Agenda

- Clocks, logical clocks, state vectors
- Optimistic replication
  - CVS, Subversion
  - Duplicated databases
  - Collaborative editing requirements
  - Operational transformation: properties for convergence, transformation functions
Clock Synchronisation

- Time is unambiguous in a centralised system
- There is no global agreement on time in a distributed system
- Example
  - Program consisting of 100 files
  - Use of `make` to recompile only changed source files
  - If input.c has time 2151 and input.o has time 2150, then recompilation needed
Clock Synchronization

- make does not call the compiler
Logical clock

- Sufficient that all machines agree on the same time (not necessarily real time)
- Lamport 1978 – rather than agreeing on what time it is, sufficient to agree on the order in which events occur
- Previous example: if input.c is older or newer than input.o
Lamport timestamps

- Happens-before relation
- $a \rightarrow b$ ($a$ happens before $b$)

Two situations:
- If $a$ and $b$ are events in the same process and $a$ occurs before $b$, then $a \rightarrow b$
- If $a$ is the event of a message being sent by one process and $b$ is the event of the message being received by another process, then $a \rightarrow b$. A message cannot be received before or at the same time it is sent

- If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$
- If neither $a \rightarrow b$ nor $b \rightarrow a$ then $a$ is concurrent with $b$
Lamport timestamps

- For every event $a$ assign $C(a)$ on which all processes agree
- If $a \rightarrow b$ then $C(a) < C(b)$
- Clock time must always increase
- Lamport solution
  - Each message carries the sending time
  - If receiver clock $<$ time of the arrived message, then receiver forwards its clock to $1 +$ sending time
Lamport timestamps

(a)  

(b)
Lamport timestamps

• If $a$ happens before $b$ in the same process then $C(a)<C(b)$

• If $a$ and $b$ represent the sending and receiving of a message, $C(a)<C(b)$

• For all distinctive events $a$ and $b$, $C(a)\neq C(b)$
  ▪ Attach the number of the process to the lower order of the time
  ▪ If $a$ generated by process 1 at time 40 and $b$ generated by process 2 at time 40, then $C(a)=40.1$ and $C(a)=40.2$
Vector timestamps

- Lamport timestamps limits
  - if $C(a)<C(b)$ does not imply that $a \rightarrow b$
  - $a \parallel b$ does not imply $C(a)=C(b)$

- Example: posting articles and reactions to posted articles

- Lamport timestamps do not capture causality

- Vector timestamps capture causality
  - If $VT(a)<VT(b)$, then $a$ causally precedes $b$
  - Each process $P_i$ maintains $V_i$
    - $V_i[i] =$ the no. of events that occurred so far at $P_i$
    - If $V_i[j]=k$ then $P_i$ knows that $k$ events occurred at $P_j$
Vector timestamps

• Comparison of two vectors
  - $V=W$ iff $\forall i \ V[i]=W[i]$  
  - $V<W$ iff $\forall i \ V[i]\leq W[i]$ and $\exists i \ V[i]<W[i]$  
  - $[1,2,0] < [3,2,1]$  
  - $[0,1,1] \not< [1,0,1]$
Vector timestamps – computation rules

• Process $P_i$
  - Initialisation: $\forall k \ V_i[k]=0$
  - Local event: $V_i[i]= V_i[i]+1$
  - Sending message $m: V_i[i]= V_i[i]+1$, then send $(m,V_i)$
  - Receiving message $(m,V_j)$:
    - $\forall k \ V_i[k]=\max(V_i[k], V_j[k])$
    - $V_i[i]=V_i[i]+1$
Vector timestamps – example

P₁

a  [1,0,0]
b  [2,0,0]
c  [3,0,0]

P₂

d  [0,1,0]
e  [2,2,0]
f  [2,3,0]

P₃

g  [0,0,1]
h  [0,0,2]
i  [2,3,3]
State vector

![State vector diagram](image_url)
State vector based timestamping scheme

- State vector $SV^k$ at site $k$
  - Initially $SV^k[i]=0$, $\forall i \in \{0,\ldots,N-1\}$
  - Updating rule 1: after executing a local operation, $SV^k[k]=SV^k[k]+1$
  - After executing a local operation and updating $SV^k$, the local operation is timestamped with $SV^k$ and broadcast to all remote sites
  - Updating rule 2: after executing a remote operation $O$ with $SV_O$, $SV^k[i]=\max(SV^k[i], SV_O[i])$, $\forall i \in \{0,\ldots,N-1\}$
State vector: causality preservation

- \(O_i\) generated at site \(i\) and timestamped by \(SV_{O_i}\)
- \(O_i\) not allowed to be executed at site \(k\) (\(k \neq i\)) until:
  - \(SV_{O_i}[i] = SV^k[i]+1\)
  - \(SV_{O_i}[j] \leq SV^k[j], \quad \forall j \in \{0,\ldots,N-1\}, \ j \neq i\)
Version control systems: CVS, Subversion
Lock-modify-unlock solution
Copy-modify-merge solution
Copy-modify-merge solution

Harry compares the latest version to his own

Harry reads A''

Sally reads A''

A new merged version is created

Repository

A''

Harry

Sally

The merged version is published

Repository

A''

Harry

Sally

Now both users have each others’ changes

Repository

A''

Harry

Sally

Repository

A''

Harry

Sally
Duplicated databases (Thomas Write Rule 1975) (*)

- **Model**
  - A set of independent DBMPs
  - Each DBMP has its own copy of the database
  - DBMPs communicate via messages
  - Communications are subject to failures
  - Messages between two sites are delivered in the same order they were sent
  - No use of global timestamps

- The system is correct if it eventually converges

Duplicated databases (Thomas Write Rule 1975)

DBMP\(_1\)  \quad \text{DBMP}\(_2\)

\begin{align*}
\text{op}_1 \\
\text{op}_2
\end{align*}

\text{Not possible}

DBMP\(_1\)  \quad \text{DBMP}\(_2\)  \quad \text{DBMP}\(_3\)

\begin{align*}
\text{op}_1 \\
\text{op}_2
\end{align*}

\text{Possible}
Duplicated databases (Thomas Write Rule 1975)

- The database = collection of (selector,value) pairs

- Operations:
  - Selection:
    - get(selector) returns the current associated value
  - Assignment:
    - set(selector, new_value) replaces associated value with new_value
  - Creation:
    - new(selector, initial_value) adds (selector, initial_value) entry
  - Deletion:
    - delete(selector, value) deletes existing (selector, value) pair
Duplicated databases (Thomas Write Rule 1975)

- How to guarantee that copies are consistent?

```
new(x,5)
set(x,8)
(x,9)
set(x,9)
(x,8)
```

```
DBMP_1  DBMP_2
```

Database
Thomas Timestamps

- In the face of concurrent modifications to an entry, how to select the « most recent » change?
- Thomas timestamps before Lamport timestamps!
- A timestamp is a pair \((T,D)\)
  - \(T\) is a network time standard (time-of-day)
  - \(D\) is a DBMP identifier
- Timestamps comparison
  - \((T_1,D_1) > (T_2,D_2)\) iff \((T_1 > T_2)\) or \((T_1 = T_2 \text{ and } D_1 > D_2)\)
- If \(D_1 = D_2\) and \(T_1 = T_2\), then the same operation
Database entry

- $E ::= (S, V, T)$
  - $S$ is the selector
  - $V$ is the value
  - $T$ is the timestamp = (Time, DBMP id) of the last change to the entry
Thomas write rule = last writer wins

**DBMP**

1. new(x, 5)
2. set(x, 8)
3. set(x, 9)

Database

1. (x, 5, (10h, 1))
2. (x, 8, (10h02, 1))
3. (x, 9, (10h03, 2))

((10h02, 1) < (10h03, 2))

**DBMP**

1. (x, 5, (10h, 1))
2. set(x, 9)
3. (x, 9, (10h03, 2))

((10h03, 2) > (10h02, 1))
Creation/update

- Assume the creation will arrive and create the entry right away
- Creation operation ignored at arrival
Creation/update

DBMP_1

new(x,2)

DBMP_2

set(x,3)

(x,2, ...)

DBMP_3

(x,3, ...)

ignored
Deletion

- Solution: never remove an entry, mark « deleted » flag
Tombstones

- $E ::= (S,V,F,T)$
  - $S$ is the selector
  - $V$ is the value
  - $F$ is the deleted/not-deleted flag
  - $T$ is the timestamp $= (\text{Time, DBMP id})$ of the last change to the entry

- $F=t$ if deleted
- $F=f$ if not-deleted
Tombstones

DBMP₁
(x, 3, f, (10h, 2))
delete(x)
(x, 3, t, (10h₀₁, 1))

DBMP₂
(x, 3, f, (10h, 2))
set(x, 5)
(x, 5, f, (10h₀₂, 2))

DBMP₃
(x, 3, f, (10h, 2))
(x, 5, f, (10h₀₂, 2))

DBMP₄
(x, 3, f, (10h, 2))
(x, 3, t, (10h₀₁, 1))

(x, 3, t, (10h₀₁, 1))
Tombstones prevent recreation
Tombstones

- DBMP1 cannot distinguish in which of the two cases DBMP2 is
  - DBMP1: set(x,5)  
  - DBMP2: set(x,5)

  - Recreation!
  - Divergence!

- Solution: Associate to an entry the creation timestamp
Tombstones

- $E ::= (S, V, F, CT, T)$
  - $S$ is the selector
  - $V$ is the value
  - $F$ is the deleted/not-deleted flag
  - $CT$ is the timestamp for creation
  - $T$ is the timestamp $= (\text{Time}, \text{DBMP id})$ of the last change to the entry

- If $F = f$ and $CT = T$, then creation
- If $F = f$ and $CT < T$, then assignment
- If $F = t$, then deletion
Tombstones

DBMP\textsubscript{1}

\begin{itemize}
\item \texttt{delete(x)}
\item \texttt{set(x,5)}
\item \texttt{same creation time => delete}
\end{itemize}

DBMP\textsubscript{2}

\begin{itemize}
\item \texttt{set(x,5)}
\item \texttt{same creation time => delete}
\end{itemize}
Tombstones

DBMP\(_1\)

\((x,3,f,(10h,2),(10h,2))\)

delete\((x)\)

\((x,3,t,(10h,2),(10h01,1))\)

\((x,5,f,(10h03,2),(10h03,2))\)

DBMP\(_2\)

\((x,3,f,(10h,2),(10h,2))\)

set\((x,5)\)

\((x,3,t,(10h,2),(10h01,1))\)

\((x,5,f,(10h03,2),(10h03,2))\)

Different creation time =>
recreate
Garbage collection

- Make sure of no reception of assignments with same S and the same or older CT
- Remember assumption: Modifications of a DBMP delivered in sequential order
- Each DBMP maintains two « timestamp vectors »
  - Last modifications from all DBMPs
    - LM[i] last timestamp from DBMP i
    - Modified each time an operation is received
  - Oldest timestamps received by each DBMP
    - OT[i] oldest timestamp received by DBMP i
    - Sent upon reception of a delete
- Can do garbage collection if timestamp of delete <= timestamp of min(OT)
Garbage collection

DBMP_1

LM=[[]
OT=[]

new(x)

(x,1,f,(1h,1),(1h,1))

LM=[[2h,2]]
OT=[]

LM=[(2h,2),(3h,3)]
OT=[]

DBMP_2

LM=[[]
OT=[]

LM=[(1h,1)]
OT=[]

LM=[(1h,1),(2h,2)]
OT=[]

LM=[(2h,2)]
OT=[]

new(y)

(y,2,f,(2h,2),(2h,2))

LM=[(1h,1),(3h,3)]
OT=[]

DBMP_3

LM=[[]
OT=[]

LM=[(1h,1)]
OT=[]

LM=[(1h,1),(2h,2)]
OT=[]

LM=[(1h,1)]
OT=[]

new(z)

(z,3,f,(3h,3),(3h,3))
Garbage collection

DBMP₁
LM=[(2h,2),(3h,3)]
OT=[]
delete(z)
LM=([z,3,t,(3h,3),(4h,1)]
LM=[(2h,2),(3h,3)]
OT=([(3h,2)]
LM=[(2h,2),(3h,3)]
OT=([(3h,2),(2h,3)]

DBMP₂
LM=[(1h,1),(3h,3)]
OT=[]
LM=[(4h,1),(3h,3)]
OT=[]
LM=[(4h,1),(2h,2)]
OT=([(3h,2)]

DBMP₃
LM=[(1h,1),(2h,2)]
OT=[]
LM=[(4h,1),(2h,2)]
OT=([(3h,2)]
LM=[(4h,1),(2h,2)]
OT=([(3h,2)]
LM=[(4h,1),(2h,2)]
OT=([(3h,2)]
Garbage collection

DBMP_1
LM=[[2h,2),(3h,3)]
OT=[[3h,2),(2h,3)]

LM=[[5h,2),(3h,3)]
OT=[[3h,2),(2h,3)]

LM=[[5h,2),(3h,3)]
OT=[[3h,2),(4h,3)]

DBMP_2
LM=[[4h,1),(3h,3)]
OT=[[2h,3]]

delete(y)

LM=[[4h,1),(3h,3)]
OT=[[2h,3]]

LM=[[4h,1),(3h,3)]
OT=[[3h,1),(2h,3)]

LM=[[4h,1),(3h,3)]
OT=[[3h,1),(4h,3)]

DBMP_3
LM=[[4h,1),(2h,2)]
OT=[[3h,2]]

LM=[[4h,1),(5h,2)]
OT=[[3h,2]]

LM=[[4h,1),(5h,2)]
OT=[[3h,1),(3h,2)]

LM=[[4h,1),(5h,2)]
OT=[[3h,1),(3h,2)]

LM=[[4h,1),(5h,2)]
OT=[[3h,1),(4h,3)]
Garbage collection

\[\begin{align*}
\text{DBMP}_1 : \\
\text{LM} &= [(5h,2),(3h,3)] \\
\text{OT} &= [(3h,2),(4h,3)] \\
\end{align*}\]

\[\begin{align*}
\text{DBMP}_2 : \\
\text{LM} &= [(4h,1),(3h,3)] \\
\text{OT} &= [(3h,1),(4h,3)] \\
\end{align*}\]

\[\begin{align*}
\text{DBMP}_3 : \\
\text{LM} &= [(4h,1),(5h,2)] \\
\text{OT} &= [(3h,1),(3h,2)] \\
\end{align*}\]

\[
\text{delete}(x) \\
(x,1,f,(1h,1),(6h,3))
\]

- z can be garbaged
Collaborative editing: from users to community of users

“Isn’t it chaotic to all edit in the same document, even the same paragraph, at the same time?”

“Why would a group ever want to edit in the same line of text at the same time?”

GROVE, 1989
Collaborative editing: from users to community of users

2013: MOOC “Fundamentals of Online Education: Planning and Applications” with 40,000 participants.
2016: Nuit debout, more than 70 people edit a pad
2018: online CSCW PC meeting with 120 members
Collaborative editing: from users to community of users

Real-time Wikipedia

Ponte Morandi: Revision history

View logs for this page (view filter log)

Search for revisions
From year (and earlier): 2018
From month (and earlier): all
Tag filter: Show

For any version listed below, click on its date to view it. For more help, see Help:Page history and Help:Edit summary.

External tools: Revision history statistics • Revision history search • Edits by user • Number of watchers • Page view statistics • Fix dead links

(cur) = difference from current version, (prev) = difference from preceding version, m = minor edit, → = section edit, ← = automatic edit summary

(newest I oldest) View (newer 24 | older 24) (20 | 50 | 100 | 250 | 500)

Compare selected revisions

- (cur) 12:22, 14 August 2018 Pigsowthwing (talk | contribs) m . . (4,619 bytes) (+1) . . (→top: ce) (undo)

- (cur) 12:22, 14 August 2018 Pigsowthwing (talk | contribs) m . . (4,620 bytes) (+4) . . (→References: ce) (undo)

- (cur) 12:22, 14 August 2018 Pigsowthwing (talk | contribs) m . . (4,624 bytes) (+4) . . (→Collapse of the bridge: redirected here) (undo)

- (cur) 12:21, 14 August 2018 Pigsowthwing (talk | contribs) (m . . (4,668 bytes) (+29) . . (→top: fmt) (undo)

- (cur) 12:21, 14 August 2018 Pigsowthwing (talk | contribs) (m . . (4,639 bytes) (+12) . . (→[current]) (undo)

- (cur) 12:20, 14 August 2018 Pigsowthwing (talk | contribs) (m . . (4,627 bytes) (-29) . . (→ce) (undo)

- (cur) 12:20, 14 August 2018 37.159.90.23 (talk) . . (4,656 bytes) (0) . . (→undo) (Tags: Mobile edit, Mobile web edit)

- (cur) 12:19, 14 August 2018 Themandrak (talk | contribs) m . . (4,656 bytes) (+269) . . (Added reference) (undo)

- (cur) 12:19, 14 August 2018 Avaya1 (talk | contribs) . . (4,387 bytes) (+17) . . (undo)

- (cur) 12:19, 14 August 2018 Prioryman (talk | contribs) . . (4,370 bytes) (+46) . . (→top: bold name) (undo)

- (cur) 12:19, 14 August 2018 Gianluigi02 (talk | contribs) . . (4,324 bytes) (+44) . . (→undo) (Tags: Mobile edit, Mobile web edit)

- (cur) 12:18, 14 August 2018 Denimadep (talk | contribs) . . (4,368 bytes) (-2) . . (replaced "closed" with "collapsed" in infobox) (undo)

- (cur) 12:17, 14 August 2018 37.74.150.97 (talk) . . (4,370 bytes) (+167) . . (→cat) (undo)

- (cur) 12:16, 14 August 2018 Pigsowthwing (talk | contribs) . . (4,203 bytes) (-42) . . (→top: redirected uhre) (undo)

- (cur) 12:15, 14 August 2018 Avaya1 (talk | contribs) . . (4,245 bytes) (+4) . . (undo)

- (cur) 12:14, 14 August 2018 Avaya1 (talk | contribs) . . (4,241 bytes) (+75) . . (undo)

- (cur) 12:14, 14 August 2018 Prioryman (talk | contribs) . . (4,166 bytes) (+17) . . (→top: add closed, it'll certainly never be used again now) (undo)

- (cur) 12:13, 14 August 2018 Prioryman (talk | contribs) . . (4,149 bytes) (+1) . . (→top: English spelling) (undo)

- (cur) 12:13, 14 August 2018 Urbourbo (talk | contribs) . . (4,150 bytes) (+44) . . (→Collapse of the bridge) (undo)

- (cur) 12:13, 14 August 2018 Avaya1 (talk | contribs) . . (4,106 bytes) (+327) . . (undo)

- (cur) 12:12, 14 August 2018 Pieceofmetalwork (talk | contribs) m . . (3,779 bytes) (+2) . . (undo)

- (cur) 12:12, 14 August 2018 Prioryman (talk | contribs) m . . (3,777 bytes) (-29) . . (fix) (undo)

- (cur) 12:12, 14 August 2018 Pieceofmetalwork (talk | contribs) . . (3,806 bytes) (+40) . . (undo)

- (cur) 12:12, 14 August 2018 Prioryman (talk | contribs) . . (3,766 bytes) (+354) . . (→Collapse of the bridge: more) (undo)
Limitations of Central Authority Systems
Peer-to-Peer Collaborative Systems
Collaboration Modes

Concurrent Changes
Collaboration Modes

Offline Work

conflicts
Collaboration Modes
Ad-hoc Collaboration
Operational transformation

- Domain of application: collaborative editing
- Document replication
  - Disconnected work
  - Better response time for real-time collaboration
Operational transformation

- Optimistic replication model
  - An operation is:
    - Locally executed,
    - Sent to other sites,
    - Received by a site,
    - Transformed according to concurrent operations,
    - Executed on local copy

- 2 components:
  - An integration algorithm: diffusion, integration
  - Some transformation functions
Operational transformation

- Textual documents seen as a sequence of characters
- Operations
  - $\text{ins}(p,c)$
  - $\text{del}(p)$
- Three main issues
  - Causality preservation
  - Intention preservation
  - Convergence
Causality

Site 1

"x"

"y"

Site 2

"x"

"y"

\[ \text{op}_1 = \text{ins}(1, y) \]

\[ \text{op}_2 = \text{del}(1) \]

Site 3

"x"

Site 2

"x"

\[ \text{op}_1 = \text{ins}(1, y) \]

\[ \text{op}_2 = \text{del}(1) \]

\[ \text{op}_1 = \text{ins}(1, y) \]

\[ \text{op}_2 = \text{del}(1) \]

\[ \text{op}_1 = \text{ins}(1, y) \]

\[ \text{op}_2 = \text{del}(1) \]

\[ \text{op}_1 = \text{ins}(1, y) \]
Causality

Site 1

[0,0,0] "X"

op₁=ins(1,y)

[1,0,0] "YX"

op₂=del(1)

[1,1,0] "X"

Site 2

[0,0,0] "X"

op₁=ins(1,y)

[1,0,0] "YX"

Site 3

[0,0,0] "X"

op₂=del(1)

[1,1,0] "X"

op₁=ins(1,y)

[1,0,0] "Y"

delayed
Intention

- Intention of an operation is the observed effect as result of its execution on its generation state
- Passing from initial state “ab” to final state “aXb” we can observe:
  - ins(2,X)
  - ins(a<X<b)
  - ins(a<X)
  - ins(X<b)
Preserving user intention (*)

- For any operation op, the effects of executing op at all sites should be the same as the intention of op.
- The effect of executing O does not change the effects of independent operations.

Intention violation

Site 1

“concurrency control”

\( op_1 = \text{ins}(7, r) \)

“concurrency control”

Site 2

“concurrency control”

\( op_2 = \text{ins}(17, o) \)

“concurrency control”

\( op_2 = \text{ins}(17, o) \)

“concurrency control”

\( op_1 = \text{ins}(7, r) \)
Intention violation + divergence

Site 1

"concurrency control"

$op_1 = \text{ins}(7, r)$

"concurrency control"

Site 2

"concurrency control"

$op_2 = \text{ins}(17, o)$

"concurrency control"

$op_1 = \text{ins}(7, r)$

Site 1

"concurrency control"

"concurrency control"

Site 2

"concurrency control"

$op_2 = \text{ins}(17, o)$

"concurrency control"
Intention preservation

\[ T(\text{ins}(p_1, c_1), \text{ins}(p_2, c_2)) :\]
\[
\text{if } (p_1 < p_2) \quad \text{return } \text{ins}(p_1, c_1)
\]
\[
\text{else} \quad \text{return } \text{ins}(p_1 + 1, c_1)
\]
\[
\text{endif}
\]
Example transformation functions

\[ T(ins(p_1, c_1), ins(p_2, c_2)) := \]
\[ \begin{align*}
  & \text{if } (p_1 < p_2) \text{ return } ins(p_1, c_1) \\
  & \text{else return } ins(p_1 + 1, c_1)
\end{align*} \]

\[ T(ins(p_1, c_1), del(p_2)) := \]
\[ \begin{align*}
  & \text{if } (p_1 \leq p_2) \text{ return } ins(p_1, c_1) \\
  & \text{else return } ins(p_1 - 1, c_1) \\
  & \text{endif}
\end{align*} \]

\[ T(del(p_1), ins(p_2, c_2)) := \]
\[ \begin{align*}
  & \text{if } (p_1 < p_2) \text{ return } del(p_1) \\
  & \text{else return } del(p_1 + 1)
\end{align*} \]

\[ T(del(p_1), del(p_2)) := \]
\[ \begin{align*}
  & \text{if } (p_1 < p_2) \text{ return } del(p_1) \\
  & \text{else if } (p_1 > p_2) \text{ return } del(p_1 - 1) \\
  & \text{else return } id()
\end{align*} \]
**Convergence but no intention preservation**

**Thomas Write Rule**

**Site 1**

\[
\text{op}_1 = \text{set}(s, \text{"AXB"})
\]

\[
\begin{align*}
(s, \text{"AB"}, f, (0h, 1), (0h, 1)) \\
(s, \text{"AXB"}, f, (0h, 1), (9h01, 1))
\end{align*}
\]

**Site 2**

\[
\begin{align*}
(s, \text{"AB"}, f, (0h, 1), (0h, 1)) \\
(s, \text{"AXB"}, f, (0h, 1), (9h01, 1))
\end{align*}
\]

**Site 3**

\[
\text{op}_2 = \text{set}(s, \text{"AYB"})
\]

\[
\begin{align*}
(s, \text{"AB"}, f, (0h, 1), (0h, 1)) \\
(s, \text{"AYB"}, f, (0h, 1), (9h02, 3))
\end{align*}
\]
Convergence – TP1 property

Site 1

\[ \text{Site 2} \]

\[ \text{op}_1 \]

\[ \text{op}_2 \]

\[ \text{op'}_1 \]

\[ \text{op'}_2 \]

- \( T(\text{op}_2: \text{operation}, \text{op}_1: \text{operation}) = \text{op'}_2 \)
  - \( \text{op}_1 \) and \( \text{op}_2 \) concurrent, defined on a state \( S \)
  - \( \text{op'}_2 \) same effects as \( \text{op}_2 \), defined on \( S.\text{op}_1 \)

\[
[TP1] \quad \text{op}_1 \circ T(\text{op}_2, \text{op}_1) \equiv \text{op}_2 \circ T(\text{op}_1, \text{op}_2)
\]
Convergence – TP2 property

\[\text{TP2} \quad T(op_3, op_1 \circ T(op_2, op_1)) = T(op_3, op_2 \circ T(op_1, op_2))\]
OT Problems

- Design and verify Transformation functions $T$
- $T$ also known as transpose_fd
- Verification of conditions TP1 and TP2
  - Combinatorial explosion (>100 cases for a string)
  - Iterative process
  - Repetitive and error prone task
Partial concurrency

Site 1
"telefone"

\[ op_1 = \text{ins}(5, p) \]

"telepfone"

\[ op_3 = \text{ins}(6, h) \]

"telephfone"

\[ op_2 = \text{del}(5) \]

Site 2
"telefone"

\[ op_1', = \text{ins}(5, p) \]

"telepone"

\[ op_2', = \text{del}(6) \]

\[ op_2'' = \text{del}(7) \]

\[ op_1' = \text{T}(op_1, op_2) = \text{ins}(5) \]

\[ T(op_3, op_2) \text{ not allowed to be performed !!!} \]
Partial concurrency

Site 1

“telefone”

$op_1 = \text{ins}(5, p)$

“telefone”

$op_3 = \text{ins}(6, h)$

“telefone”

$op_2 = \text{del}(5)$

Site 2

“telefone”

$op_1 = \text{ins}(5, p)$

“telefone”

“telefone”

$op_3 = \text{ins}(6, h)$

“telefone”

 transpose_bk

$op_2 = T(op_2, op_1) = \text{del}(6)$

$op_1 = \text{ins}(5, p)$

Site 2

“telefone”

“telefone”

$op_3 = T(op_3, op_2) = \text{ins}(6, h)$

??

$op_2 = \text{del}(7)$

“telefone”

“telefone”

“telefone”

“telefone”

“telefone”

“telefone”

Claudia Ignat, ignatcla@loria.fr
Partial concurrency

- \(\text{transpose}_\text{bk}(\text{op}_1, \text{op}'_2) = (\text{op}_2, \text{op}'_1)\)
  - \(\text{op}'_2 = T(\text{op}_2, \text{op}_1)\)
    - Therefore, \(\text{op}_2 = T^{-1}(\text{op}'_2, \text{op}_1)\)
  - \(\text{op}'_1 = T(\text{op}_1, \text{op}_2)\)
Example transformation functions

\[ T(\text{ins}(p_1, c_1), \text{ins}(p_2, c_2)) :\]
\[
\begin{align*}
  & \text{if } (p_1 < p_2) \text{ return } \text{ins}(p_1, c_1) \\
  & \text{else return } \text{ins}(p_1+1, c_1)
\end{align*}
\]

\[ T(\text{ins}(p_1, c_1), \text{del}(p_2)) :\]
\[
\begin{align*}
  & \text{if } (p_1 \leq p_2) \text{ return } \text{ins}(p_1, c_1) \\
  & \text{else return } \text{ins}(p_1-1, c_1) \\
  & \text{endif}
\end{align*}
\]

\[ T(\text{del}(p_1), \text{ins}(p_2, c_2)) :\]
\[
\begin{align*}
  & \text{if } (p_1 < p_2) \text{ return } \text{del}(p_1) \\
  & \text{else return } \text{del}(p_1+1)
\end{align*}
\]

\[ T(\text{del}(p_1), \text{del}(p_2)) :\]
\[
\begin{align*}
  & \text{if } (p_1 < p_2) \text{ return } \text{del}(p_1) \\
  & \text{else if } (p_1 > p_2) \text{ return } \text{del}(p_1-1) \\
  & \text{else return } \text{id}()
\end{align*}
\]
Ressel transformation functions (*)

\[
T(\text{ins}(p_1,c_1,u_1), \text{ins}(p_2,c_2,u_2)) :- \\
\quad \text{if } ((p_1<p_2) \text{ or } (p_1=p_2 \text{ and } u_1<u_2)) \text{ return } \text{ins}(p_1,c_1,u_1) \\
\quad \text{else return } \text{ins}(p_1+1,c_1,u_1)
\]

\[
T(\text{ins}(p_1,c_1,u_1), \text{del}(p_2,u_2)) :- \\
\quad \text{if } (p_1\leq p_2) \text{ return } \text{ins}(p_1,c_1,u_1) \\
\quad \text{else return } \text{ins}(p_1-1,c_1,u_1) \\
\quad \text{endif}
\]

\[
T(\text{del}(p_1,u_1), \text{ins}(p_2,c_2,u_2)) :- \\
\quad \text{if } (p_1<p_2) \text{ return } \text{del}(p_1,u_1) \\
\quad \text{else return } \text{del}(p_1+1,u_1)
\]

\[
T(\text{del}(p_1,u_1), \text{del}(p_2,u_2)) :- \\
\quad \text{if } (p_1<p_2) \text{ return } \text{del}(p_1,u_1) \\
\quad \text{else if } (p_1>p_2) \text{ return } \text{del}(p_1-1,u_1) \\
\quad \text{else return } \text{id}()
\]

TP1 ok, but not TP2 !
False-tie problem

Site 1  “abc”

Site 2  “abc”

Site 3  “abc”

$\text{op}_1 = \text{Insert}(2, x)$

$\text{op}_2 = \text{Delete}(2, b)$

$\text{op}_3 = \text{Insert}(3, y)$

“axbc”

“ac”

“abyc”

$\text{op}_1' = \text{Insert}(2, x)$

$\text{op}_3' = \text{Insert}(2, y)$

“axyc”?  “ayxc”?
TTF (Tombstone Transformation Functions) Approach (*)

- Keep “tombstones” of deleted elements

Tombstone Transformation Functions

- \( T(\text{insert}(p_1, el_1, sid_1), \text{insert}(p_2, el_2, sid_2)) \) {
  
  if \( p_1 < p_2 \) return \( \text{insert}(p_1, el_1, sid_1) \)
  
  else if \( p_1 = p_2 \) and \( sid_1 < sid_2 \) return \( \text{insert}(p_1, el_1, sid_1) \)
  
  else return \( \text{insert}(p_1 + 1, el_1, sid_1) \)

- \( T(\text{insert}(p_1, el_1, sid_1), \text{delete}(p_2, el_2, sid_2)) \) {
  
  return \( \text{insert}(p_1, el_1, sid_1) \)

- \( T(\text{delete}(p_1, sid_1), \text{insert}(p_2, sid_2)) \) {
  
  if \( p_1 < p_2 \) return \( \text{delete}(p_1, sid_1) \)
  
  else return \( \text{delete}(p_1 + 1, sid_1) \)

- \( T(\text{delete}(p_1, sid_1), \text{delete}(p_2, sid_2)) \) {
  
  return \( \text{delete}(p_1, sid_1) \)
Compacted storage model

- Compacted model = sequence of (character, abs_pos)
Delta storage model

- Delta model = sequence of (character, offset)
Models comparison

• Basic Model
  ▪ Deleted characters are kept
  ▪ Size of the model is growing infinitely

• Compacted Model
  ▪ Update absolute position of all characters located after the effect position

• Delta Model
  ▪ Update the offset of next character

• Our observations
  ▪ View2model can be optimised (caret position)
  ▪ Overhead of view2model is not significant