Overview

1 Lecture 2: Races

phenomenon in MSCs that complicates their interpretation

formal definition

algorithm

input: MSC

output: MSC has a race?

Theoretical Foundations of the UML

Lecture 2: Races

Joost-Pieter Katoen

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

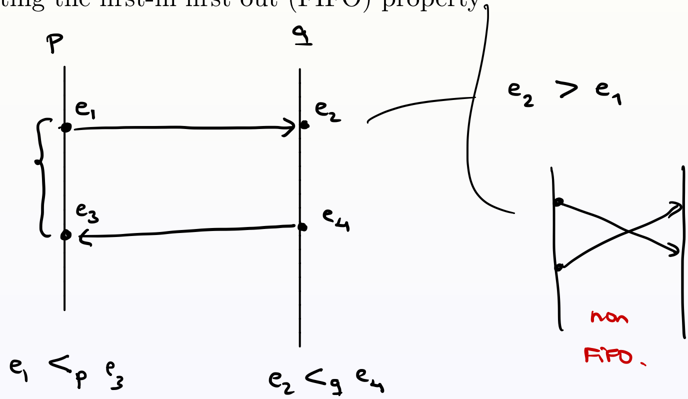
moves.rwth-aachen.de/teaching/ss-20/fuml/

April 21, 2020

Summary of Lecture #1

Summary of Lecture #1

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



Summary of Lecture #1

1. A Message Sequence Chart is a partial order
   - between send and receive events
   - totally ordered per process — vertical ordering
   - receive events happen after their send events — message ordering
   - 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

Summary of Lecture #1

1. A Message Sequence Chart is a partial order
   - between send and receive events
   - totally ordered per process — vertical ordering
   - receive events happen after their send events — message ordering
   - 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

MSC M $\longleftrightarrow$ Lin (M)

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

Summary of Lecture #1

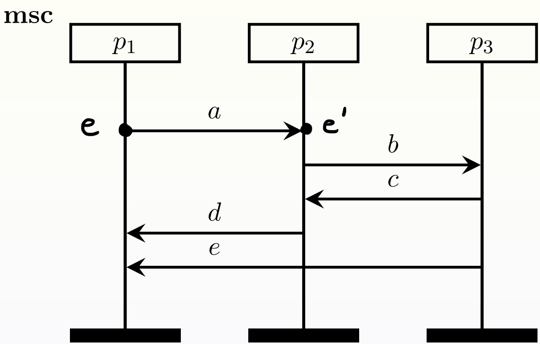
1. A Message Sequence Chart is a partial order
   - between send and receive events
   - totally ordered per process — vertical ordering
   - receive events happen after their send events — message ordering
   - respecting the first-in first out (FIFO) property

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 $\overbrace{\text{linearizations}}^{\text{Lin}(M)}$

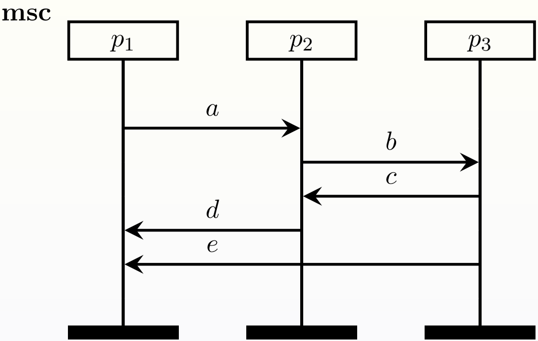
Example



$\ell(e) = \,!(p_1, p_2, a)$

$\ell(e') = \,?(p_2, p_1, a)$

Example



These pictures are formalized using partial orders.

Message Sequence Chart (MSC) (1)

Definition

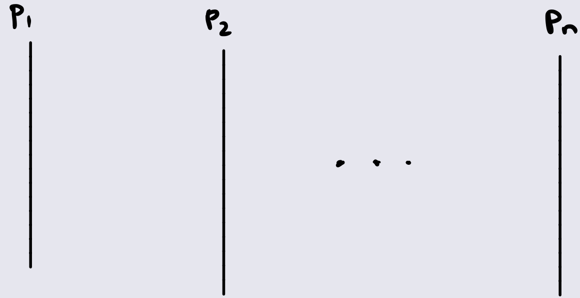
An MSC $M = (\mathcal{P}, E, \mathcal{C}, l, m, \preceq)$ with:

Message Sequence Chart (MSC) (1)

Definition

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

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



Message Sequence Chart (MSC) (1)

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

$$E = \underbrace{\biguplus_{p \in \mathcal{P}} E_p}_{\text{vertically}} = \underbrace{E_? \cup E_!}_{\text{horizontally}}$$

Message Sequence Chart (MSC) (1)

Definition

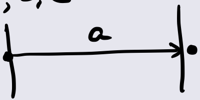
An MSC $M = (\mathcal{P}, E, \mathcal{C}, l, m, \preceq)$ with:

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

$$E = \biguplus_{p \in \mathcal{P}} E_p = E_? \mathbin{\dot\cup} E_!$$

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

a, b, c



Message Sequence Chart (MSC) (1)

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

$$E = \biguplus_{p \in \mathcal{P}} E_p = E_? \,\dot{\cup}\, E_!$$

- $\mathcal{C}$, a finite set of message contents
- $l : E \to Act$, a labelling function defined by:

$$l(e) = \begin{cases} !(p,q,a) & \text{if} \quad e \in E_p \cap E_! \\ ?(p,q,a) & \text{if} \quad e \in E_p \cap E_? \end{cases}, \text{for } p \neq q \in \mathcal{P},\ a \in \mathcal{C}$$

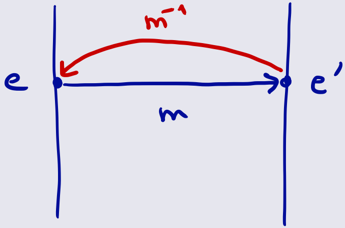
Message Sequence Chart (MSC) (2)

Message Sequence Chart (MSC) (2)

Definition

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

$$m(e) = e' \wedge l(e) = !(p,q,a) \text{ implies } l(e') = ?(q,p,a) \quad (p \neq q,\ a \in \mathcal{C})$$



$m(e) = e'$

Message Sequence Chart (MSC) (2)

Definition

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

$$m(e) = e' \wedge l(e) = !(p,q,a) \text{ implies } l(e') = ?(q,p,a) \quad (p \neq q,\ a \in \mathcal{C})$$

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

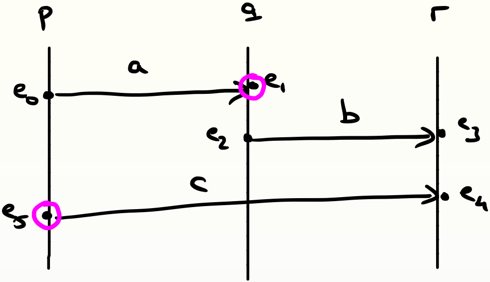
$$\preceq = \Big(\overbrace{\underbrace{\bigcup_{p \in \mathcal{P}} <_p}_{}}^{\text{vertical}} \cup \overbrace{\underbrace{\{(e, m(e)) \mid e \in E_!\}}_{}}^{\text{horizontal}} \Big)^*$$

$<_p$ is a total order = "top-to-bottom" order on process p

communication order $<_c$

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

Example



$m(e_2) = e_3$

$m(e_5) = e_4$

$<_p$: $e_0 <_p e_5$

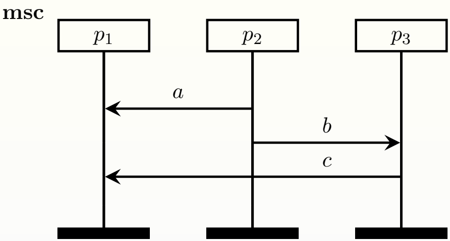
$<_q$: $e_1 <_q e_2$

$<_r$: $e_3 <_r e_4$

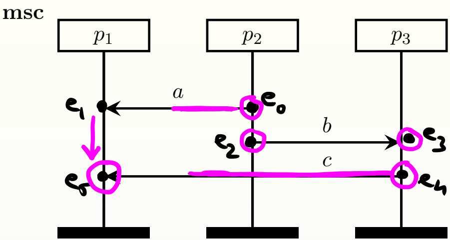
Hasse diagram

$$
\begin{tikzcd}
e_0 \arrow[r] \arrow[dd] & e_1 \arrow[d] & \\
 & e_2 \arrow[r] & e_3 \arrow[d] \\
e_5 \arrow[rr] & & e_4
\end{tikzcd}
$$

Visual order can be misleading



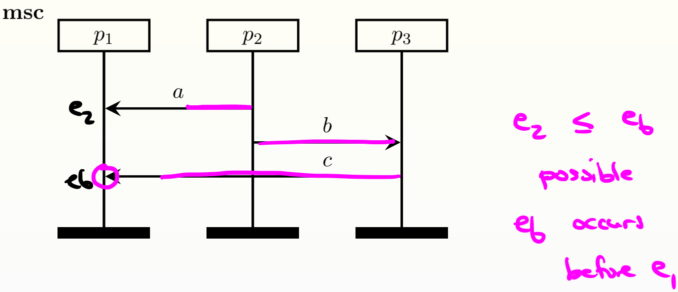
Visual order can be misleading



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

!(p_2, p_1, a)
!(p_2, p_3, b)
?(p_3, p_2, b)
!(p_3, p_1, c)
!(p_1, p_3, c)
!(p_1, p_2, 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.

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.

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 — input: MSC M

output: M has a race or not.

Causal order

defined in a differet way than

$\preceq$: visual order $\longrightarrow$ part of the MSC definition.

Causal order

Main principles:

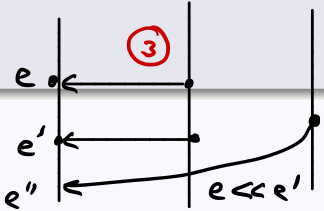
1 - Send events should happen before their matching receive events — similar as for $\preceq$
2 - The ordering of events wrt. sends on same process is unaffected
3 - Receive events on a process sent from the same process are ordered as their sends

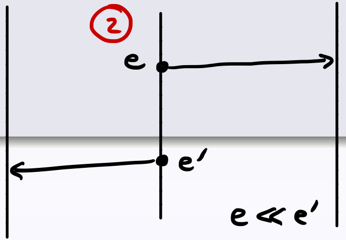
visual order

Definition

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

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





Main principles:

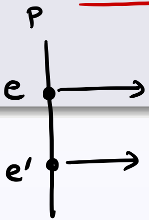
- Send events should happen before their matching receive events
- 2 The ordering of events wrt. sends on same process is unaffected
- Receive events on a process sent from the same process are ordered as their sends

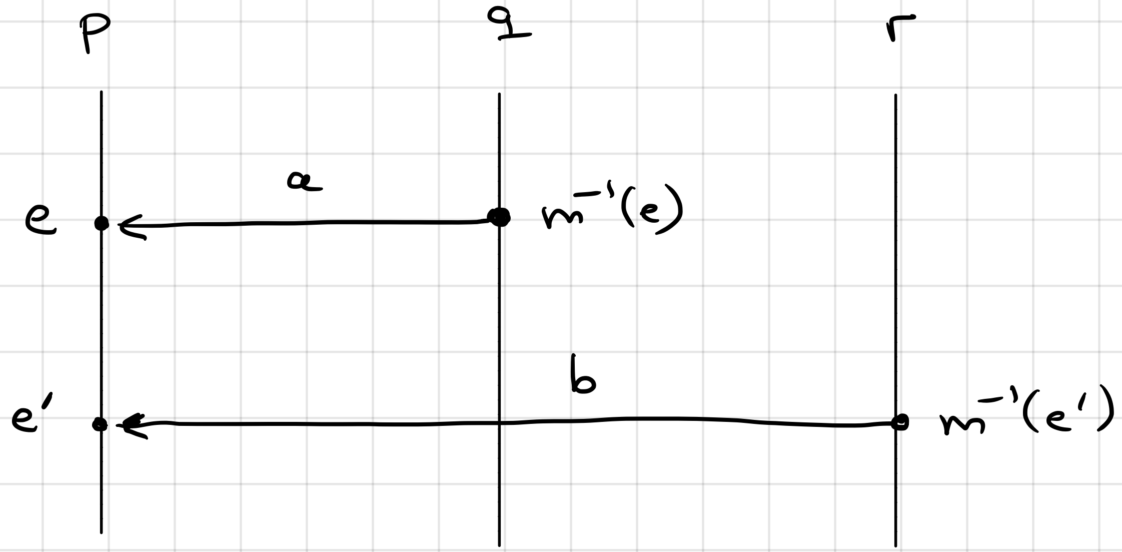
Definition

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

$$\begin{array}{lll} \underline{e \ll e'} & \text{iff} & e' = m(e) \\ & \text{or} & \underline{e <_p e'} \text{ and } \underline{E_!} \cap \{e, e'\} \neq \varnothing \end{array}$$

2





$e \not< e'$ because there is no process u such that

$$m^{-1}(e) <_u m^{-1}(e')$$

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

Causal order

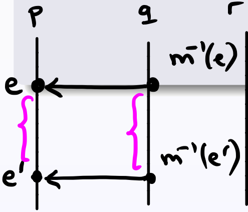
Main principles:

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

Definition

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

$$
\begin{array}{lll}
e \ll e' & \text{iff} & e' = m(e) \\
 & \text{or} & e <_p e' \text{ and } E_! \cap \{e, e'\} \neq \varnothing \\
 & \text{or} & e, e' \in \underbrace{E_p \cap E_?}_{\text{both at } p} \text{ and } \underbrace{m^{-1}(e) <_q m^{-1}(e')}_{\text{both at } q}
\end{array}
$$



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

either (or both) e and e' are sends

Definition

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

$$
\begin{array}{lll}
e \ll e' & \text{iff} & e' = m(e) \\
& \text{or} & e <_p e' \text{ and } E_! \cap \{e, e'\} \neq \varnothing \\
& \text{or} & e, e' \in E_p \cap E_? \text{ and } m^{-1}(e) <_q m^{-1}(e')
\end{array}
$$

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

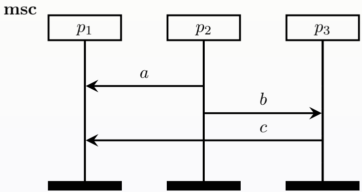
$$
\begin{array}{lll}
e \ll e' & \text{iff} & e' = m(e) \\
 & \text{or} & e <_p e' \text{ and } E_! \cap \{e, e'\} \neq \varnothing \\
 & \text{or} & e, e' \in E_p \cap E_? \text{ and } m^{-1}(e) <_q m^{-1}(e')
\end{array}
$$

Causal order: example

Definition

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

$$\begin{array}{lll} e \ll e' & \text{iff} & e' = m(e) \\ & \text{or} & e <_p e' \text{ and } E_! \cap \{e, e'\} \neq \varnothing \\ & \text{or} & e, e' \in E_p \cap E_? \text{ and } m^{-1}(e) <_q m^{-1}(e') \end{array}$$

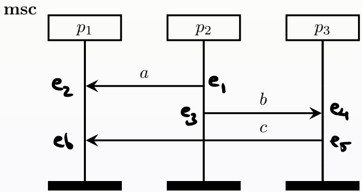


Causal order: example

Definition

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

$$
\begin{array}{lll}
e \ll e' & \text{iff} & e' = m(e) \\
 & \text{or} & e <_p e' \text{ and } E_! \cap \{e, e'\} \neq \varnothing \\
 & \text{or} & e, e' \in E_p \cap E_? \text{ and } m^{-1}(e) <_q m^{-1}(e')
\end{array}
$$



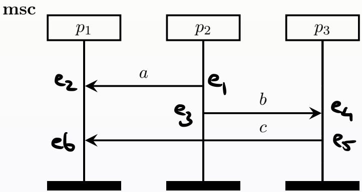
Example

Causal order: example

Definition

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

$$
\begin{array}{lll}
\underline{e \ll e'} & \text{iff} & e' = m(e) \qquad 1 \\
 & \text{or} & e <_p e' \text{ and } E_! \cap \{e, e'\} \neq \varnothing \\
 & \text{or} & e, e' \in E_p \cap E_? \text{ and } m^{-1}(e) <_q m^{-1}(e')
\end{array}
$$



$m(e_1) = e_2$

$\rightarrow \quad e_1 \ll e_2$

Example

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

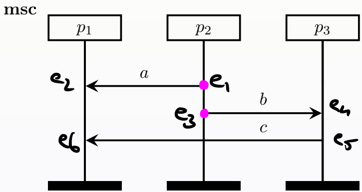
Causal order: example

Definition

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

$$
\begin{array}{lll}
e \ll e' & \text{iff} & e' = m(e) \\
 & \text{or} & e <_p e' \text{ and } E_! \cap \{e, e'\} \neq \varnothing \\
 & \text{or} & e, e' \in E_p \cap E_? \text{ and } m^{-1}(e) <_q m^{-1}(e')
\end{array}
$$

2



$e_4 <_{p_3} e_5$

$\{e_4, e_5\} \cap E_! \neq \varnothing$

$\xrightarrow{2}$ $e_4 \ll e_5$

Example

$e_1 \ll e_2, \quad e_3 \ll e_4, \quad e_5 \ll e_6, \quad \underbrace{e_1 \ll e_3}_{2}, \quad \underbrace{e_4 \ll e_5}_{2}$

$\cancel{e_5,}$

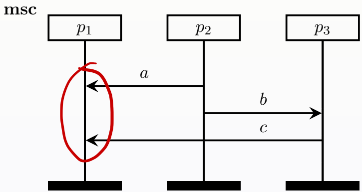
Causal order: example

Definition

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

$$
\begin{array}{lll}
e \ll e' & \text{iff} & e' = m(e) \\
 & \text{or} & e <_p e' \text{ and } E_! \cap \{e, e'\} \neq \varnothing \\
 & \text{or} & e, e' \in E_p \cap E_? \text{ and } m^{-1}(e) <_q m^{-1}(e')
\end{array}
$$

3



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:

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

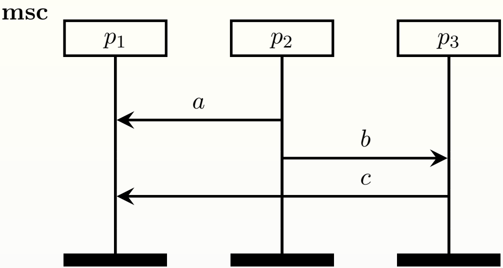
where $\ll^* \subseteq E \times E$ is the reflexive and transitive closure of $\ll$.

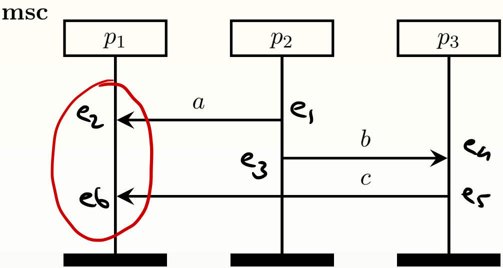
As relation $\ll^*$ contains at most all orderings in $\preceq$,
the MSC M has a race whenever $\preceq \not\subseteq \ll^*$.

visual order

causal order

Race: example





<<

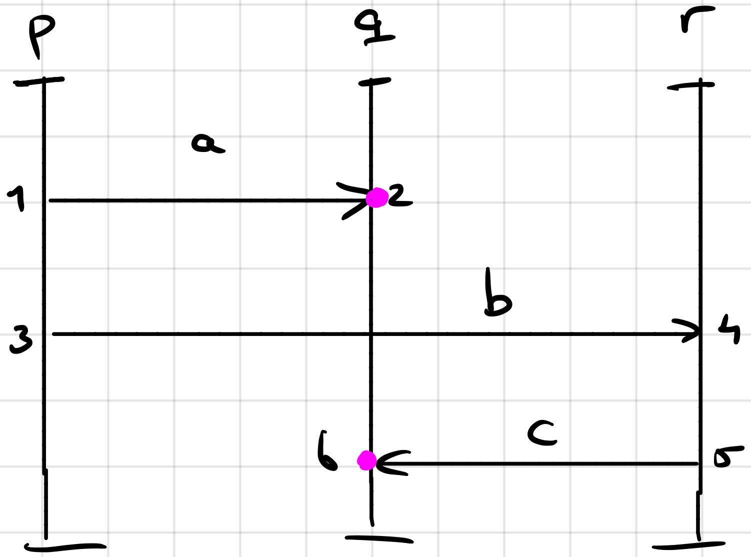
Visual order versus causal order

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

$\ll$:

$1 \ll 3$

$4 \ll 5$

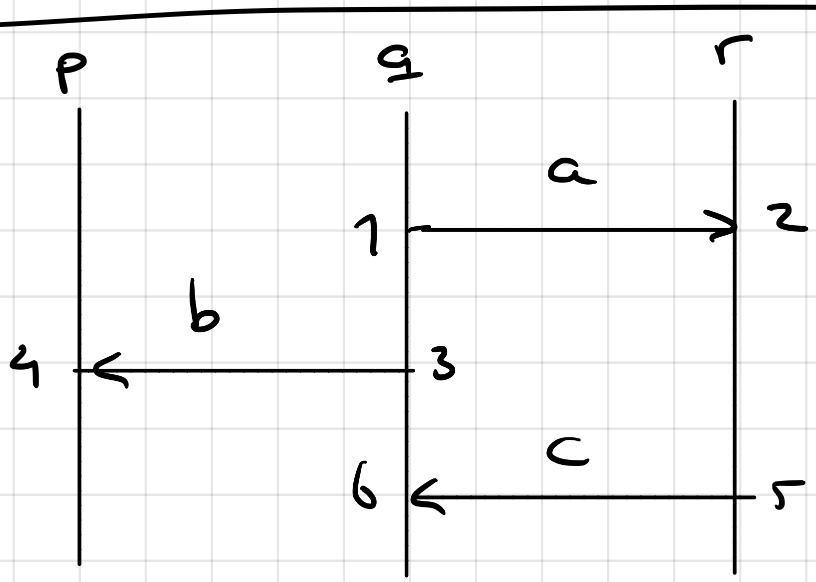
$1 \ll 2$

$3 \ll 4$

$5 \ll 6$

$\preceq$: $\ll$ + $\underbrace{2 \preceq 6}$ because $2 <_p 6$

not $2 \ll 6$!



MSC has no race.

$\ll$: $1 \ll 2$, $3 \ll 4$, $5 \ll 6$, $2 \ll 5$,

$1 \ll 3$, $3 \ll 6$

$=$ $\preceq$ visual order

Why are races problematic?

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

$$\exists e, e' \in E_? \;.\; (e <_p e' \text{ and } e \not\ll^* e')$$

"vertical order"

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

Checking whether an MSC has a race

- MSC M has a race if $\preceq \not\subseteq \ll^*$
- How to check whether MSC M has a race?

  $\boxed{\textit{compute } \ll^*}$ *and check whether* $\preceq \subseteq \ll^*$

no

- transitive closure $\ll^*$ is computed using Floyd-Warshall's algorithm

relation

  - 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 $\mathcal{O}(|E|^3)$
  - by using some specifics of MSC, this is reduced to $\mathcal{O}(|E|^2)$

set of events in MSC M

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

[1] for digraphs without negative cycles.

Algorithm

$\underbrace{\text{compute } \ll^*}_{\text{Warshall's algorithm}}$ and compare with $\preceq$

$X = E$ for MSCs

$\boxed{\text{Warshall's algorithm: } \underline{\text{input}}: R \subseteq X \times X \text{ where } X \text{ is a set} \quad \underline{\text{output}}: R^*}$

Computing $\ll^*$: Warshall's algorithm

Algorithm

$\underbrace{\text{compute } \ll^*}_{\text{Warshall's algorithm}}$ and compare with $\preceq$

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

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

(our setting: $X = E, R = \ll,\ R^* = \ll^*$)

Warshall's algorithm: preliminaries

Warshall's algorithm: preliminaries

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

R

$X = \{x_1, \ldots, x_4\}$

$R = \{(x_1, x_2), (x_3, x_4), (x_2, x_4)\}$

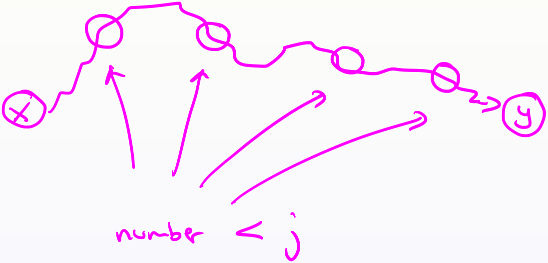
grph (R) :

$$\begin{tikzcd} x_1 \arrow[r] & x_2 \arrow[rr, bend left] & x_3 \arrow[r] & x_4 \end{tikzcd}$$

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 $\overset{j}{\Longrightarrow}$ as follows:

  $\boxed{x \overset{j}{\Longrightarrow} y}$ 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



$x \overset{j}{\Longrightarrow} y$

Warshall's algorithm: preliminaries

- assume: graph vertices are numbered $\underbrace{\{1, 2, \ldots, n\}}$ where $n = |E|$
- for $j \in \{1, \ldots, n+1\}$ define relation $\stackrel{j}{\Longrightarrow}$ as follows:
  $x \stackrel{j}{\Longrightarrow} y$ 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:

$$
\begin{array}{llll}
(1) & \boxed{x \Longrightarrow y} & \text{iff} & \underline{x \stackrel{n+1}{\Longrightarrow} y} \\
(2) & x \stackrel{1}{\Longrightarrow} y & \text{iff} & x = y \text{ or } x \ll y \\
(3) & x \stackrel{y+1}{\Longrightarrow} z & \text{iff} & \underline{x \stackrel{y}{\Longrightarrow} z} \text{ or } x \stackrel{y}{\Longrightarrow} y \stackrel{y}{\Longrightarrow} z
\end{array}
$$

$x \xrightarrow{<n+1} y$

initialisation

$\stackrel{j}{\Longrightarrow}$ by induction on j — start $j=1$ $x \stackrel{1}{\Longrightarrow} y$

Warshall's algorithm: preliminaries

$(x, y) \in R^*$

- assume: graph vertices are numbered $\{1, 2, \ldots, n\}$ where $n = |E|$
- for $j \in \{1, \ldots, n{+}1\}$ define relation $\overset{j}{\Longrightarrow}$ as follows:

  $x \overset{j}{\Longrightarrow} y$ 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) $x \Longrightarrow y$ iff $x \overset{n+1}{\Longrightarrow} y$ termination condition

$$\boxed{\begin{array}{lll} (2) & x \overset{1}{\Longrightarrow} y \quad \text{iff} & x = y \text{ or } x \ll y \\ (3) & x \overset{y+1}{\Longrightarrow} z \quad \text{iff} & x \overset{y}{\Longrightarrow} z \text{ or } x \overset{y}{\Longrightarrow} y \overset{y}{\Longrightarrow} z \end{array}}$$

- Algorithm: determine the relations $\overset{1}{\Longrightarrow}, \ldots, \overset{n}{\Longrightarrow}, \overset{n+1}{\Longrightarrow}$ 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 $\overset{j}{\Longrightarrow}$ as follows:
  $x \overset{j}{\Longrightarrow} y$ 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) $x \Longrightarrow y$ iff $x \overset{n+1}{\Longrightarrow} y$
  (2) $x \overset{1}{\Longrightarrow} y$ iff $x = y$ or $x \ll y$
  (3) $x \overset{y+1}{\Longrightarrow} z$ iff $x \overset{y}{\Longrightarrow} z$ or $x \overset{y}{\Longrightarrow} y \overset{y}{\Longrightarrow} z$
- Algorithm: determine the relations $\overset{1}{\Longrightarrow}, \ldots, \overset{n}{\Longrightarrow}, \overset{n+1}{\Longrightarrow}$ iteratively using properties (2) + (3); Result is then given by (1).
- Store $\overset{i}{\Longrightarrow}$ in a boolean matrix $\underline{C}$ of cardinality $\underline{|E| \times |E|}$

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 $\overset{j}{\Longrightarrow}$ as follows:
  $x \overset{j}{\Longrightarrow} y$ 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) $x \Longrightarrow y$ iff $x \overset{n+1}{\Longrightarrow} y$ termination

  $$\boxed{\begin{array}{llll} (2) & x \overset{1}{\Longrightarrow} y & \text{iff} & x = y \text{ or } x \ll y \\ (3) & x \overset{y+1}{\Longrightarrow} z & \text{iff} & x \overset{y}{\Longrightarrow} z \text{ or } x \overset{y}{\Longrightarrow} y \overset{y}{\Longrightarrow} z \end{array}}$$

  initialisation

  loop
- Algorithm: determine the relations $\overset{1}{\Longrightarrow}, \ldots, \overset{n}{\Longrightarrow}, \overset{n+1}{\Longrightarrow}$ iteratively using properties (2) + (3); Result is then given by (1).
- Store $\overset{i}{\Longrightarrow}$ in a boolean matrix C of cardinality $|E| \times |E|$
- Postcondition: $C[x, y] = \texttt{true}$ iff $(x, y) \in R^*$
- Precondition: $\forall x, y \in X \;.\; C[x, y] = \texttt{false}$

Warshall's algorithm

/* first compute $x \xRightarrow{1} y$ */

for $x := 1$ to n do

for $y := 1$ to n do

$C[x, y] := (x = y \text{ or } \underbrace{\overbrace{(x, y) \in R}}_{x \ll y})$

initialisation

2

/* loop invariant: after the j-th iteration of */

/* outermost loop it holds: $C[x, y] =$ `true` iff $x \xRightarrow{j+1} y$ */

1. for $y := 1$ to n do $\xRightarrow{2} \ldots \xRightarrow{n+1}$

2. for $x := 1$ to n do

if $C[x, y]$ then $x \xRightarrow{y} y$

3. for $z := 1$ to n do

if $C[y, z]$ then $y \xRightarrow{y} z$

$\underbrace{C[x, z] := \texttt{true}}_{x \xRightarrow{y+1} z}$

loop 3

Correctness and complexity

Lemma: correctness

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

Proof.

if: trivial; *only if*: by induction on j. $\square$

<u>Claim</u>: after j iterations (for any $0 \leq j \leq n$):

$$k \overset{j+1}{\Longrightarrow} m \quad \text{implies} \quad C[k,m] = 1$$

<u>Proof</u>: by induction on j.

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

2) ind. step: let $j > 0$ and assume $k \overset{j+1}{\Longrightarrow} m$.

   a) if $C[k,m] = 1$, done ✓ $\qquad k \overset{j}{\Longrightarrow} m$

   b) assume $C[k,m] = 0$. Then by ind. hyp., it follows $k \overset{j}{\not\Longrightarrow} m$. But since $k \overset{j+1}{\Longrightarrow} m$ iff $k \overset{j}{\Longrightarrow} m$ or $k \overset{j}{\Longrightarrow} j \overset{j}{\Longrightarrow} m$ (by (3)) it follows $k \overset{j}{\Longrightarrow} j \overset{j}{\Longrightarrow} m$.

   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 $\boxtimes$

Correctness and complexity

Lemma: correctness

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

Proof.

if: trivial; *only if*: by induction on j. □

Complexity

Worst-case time complexity of Warshall's algorithm : $\mathcal{O}(n^3)$ with $n = |X|$

Proof.

follows from the fact that there is a triple-nested loop of which each loop has at most n iterations. □

Efficiency improvement [Alur et al. '96]

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

arbitrary

Efficiency improvement [Alur et al. '96]

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

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

Efficiency improvement [Alur et al. '96]

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 $\ll$, $\underbrace{\text{the complexity}}_{\mathcal{O}(n^3)}$ can be improved.

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 $\ll$:

Efficiency improvement [Alur et al. '96]

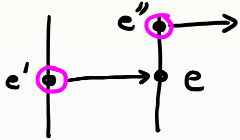
Warshall's algorithm computes R^* for every binary relation $R \subseteq X \times X$.

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 (why?)



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

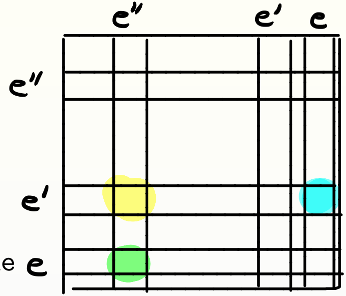
$$e \ll e' \text{ and } \neg(\exists e'' \notin \{e, e'\}.\ e \ll e'' \wedge e'' \ll e')$$

Efficiency improvement [Alur et al. '96]

The main loop of Warshall's algorithm:

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

C :



$C[e', e'']$

Efficiency improvement [Alur et al. '96]

The main loop of Warshall's algorithm:

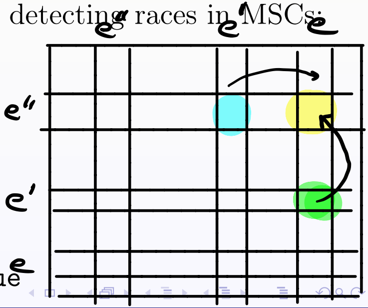
$\mathcal{O}(n^3)$

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

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

$\mathcal{O}(n^2)$

for $e := 1$ to n do
  for $e' := e - 1$ downto 1 do
    if (not $C[e', e]$ and $e' \ll e$) then
      $C[e', e] :=$ `true`
      for $e'' := 1$ to $e' - 1$ do
        if $C[e'', e']$ then
          $C[e'', e] :=$ `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.

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.

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.

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.

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:

$$\begin{array}{l} \text{for } e'' := 1 \text{ to } e' - 1 \text{ do} \\ \qquad \text{if } C[e'', e'] \text{ then } C[e'', e] := \texttt{true} \end{array}$$

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 $\mathcal{O}(n^2)$.

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:

$$\begin{array}{l} \text{for } e'' := 1 \text{ to } e' - 1 \text{ do} \\ \qquad \text{if } C[e'', e'] \text{ then } C[e'', e] := \texttt{true} \end{array}$$

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.

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.

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:

$$\begin{array}{l} \text{for } e'' := 1 \text{ to } e' - 1 \text{ do} \\ \qquad \text{if } C[e'', e'] \text{ then } C[e'', e] := \texttt{true} \end{array}$$

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 $n^2 + 2 \cdot n$. □