\)

- Algorithm: determine the relations \(\xrightarrow{1}, \ldots, \xrightarrow{n}, \xrightarrow{n+1}\) iteratively using properties (2) + (3); Result is then given by (1).

- Store \(\xrightarrow{i}\) in a boolean matrix \(C\) of cardinality \(|E| \times |E|\)

- Postcondition: \(C[x, y] = \text{true}\) iff \((x, y) \in R^*\)

- Precondition: \(\forall x, y \in X \cdot C[x, y] = \text{false}\)
Warshall's algorithm

/* first compute $x \xrightarrow{1} y$ */
for $x := 1$ to $n$ do
  for $y := 1$ to $n$ do
    $C[x, y] := (x = y$ or $(x, y) \in R$) $x \leqslant y$

/* loop invariant: after the $j$-th iteration of */
/* outermost loop it holds: $C[x, y] = \text{true}$ iff $x \xrightarrow{j+1} y$ */

1. for $y := 1$ to $n$ do
   for $x := 1$ to $n$ do
     if $C[x, y]$ then
       for $z := 1$ to $n$ do
         if $C[y, z]$ then
           $C[x, z] := \text{true}$

2. /* initialisation */

3. loop
Correctness and complexity

Lemma: correctness

After \( j \) iterations: \( x \xrightarrow{j+1} y \) iff \( C[x, y] = \text{true} \).

Proof.

*if*: trivial; *only if*: by induction on \( j \).

Joost-Pieter Katoen  Theoretical Foundations of the UML  20/23
Claim: after $j$ iterations (for any $0 \leq j \leq n$):

$$k \xrightarrow{j+m} m \implies C[k,m] = 1$$

Proof: by induction on $j$.

1) base case: $j=0$: it follows from the initialisation.

2) Ind. step: Let $j > 0$ and assume $k \xrightarrow{j+m} m$.

   a) if $C[k,m] = 1$, done $\checkmark$ $k \xrightarrow{j} m$
   
   b) assume $C[k,m] = 0$. Then by ind. hyp., it follows $k \not\xrightarrow{j} m$. But since $k \xrightarrow{j+m} m$

   iff $k \xrightarrow{j} m$ or $k \xrightarrow{j} j \xrightarrow{j} m$ (by (3))

   it follows $k \xrightarrow{j} j \xrightarrow{j} m$.

   Thus $C[k,j] = \text{true}$ and $C[j,m] = \text{true}$

   Then during the $j$-th iteration $C[k,m]$ is set to true $\blacksquare$
Correctness and complexity

Lemma: correctness

After $j$ iterations: $x \xrightarrow{j+1} y$ iff $C[x, y] = \text{true}$.  

Proof.

if: trivial; only if: by induction on $j$.

Complexity

Worst-case time complexity of Warshall’s algorithm: $O(n^3)$ with $n = |X|$

Proof.

follows from the fact that there is a triple-nested loop of which each loop has at most $n$ iterations.
Warshall’s algorithm computes $R^*$ for every binary relation $R \subseteq X \times X$. 

arbitrary
Warshall’s algorithm computes $R^*$ for every binary relation $R \subseteq X \times X$.

Recall: our interest is in determining $R^*$ for $R$.
Warshall’s algorithm computes $R^*$ for every binary relation $R \subseteq X \times X$. Recall: our interest is in determining $R^*$ for $R = \ll$. Using some properties of $\ll$, the complexity can be improved.

$O(n^3)$
Warshall’s algorithm computes $R^*$ for every binary relation $R \subseteq X \times X$.

Recall: our interest is in determining $R^*$ for $R = \ll$

Using some properties of $\ll$, the complexity can be improved.

Exploit that for $\ll$: 

\[ \text{[Alur et al. '96]} \]
Efficiency improvement

Warshall’s algorithm computes $R^*$ for every binary relation $R \subseteq X \times X$.

Recall: our interest is in determining $R^*$ for $R = \ll$

Using some properties of $\ll$, the complexity can be improved.

Exploit that for $\ll$:

1. $\ll$ is acyclic (as it is a partial order)
2. the number of immediate predecessors of $e \in E$ under $\ll$ is at most two

Note that $e$ is an immediate predecessor of $e'$ (under $\ll$) if:

$$e \ll e' \text{ and } \neg(\exists e'' \notin \{e, e'\}. e \ll e'' \land e'' \ll e')$$
The main loop of Warshall’s algorithm:

\[
\text{for } e := 1 \text{ to } n \text{ do }
\left\{ \begin{array}{c}
\text{for } e' := 1 \text{ to } n \text{ do }
\quad \text{if } C[e', e] \text{ then }
\left\{ \begin{array}{c}
\text{for } e'' := 1 \text{ to } n \text{ do }
\quad \text{if } C[e, e''] \text{ then }
\quad C[e', e''] := \text{true}
\end{array} \right.
\end{array} \right.
\]
Efficiency improvement [Alur et al. ’96]

The main loop of Warshall’s algorithm:

\[
\text{for } e := 1 \text{ to } n \text{ do } \\
\hspace{1cm} \text{for } e' := 1 \text{ to } n \text{ do } \\
\hspace{2cm} \text{if } C[e', e] \text{ then } \\
\hspace{3cm} \text{for } e'' := 1 \text{ to } n \text{ do } \\
\hspace{4cm} \text{if } C[e, e''] \text{ then } \\
\hspace{5cm} C[e', e''] := \text{true}
\]

The main loop of Alur et. al.’s algorithm for detecting races in MSCs:

\[
\text{for } e := 1 \text{ to } n \text{ do } \\
\hspace{1cm} \text{for } e' := e - 1 \text{ downto } 1 \text{ do } \\
\hspace{2cm} \text{if } (\text{not } C[e', e] \text{ and } e' \ll e) \text{ then } \\
\hspace{3cm} C[e', e] := \text{true}
\]

\[
\text{for } e'' := 1 \text{ to } e' - 1 \text{ do } \\
\hspace{2cm} \text{if } C[e'', e'] \text{ then } \\
\hspace{3cm} C[e'', e] := \text{true}
\]
Detecting races in MSCs

**Theorem**

Let \( M \) be an MSC with set \( E \) of events and \( n = |E| \). Checking whether \( M \) has a race can be done in \( \mathcal{O}(n^2) \).

**Proof.**
**Theorem**

Let $M$ be an MSC with set $E$ of events and $n = |E|$. Checking whether $M$ has a race can be done in $O(n^2)$.

**Proof.**

Since $\ll$ is acyclic, we number the events such that the numbering defines a total order that is consistent with visual order $\preceq$. This can be done in $O(n)$ using a standard topological sort.
Detecting races in MSCs

**Theorem**

Let $M$ be an MSC with set $E$ of events and $n = |E|$. Checking whether $M$ has a race can be done in $O(n^2)$.

**Proof.**

Since $\ll$ is acyclic, we number the events such that the numbering defines a total order that is consistent with visual order $\preceq$. This can be done in $O(n)$ using a standard topological sort. Then observe that the innermost loop:

$$
\begin{align*}
\text{for } e'' := 1 \text{ to } e' - 1 \text{ do} \\
\quad \text{if } C[e'', e'] \text{ then } C[e'', e] := \text{true}
\end{align*}
$$

of the triple-nested main loop is executed for $(e, e')$ only if $e'$ is an immediate predecessor of $e$ under $\ll$. 
Detecting races in MSCs

**Theorem**

Let $M$ be an MSC with set $E$ of events and $n = |E|$. Checking whether $M$ has a race can be done in $O(n^2)$.

**Proof.**

Since $\ll$ is acyclic, we number the events such that the numbering defines a total order that is consistent with visual order $\preceq$. This can be done in $O(n)$ using a standard topological sort. Then observe that the innermost loop:

```
for e'' := 1 to e' - 1 do
  if C[e'', e'] then C[e'', e] := true
```

of the triple-nested main loop is executed for $(e, e')$ only if $e'$ is an immediate predecessor of $e$ under $\ll$. As for MSCs, an event can have at most two immediate predecessors, the innermost two loop is executed at most $2 \cdot n$ times in total.
Theorem

Let $M$ be an MSC with set $E$ of events and $n = |E|$. Checking whether $M$ has a race can be done in $\mathcal{O}(n^2)$.

Proof.

Since $\ll$ is acyclic, we number the events such that the numbering defines a total order that is consistent with visual order $\preceq$. This can be done in $\mathcal{O}(n)$ using a standard topological sort. Then observe that the innermost loop:

$$\text{for } e'' := 1 \text{ to } e' - 1 \text{ do}$$
$$\quad \text{if } C[e'', e'] \text{ then } C[e'', e] := \text{true}$$

of the triple-nested main loop is executed for $(e, e')$ only if $e'$ is an immediate predecessor of $e$ under $\ll$. As for MSCs, an event can have at most two immediate predecessors, the innermost two loop is executed at most $2 \cdot n$ times in total. This yields a total worst-case time complexity of $n^2 + 2 \cdot n$. □