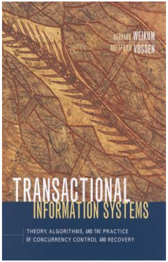
Transactional Information Systems:

Theory, Algorithms, and the Practice of Concurrency Control and Recovery

Gerhard Weikum and Gottfried Vossen

© 2002 Morgan Kaufmann
ISBN 1-55860-508-8


GERHARD WEIKUM
GOTTFRIED VOSSEN
TRANSACTIONAL
INFORMATION SYSTEMS
THEORY, ALGORITHMS, AND THE PRACTICE
OF CONCURRENCY CONTROL AND RECOVERY

"Teamwork is essential. It allows you to blame someone else."(Anonymous)

Part II: Concurrency Control

- 3 Concurrency Control: Notions of Correctness for the Page Model
- 4 Concurrency Control Algorithms
- 5 Multiversion Concurrency Control
- 6 Concurrency Control on Objects: Notions of Correctness
- 7 Concurrency Control Algorithms on Objects
- 8 Concurrency Control on Relational Databases
- 9 Concurrency Control on Search Structures
- 10 Implementation and Pragmatic Issues

Chapter 3: Concurrency Control – Notions of Correctness for the Page Model

- **3.2 Canonical Synchronization Problems**
- 3.3 Syntax of Histories and Schedules
- 3.4 Correctness of Histories and Schedules
- 3.5 Herbrand Semantics of Schedules
- 3.6 Final-State Serializability
- 3.7 View Serializability
- 3.8 Conflict Serializability
- 3.9 Commit Serializability
- 3.10 An Alternative Criterion: Interleaving Specifications
- 3.11 Lessons Learned

"Nothing is as practical as a good theory." (Albert Einstein)

Lost Update Problem

P1	Time	P2
	/* x = 100 */	
r (x)	**1**	
	2	r (x)
x := x+100	**4**	x := x+200
w (x)	**5**	
	/* x = 200 */	
	6	w (x)
	/* x = 300 */	

↑

update “lost”

Lost Update Problem

P1	**Time**	**P2**
	/* x = 100 */	
r (x)	**1**	
	2	**r (x)**
x := x+100	**4**	**x := x+200**
w (x)	**5**	
	/* x = 200 */	
	6	**w (x)**
	/* x = 300 */	

↑

update “lost”

Observation: problem is the interleaving $r_1(x)\ r_2(x)\ w_1(x)\ w_2(x)$

Inconsistent Read Problem

P1	Time	P2
	1	r (x)
	2	x := x – 10
	3	w (x)
sum := 0	4	
r (x)	5	
r (y)	6	
sum := sum +x	7	
sum := sum + y	8	
	9	r (y)
	10	y := y + 10
	11	w (y)

↑

“sees” wrong sum

Inconsistent Read Problem

P1	Time	P2
	1	r (x)
	2	x := x – 10
	3	w (x)
sum := 0	4	
r (x)	5	
r (y)	6	
sum := sum +x	7	
sum := sum + y	8	
	9	r (y)
	10	y := y + 10
	11	w (y)

↑

“sees” wrong sum

Observations:

problem is the interleaving $r_2(x)\ w_2(x)\ r_1(x)\ r_1(y)\ r_2(y)\ w_2(y)$

no problem with sequential execution

Dirty Read Problem

P1	Time	P2
r (x)	1	
x := x + 100	2	
w (x)	3	
	4	r (x)
	5	x := x - 100
failure & rollback	6	
	7	w (x)

↑

cannot rely on validity
of previously read data

Dirty Read Problem

P1	Time	P2
r (x)	1	
x := x + 100	2	
w (x)	3	
	4	r (x)
	5	x := x - 100
failure & rollback	6	
	7	w (x)

↑

cannot rely on validity
of previously read data

Observation: transaction rollbacks could affect concurrent transactions

Chapter 3: Concurrency Control – Notions of Correctness for the Page Model

- 3.2 Canonical Synchronization Problems
- **3.3 Syntax of Histories and Schedules**
- 3.4 Correctness of Histories and Schedules
- 3.5 Herbrand Semantics of Schedules
- 3.6 Final-State Serializability
- 3.7 View Serializability
- 3.8 Conflict Serializability
- 3.9 Commit Serializability
- 3.10 An Alternative Criterion: Interleaving Specifications
- 3.11 Lessons Learned

Schedules and Histories

Definition 3.1 (Schedules and histories):
Let $T=\{t_1, ..., t_n\}$ be a set of transactions, where each $t_i \in T$ has the form $t_i=(op_i, <_i)$ with op_i denoting the operations of t_i and $<_i$ their ordering.

(i) A **history** for T is a pair $s=(op(s),<_s)$ s.t.

(a) $op(s) \subseteq \cup_{i=1..n}\, op_i \cup \cup_{i=1..n} \{a_i, c_i\}$

(b) for all i, $1 \le i \le n$: $c_i \in op(s) \Leftrightarrow a_i \notin op(s)$

(c) $\cup_{i=1..n} <_i \subseteq <_s$

(d) for all i, $1 \le i \le n$, and all $p \in op_i$: $p <_s c_i$ or $p <_s a_i$

(e) for all $p, q \in op(s)$ s.t. at least one of them is a write and both access the same data item: $p <_s q$ or $q <_s p$

(ii) A **schedule** is a prefix of a history.

Schedules and Histories

Definition 3.1 (Schedules and histories):

Let $T=\{t_1, ..., t_n\}$ be a set of transactions, where each $t_i \in T$ has the form $t_i=(op_i, <_i)$ with op_i denoting the operations of t_i and $<_i$ their ordering.

(i) A **history** for T is a pair $s=(op(s),<_s)$ s.t.

(a) $op(s) \subseteq \cup_{i=1..n} op_i \cup \cup_{i=1..n} \{a_i, c_i\}$

(b) for all i, $1 \le i \le n$: $c_i \in op(s) \Leftrightarrow a_i \notin op(s)$

(c) $\cup_{i=1..n} <_i \subseteq <_s$

(d) for all i, $1 \le i \le n$, and all $p \in op_i$: $p <_s c_i$ or $p <_s a_i$

(e) for all $p, q \in op(s)$ s.t. at least one of them is a write and both access the same data item: $p <_s q$ or $q <_s p$

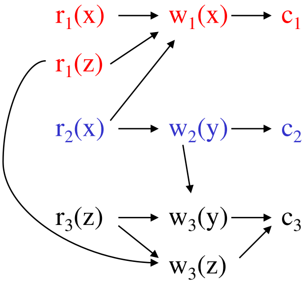
(ii) A **schedule** is a prefix of a history.

Definition 3.2 (Serial history):

A history s is **serial** if for any two transactions t_i and t_j in s, where $i \neq j$, all operations from t_i are ordered in s before all operations from t_j or vice versa.

Example Schedules and Notation

Example 3.4:



trans(s):=

$\{t_i \mid s \text{ contains step of } t_i\}$

commit(s):=

$\{t_i \in \text{trans}(s) \mid c_i \in s\}$

abort(s):=

$\{t_i \in \text{trans}(s) \mid a_i \in s\}$

active(s):=

$\text{trans}(s) - (\text{commit}(s) \cup \text{abort}(s))$

Example 3.6:

$r_1(x)\ r_2(z)\ r_3(x)\ w_2(x)\ w_1(x)\ r_3(y)\ r_1(y)\ w_1(y)\ w_2(z)\ w_3(z)\ c_1\ a_3$

Chapter 3: Concurrency Control – Notions of Correctness for the Page Model

- 3.2 Canonical Synchronization Problems
- 3.3 Syntax of Histories and Schedules
- **3.4 Correctness of Histories and Schedules**
- 3.5 Herbrand Semantics of Schedules
- 3.6 Final-State Serializability
- 3.7 View Serializability
- 3.8 Conflict Serializability
- 3.9 Commit Serializability
- 3.10 An Alternative Criterion: Interleaving Specifications
- 3.11 Lessons Learned

Correctness of Schedules

1. Define equivalence relation≈≈on set S of all schedules.
2. “Good” schedules are those in the equivalence classes of serial schedules.

- Equivalence must be efficiently decidable.
- “Good” equivalence classes should be “sufficiently large”.

For the moment,
disregard aborts: assume that all transactions are committed.

Activity

- What is an equivalence relation?

- List the three defining conditions!

Chapter 3: Concurrency Control – Notions of Correctness for the Page Model

- 3.2 Canonical Synchronization Problems
- 3.3 Syntax of Histories and Schedules
- 3.4 Correctness of Histories and Schedules
- **3.5 Herbrand Semantics of Schedules**
- 3.6 Final-State Serializability
- 3.7 View Serializability
- 3.8 Conflict Serializability
- 3.9 Commit Serializability
- 3.10 An Alternative Criterion: Interleaving Specifications
- 3.11 Lessons Learned

Herbrand Semantics of Schedules

Definition 3.3 (Herbrand Semantics of Steps):

For schedule s the **Herbrand semantics H_s** of steps $r_i(x)$, $w_i(x) \in$ op(s) is:

(i) $H_s[r_i(x)] := H_s[w_j(x)]$ where $w_j(x)$ is the last write on x in s before $r_i(x)$.

(ii) $H_s[w_i(x)] := f_{ix}(H_s[r_i(y_1)], ..., H_s[r_i(y_m)])$ where the $r_i(y_j)$, $1 \le j \le m$, are all read operations of t_i that occcur in s before $w_i(x)$ and f_{ix} is an uninterpreted m-ary function symbol.

Herbrand Semantics of Schedules

Definition 3.3 (Herbrand Semantics of Steps):
For schedule s the **Herbrand semantics H_s** of steps $r_i(x)$, $w_i(x) \in op(s)$ is:
(i) $H_s[r_i(x)] := H_s[w_j(x)]$ where $w_j(x)$ is the last write on x in s before $r_i(x)$.
(ii) $H_s[w_i(x)] := f_{ix}(H_s[r_i(y_1)], ..., H_s[r_i(y_m)])$ where
the $r_i(y_j)$, $1 \le j \le m$, are all read operations of t_i that occcur in s before $w_i(x)$
and f_{ix} is an uninterpreted m-ary function symbol.

Definition 3.4 (Herbrand Universe):
For data items D={x, y, z, ...} and transactions t_i, $1 \le i \le n$,
the **Herbrand universe HU** is the smallest set of symbols s.t.
(i) $f_{0x}(\,) \in HU$ for each $x \in D$ where f_{0x} is a constant, and
(ii) if $w_i(x) \in op_i$ for some t_i, there are m read operations $r_i(y_1)$, ..., $r_i(y_m)$
that precede $w_i(x)$ in t_i, and $v_1, ..., v_m \in HU$, then $f_{ix}(v_1, ..., v_m) \in HU$.

Herbrand Semantics of Schedules

Definition 3.3 (Herbrand Semantics of Steps):
For schedule s the **Herbrand semantics H_s** of steps $r_i(x)$, $w_i(x) \in op(s)$ is:
(i) $H_s[r_i(x)] := H_s[w_j(x)]$ where $w_j(x)$ is the last write on x in s before $r_i(x)$.
(ii) $H_s[w_i(x)] := f_{ix}(H_s[r_i(y_1)], ..., H_s[r_i(y_m)])$ where the $r_i(y_j)$, $1 \le j \le m$, are all read operations of t_i that occcur in s before $w_i(x)$ and f_{ix} is an uninterpreted m-ary function symbol.

Definition 3.4 (Herbrand Universe):
For data items D={x, y, z, ...} and transactions t_i, $1 \le i \le n$,
the **Herbrand universe HU** is the smallest set of symbols s.t.
(i) $f_{0x}(\,) \in$ HU for each $x \in D$ where f_{0x} is a constant, and
(ii) if $w_i(x) \in op_i$ for some t_i, there are m read operations $r_i(y_1), ..., r_i(y_m)$ that precede $w_i(x)$ in t_i, and $v_1, ..., v_m \in$ HU, then $f_{ix}(v_1, ..., v_m) \in$ HU.

Definition 3.5 (Schedule Semantics):
The **Herbrand semantics of a schedule** s is the mapping
$H[s]: D \rightarrow HU$ defined by $H[s](x) := H_s[w_i(x)]$,
where $w_i(x)$ is the last operation from s writing x, for each $x \in D$.

Herbrand Semantics: Example

$s = w_0(x)\ w_0(y)\ c_0\ r_1(x)\ r_2(y)\ w_2(x)\ w_1(y)\ c_2\ c_1$

$H_s[w_0(x)] = f_{0x}(\ )$
$H_s[w_0(y)] = f_{0y}(\ )$
$H_s[r_1(x)] = H_s[w_0(x)] = f_{0x}(\ )$
$H_s[r_2(y)] = H_s[w_0(y)] = f_{0y}(\ )$
$H_s[w_2(x)] = f_{2x}(H_s[r_2(y)]) = f_{2x}(f_{0y}(\ ))$
$H_s[w_1(y)] = f_{1y}(H_s[r_1(x)]) = f_{1y}(f_{0x}(\ ))$

$H[s](x) = H_s[w_2(x)] = f_{2x}(f_{0y}(\ ))$
$H[s](y) = H_s[w_1(y)] = f_{1y}(f_{0x}(\ ))$

Chapter 3: Concurrency Control – Notions of Correctness for the Page Model

- 3.2 Canonical Synchronization Problems
- 3.3 Syntax of Histories and Schedules
- 3.4 Correctness of Histories and Schedules
- 3.5 Herbrand Semantics of Schedules
- **3.6 Final-State Serializability**
- 3.7 View Serializability
- 3.8 Conflict Serializability
- 3.9 Commit Serializability
- 3.10 An Alternative Criterion: Interleaving Specifications
- 3.11 Lessons Learned

Final-State Equivalence

Definition 3.6 (Final State Equivalence):
Schedules s and s' are called **final state equivalent**, denoted $s \approx_f s'$, if op(s)=op(s') and H[s]=H[s'].

Final-State Equivalence

Definition 3.6 (Final State Equivalence):
Schedules s and s' are called **final state equivalent**, denoted $s \approx_f s'$, if op(s)=op(s') and H[s]=H[s'].

Example a:

$$\left.\begin{array}{l}
s = r_1(x)\ r_2(y)\ w_1(y)\ r_3(z)\ w_3(z)\ r_2(x)\ w_2(z)\ w_1(x) \\
s' = r_3(z)\ w_3(z)\ r_2(y)\ r_2(x)\ w_2(z)\ r_1(x)\ w_1(y)\ w_1(x) \\
H[s](x) = H_s[w_1(x)] = f_{1x}(f_{0x}(\,)) = H_{s'}[w_1(x)] = H[s'](x) \\
H[s](y) = H_s[w_1(y)] = f_{1y}(f_{0x}(\,)) = H_{s'}[w_1(y)] = H[s'](y) \\
H[s](z) = H_s[w_2(z)] = f_{2z}(f_{0x}(\,), f_{0y}(\,)) = H_{s'}[w_2(z)] = H[s'](z)
\end{array}\right\} \Rightarrow s \approx_f s'$$

Final-State Equivalence

Definition 3.6 (Final State Equivalence):
Schedules s and s' are called **final state equivalent**, denoted $s \approx_f s'$, if op(s)=op(s') and H[s]=H[s'].

Example a:

$s = r_1(x)\ r_2(y)\ w_1(y)\ r_3(z)\ w_3(z)\ r_2(x)\ w_2(z)\ w_1(x)$

$s' = r_3(z)\ w_3(z)\ r_2(y)\ r_2(x)\ w_2(z)\ r_1(x)\ w_1(y)\ w_1(x)$

$H[s](x) = H_s[w_1(x)] = f_{1x}(f_{0x}(\,)) = H_{s'}[w_1(x)] = H[s'](x)$

$H[s](y) = H_s[w_1(y)] = f_{1y}(f_{0x}(\,)) = H_{s'}[w_1(y)] = H[s'](y)$

$H[s](z) = H_s[w_2(z)] = f_{2z}(f_{0x}(\,), f_{0y}(\,)) = H_{s'}[w_2(z)] = H[s'](z)$

$\Rightarrow s \approx_f s'$

Example b:

$s = r_1(x)\ r_2(y)\ w_1(y)\ w_2(y)$

$s' = r_1(x)\ w_1(y)\ r_2(y)\ w_2(y)$

$H[s](y) = H_s[w_2(y)] = f_{2y}(f_{0y}(\,))$

$H[s'](y) = H_{s'}[w_2(y)] = f_{2y}(f_{1y}(f_{0x}(\,)))$

$\Rightarrow \neg\,(s \approx_f s')$

Reads-from Relation

Definition 3.7 (Reads-from Relation; Useful, Alive, and Dead Steps):
Given a schedule s, extended with an initial and a final transaction, t_0 and t_∞.

(i) **$r_j(x)$ reads x in s from $w_i(x)$** if $w_i(x)$ is the last write on x s.t. $w_i(x) <_s r_j(x)$.

(ii) The **reads-from relation** of s is
$RF(s) := \{(t_i, x, t_j) \mid \text{an } r_j(x) \text{ reads x from a } w_i(x)\}$.

(iii) Step p is **directly useful** for step q, denoted $p \rightarrow q$, if q reads from p, or p is a read step and q is a subsequent write step of the same transaction.
$\rightarrow^*$, the **"useful" relation**, denotes the reflexive and transitive closure of $\rightarrow$.

(iv) Step p is **alive** in s if it is useful for some step from t_∞, and **dead** otherwise.

(v) The **live-reads-from relation** of s is
$LRF(s) := \{(t_i, x, t_j) \mid \text{an alive } r_j(x) \text{ reads x from } w_i(x)\}$

Reads-from Relation

Definition 3.7 (Reads-from Relation; Useful, Alive, and Dead Steps):
Given a schedule s, extended with an initial and a final transaction, t_0 and t_∞.

(i) **$r_j(x)$ reads x in s from $w_i(x)$** if $w_i(x)$ is the last write on x s.t. $w_i(x) <_s r_j(x)$.

(ii) The **reads-from relation** of s is
$RF(s) := \{(t_i, x, t_j) \mid \text{an } r_j(x) \text{ reads x from a } w_i(x)\}$.

(iii) Step p is **directly useful** for step q, denoted $p \rightarrow q$, if q reads from p, or p is a read step and q is a subsequent write step of the same transaction.
$\rightarrow^*$, the **"useful" relation**, denotes the reflexive and transitive closure of $\rightarrow$.

(iv) Step p is **alive** in s if it is useful for some step from t_∞, and **dead** otherwise.

(v) The **live-reads-from relation** of s is
$LRF(s) := \{(t_i, x, t_j) \mid \text{an alive } r_j(x) \text{ reads x from } w_i(x)\}$

Example 3.7:

$s = r_1(x)\ r_2(y)\ w_1(y)\ w_2(y)$

$s' = r_1(x)\ w_1(y)\ r_2(y)\ w_2(y)$

$RF(s) = \{(t_0,x,t_1), (t_0,y,t_2), (t_0,x,t_\infty), (t_2,y,t_\infty)\}$

$RF(s') = \{(t_0,x,t_1), (t_1,y,t_2), (t_0,x,t_\infty), (t_2,y,t_\infty)\}$

$LRF(s) = \{(t_0,y,t_2), (t_0,x,t_\infty), (t_2,y,t_\infty)\}$

$LRF(s') = \{(t_0,x,t_1), (t_1,y,t_2), (t_0,x,t_\infty), (t_2,y,t_\infty)\}$

Final-State Serializability

Theorem 3.1:

For schedules s and s' the following statements hold.

(i) $s \approx_f s'$ iff op(s)=op(s') and LRF(s)=LRF(s').

(ii) For s let the step graph D(s)=(V,E) be a directed graph with vertices V:=op(s) and edges $E:=\{(p,q) \mid p \rightarrow q\}$, and the reduced step graph $D_1(s)$ be derived from D(s) by removing all vertices that correspond to dead steps. Then LRF(s)=LRF(s') iff $D_1(s)=D_1(s')$.

Final-State Serializability

Theorem 3.1:
For schedules s and s' the following statements hold.

(i) $s \approx_f s'$ iff op(s)=op(s') and LRF(s)=LRF(s').

(ii) For s let the step graph D(s)=(V,E) be a directed graph with vertices V:=op(s) and edges $E:=\{(p,q) \mid p \to q\}$, and the reduced step graph $D_1(s)$ be derived from D(s) by removing all vertices that correspond to dead steps. Then LRF(s)=LRF(s') iff $D_1(s)=D_1(s')$.

Corollary 3.1:
Final-state equivalence of two schedules s and s' can be decided in time that is polynomial in the length of the two schedules.

Final-State Serializability

Theorem 3.1:
For schedules s and s' the following statements hold.

(i) $s \approx_f s'$ iff op(s)=op(s') and LRF(s)=LRF(s').

(ii) For s let the step graph D(s)=(V,E) be a directed graph with vertices V:=op(s) and edges E:={(p,q) | p→q}, and the reduced step graph $D_1(s)$ be derived from D(s) by removing all vertices that correspond to dead steps. Then LRF(s)=LRF(s') iff $D_1(s)=D_1(s')$.

Corollary 3.1:
Final-state equivalence of two schedules s and s' can be decided in time that is polynomial in the length of the two schedules.

Definition 3.8 (Final State Serializability):
A schedule s is **final state serializable** if there is a serial schedule s' s.t. $s \approx_f s'$.
FSR denotes the class of all final-state serializable histories.

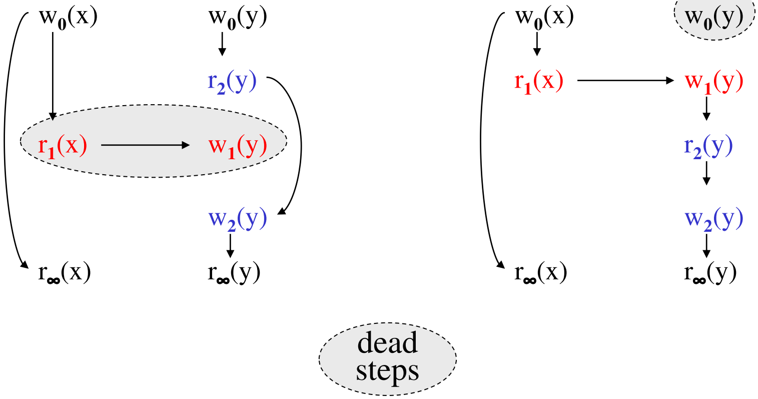
FSR: Example 3.9

s= $r_1(x)\ r_2(y)\ w_1(y)\ w_2(y)$

s'= $r_1(x)\ w_1(y)\ r_2(y)\ w_2(y)$

D(s):

D(s'):



Chapter 3: Concurrency Control – Notions of Correctness for the Page Model

- 3.2 Canonical Synchronization Problems
- 3.3 Syntax of Histories and Schedules
- 3.4 Correctness of Histories and Schedules
- 3.5 Herbrand Semantics of Schedules
- 3.6 Final-State Serializability
- **3.7 View Serializability**
- 3.8 Conflict Serializability
- 3.9 Commit Serializability
- 3.10 An Alternative Criterion: Interleaving Specifications
- 3.11 Lessons Learned

Canonical Anomalies Reconsidered

- **Lost update anomaly:**
  $L = r_1(x)\ r_2(x)\ w_1(x)\ w_2(x)\ c_1\ c_2$

  → history is not FSR

  $LRF(L) = \{(t_0,x,t_2), (t_2,x,t_\infty)\}$
  $LRF(t_1\ t_2) = \{(t_0,x,t_1), (t_1,x,t_2), (t_2,x,t_\infty)\}$
  $LRF(t_2\ t_1) = \{(t_0,x,t_2), (t_2,x,t_1), (t_1,x,t_\infty)\}$

- **Inconsistent read anomaly:**
  $I = r_2(x)\ w_2(x)\ r_1(x)\ r_1(y)\ r_2(y)\ w_2(y)\ c_1\ c_2$

  → history is FSR !

  $LRF(I) = \{(t_0,x,t_2), (t_0,y,t_2), (t_2,x,t_\infty), (t_2,y,t_\infty)\}$
  $LRF(t_1\ t_2) = \{(t_0,x,t_2), (t_0,y,t_2), (t_2,x,t_\infty), (t_2,y,t_\infty)\}$
  $LRF(t_2\ t_1) = \{(t_0,x,t_2), (t_0,y,t_2), (t_2,x,t_\infty), (t_2,y,t_\infty)\}$

Canonical Anomalies Reconsidered

- **Lost update anomaly:**
  $L = r_1(x)\ r_2(x)\ w_1(x)\ w_2(x)\ c_1\ c_2$

→ history is not FSR

$LRF(L) = \{(t_0,x,t_2), (t_2,x,t_\infty)\}$
$LRF(t_1\ t_2) = \{(t_0,x,t_1), (t_1,x,t_2), (t_2,x,t_\infty)\}$
$LRF(t_2\ t_1) = \{(t_0,x,t_2), (t_2,x,t_1), (t_1,x,t_\infty)\}$

- **Inconsistent read anomaly:**
  $I = r_2(x)\ w_2(x)\ r_1(x)\ r_1(y)\ r_2(y)\ w_2(y)\ c_1\ c_2$

→ history is FSR !

$LRF(I) = \{(t_0,x,t_2), (t_0,y,t_2), (t_2,x,t_\infty), (t_2,y,t_\infty)\}$
$LRF(t_1\ t_2) = \{(t_0,x,t_2), (t_0,y,t_2), (t_2,x,t_\infty), (t_2,y,t_\infty)\}$
$LRF(t_2\ t_1) = \{(t_0,x,t_2), (t_0,y,t_2), (t_2,x,t_\infty), (t_2,y,t_\infty)\}$

Observation: (Herbrand) semantics of all read steps matters!

View Serializability

Definition 3.9 (View Equivalence):

Schedules s and s' are **view equivalent**, denoted $s \approx_v s'$, if the following hold:

(i) op(s)=op(s')

(ii) H[s] = H[s']

(iii) $H_s[p] = H_{s'}[p]$ for all (read or write) steps

View Serializability

Definition 3.9 (View Equivalence):

Schedules s and s' are **view equivalent**, denoted $s \approx_v s'$, if the following hold:

(i) op(s)=op(s')

(ii) H[s] = H[s']

(iii) $H_s[p] = H_{s'}[p]$ for all (read or write) steps

Theorem 3.2:

For schedules s and s' the following statements hold.

(i) $s \approx_v s'$ iff op(s)=op(s') and RF(s)=RF(s')

(ii) $s \approx_v s'$ iff D(s)=D(s')

View Serializability

Definition 3.9 (View Equivalence):
Schedules s and s' are **view equivalent**, denoted $s \approx_v s'$, if the following hold:

(i) op(s)=op(s')

(ii) H[s] = H[s']

(iii) $H_s[p] = H_{s'}[p]$ for all (read or write) steps

Theorem 3.2:
For schedules s and s' the following statements hold.

(i) $s \approx_v s'$ iff op(s)=op(s') and RF(s)=RF(s')

(ii) $s \approx_v s'$ iff D(s)=D(s')

Corollary 3.2:
View equivalence of two schedules s and s' can be decided in time that is polynomial in the length of the two schedules.

View Serializability

Definition 3.9 (View Equivalence):
Schedules s and s' are **view equivalent**, denoted $s \approx_v s'$, if the following hold:
(i) op(s)=op(s')
(ii) H[s] = H[s']
(iii) $H_s[p] = H_{s'}[p]$ for all (read or write) steps

Theorem 3.2:
For schedules s and s' the following statements hold.
(i) $s \approx_v s'$ iff op(s)=op(s') and RF(s)=RF(s')
(ii) $s \approx_v s'$ iff D(s)=D(s')

Corollary 3.2:
View equivalence of two schedules s and s' can be decided in time that is polynomial in the length of the two schedules.

Definition 3.10 (View Serializability):
A schedule s is **view serializable** if there exists a serial schedule s' s.t. $s \approx_v s'$.
VSR denotes the class of all view-serializable histories.

Inconsistent Read Reconsidered

- **Inconsistent read anomaly:**

  $I = r_2(x)\ w_2(x)\ r_1(x)\ r_1(y)\ r_2(y)\ w_2(y)\ c_1\ c_2$

→ history is not VSR !

$RF(I) = \{(t_0,x,t_2), (t_2,x,t_1), (t_0,y,t_1), (t_0,y,t_2), (t_2,x,t_\infty), (t_2,y,t_\infty)\}$
$RF(t_1\ t_2) = \{(t_0,x,t_1), (t_0,y,t_1), (t_0,x,t_2), (t_0,y,t_2), (t_2,x,t_\infty), (t_2,y,t_\infty)\}$
$RF(t_2\ t_1) = \{(t_0,x,t_2), (t_0,y,t_2), (t_2,x,t_1), (t_2,y,t_1), (t_2,x,t_\infty), (t_2,y,t_\infty)\}$

Inconsistent Read Reconsidered

- **Inconsistent read anomaly:**

  $I = r_2(x)\ w_2(x)\ r_1(x)\ r_1(y)\ r_2(y)\ w_2(y)\ c_1\ c_2$

→ history is not VSR !

$RF(I) = \{(t_0,x,t_2), (t_2,x,t_1), (t_0,y,t_1), (t_0,y,t_2), (t_2,x,t_\infty), (t_2,y,t_\infty)\}$

$RF(t_1\ t_2) = \{(t_0,x,t_1), (t_0,y,t_1), (t_0,x,t_2), (t_0,y,t_2), (t_2,x,t_\infty), (t_2,y,t_\infty)\}$

$RF(t_2\ t_1) = \{(t_0,x,t_2), (t_0,y,t_2), (t_2,x,t_1), (t_2,y,t_1), (t_2,x,t_\infty), (t_2,y,t_\infty)\}$

Observation: VSR properly captures our intuition

Relationship Between VSR and FSR

Theorem 3.3:
VSR $\subset$ FSR.

Theorem 3.4:
Let s be a history without dead steps. Then s $\in$ VSR iff s $\in$ FSR.

On the Complexity of Testing VSR

Theorem 3.5:
The problem of deciding for a given schedule s whether s ∈ VSR holds is NP-complete.

Properties of VSR

Definition 3.11 (Monotone Classes of Histories)
Let s be a schedule and $T \subseteq trans(s)$. $\Pi_T(s)$ denotes the projection of s onto T.
A class E of histories is called monotone if the following holds:
if s is in E, then $\Pi_T(s)$ is in E for each $T \subseteq trans(s)$.
VSR is not monotone.

Example:

s = $w_1(x)\ w_2(x)\ w_2(y)\ c_2\ w_1(y)\ c_1\ w_3(x)\ w_3(y)\ c_3$ → ∈ VSR

$\Pi_{\{t1,\ t2\}}(s) = w_1(x)\ w_2(x)\ w_2(y)\ c_2\ w_1(y)\ c_1$ → ∉ VSR

Chapter 3: Concurrency Control – Notions of Correctness for the Page Model

- 3.2 Canonical Synchronization Problems
- 3.3 Syntax of Histories and Schedules
- 3.4 Correctness of Histories and Schedules
- 3.5 Herbrand Semantics of Schedules
- 3.6 Final-State Serializability
- 3.7 View Serializability
- **3.8 Conflict Serializability**
- 3.9 Commit Serializability
- 3.10 An Alternative Criterion: Interleaving Specifications
- 3.11 Lessons Learned

Conflict Serializability

Definition 3.12 (Conflicts and Conflict Relations):

Let s be a schedule, t, t' $\in$ trans(s), t $\neq$ t'.

(i) Two data operations p $\in$ t and q $\in$ t' are in **conflict** in s if they access the same data item and at least one of them is a write.

(ii) {(p, q)} | p, q are in conflict and p $<_s$ q} is the **conflict relation** of s.

Conflict Serializability

Definition 3.12 (Conflicts and Conflict Relations):
Let s be a schedule, t, t' $\in$ trans(s), t $\neq$ t'.

(i) Two data operations p $\in$ t and q $\in$ t' are in **conflict** in s if they access the same data item and at least one of them is a write.

(ii) {(p, q)} | p, q are in conflict and p $<_s$ q} is the **conflict relation** of s.

Definition 3.13 (Conflict Equivalence):
Schedules s and s' are **conflict equivalent**, denoted s $\approx_c$ s', if op(s) = op(s') and conf(s) = conf(s').

Conflict Serializability

Definition 3.12 (Conflicts and Conflict Relations):
Let s be a schedule, t, t' $\in$ trans(s), t $\neq$ t'.

(i) Two data operations p $\in$ t and q $\in$ t' are in **conflict** in s if they access the same data item and at least one of them is a write.

(ii) $\{(p, q)\} \mid p, q$ are in conflict and $p <_s q\}$ is the **conflict relation** of s.

Definition 3.13 (Conflict Equivalence):
Schedules s and s' are **conflict equivalent**, denoted $s \approx_c s'$, if op(s) = op(s') and conf(s) = conf(s').

Definition 3.14 (Conflict Serializability):
Schedule s is **conflict serializable** if there is a serial schedule s' s.t. $s \approx_c s'$.
CSR denotes the class of all conflict serializable schedules.

Conflict Serializability

Definition 3.12 (Conflicts and Conflict Relations):
Let s be a schedule, t, t' $\in$ trans(s), t $\neq$ t'.

(i) Two data operations $p \in t$ and $q \in t'$ are in **conflict** in s if they access the same data item and at least one of them is a write.

(ii) $\{(p, q)\} \mid p, q$ are in conflict and $p <_s q\}$ is the **conflict relation** of s.

Definition 3.13 (Conflict Equivalence):
Schedules s and s' are **conflict equivalent**, denoted $s \approx_c s'$, if op(s) = op(s') and conf(s) = conf(s').

Definition 3.14 (Conflict Serializability):
Schedule s is **conflict serializable** if there is a serial schedule s' s.t. $s \approx_c s'$.
CSR denotes the class of all conflict serializable schedules.

Example a: $r_1(x)\ r_2(x)\ r_1(z)\ w_1(x)\ w_2(y)\ r_3(z)\ w_3(y)\ c_1\ c_2\ w_3(z)\ c_3$ $\rightarrow \in$ CSR

Example b: $r_2(x)\ w_2(x)\ r_1(x)\ r_1(y)\ r_2(y)\ w_2(y)\ c_1\ c_2$ $\rightarrow \notin$ CSR

Properties of CSR

Theorem 3.8:
CSR $\subset$ VSR

Example: s = $w_1(x)\ w_2(x)\ w_2(y)\ c_2\ w_1(y)\ c_1\ w_3(x)\ w_3(y)\ c_3$
s $\in$ VSR, but s $\notin$ CSR.

Theorem 3.9:
(i) CSR is monotone.
(ii) s $\in$ CSR $\Leftrightarrow$ $\Pi_T(s) \in$ VSR for all T $\subseteq$ trans(s)
(i.e., CSR is the largest monotone subset of VSR).

Activity

- What is a directed graph?

- Think of ways to associate a graph with a schedule!

Conflict Graph

Definition 3.15 (Conflict Graph):
Let s be a schedule. The **conflict graph** G(s) = (V, E) is a directed graph with vertices V := commit(s) and
edges E := {(t, t') | t ≠ t' and there are steps p ∈ t, q ∈ t' with (p, q) ∈ conf(s)}.

Conflict Graph

Definition 3.15 (Conflict Graph):
Let s be a schedule. The **conflict graph** G(s) = (V, E) is a directed graph with vertices V := commit(s) and edges E := {(t, t') | t ≠ t' and there are steps p ∈ t, q ∈ t' with (p, q) ∈ conf(s)}.

Theorem 3.10:
Let s be a schedule. Then s ∈ CSR iff G(s) is acyclic.

Corollary 3.4:
Testing if a schedule is in CSR can be done in time polynomial to the schedule's number of transactions.

Conflict Graph

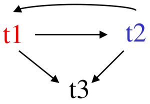
Definition 3.15 (Conflict Graph):
Let s be a schedule. The **conflict graph** G(s) = (V, E) is a directed graph with vertices V := commit(s) and edges E := {(t, t') | t ≠ t' and there are steps p ∈ t, q ∈ t' with (p, q) ∈ conf(s)}.

Theorem 3.10:
Let s be a schedule. Then s ∈ CSR iff G(s) is acyclic.

Corollary 3.4:
Testing if a schedule is in CSR can be done in time polynomial to the schedule's number of transactions.

Example 3.12:
s = $r_1(y)\ r_3(w)\ r_2(y)\ w_1(y)\ w_1(x)\ w_2(x)\ w_2(z)\ w_3(x)\ c_1\ c_3\ c_2$

G(s):



Activity

- What is a characterization (in a mathematical sense)?
- How do you prove a necessary and sufficient condition?
- What needs to be shown for the serializability theorem?

Proof of the Conflict-Graph Theorem

(i) Let s be a schedule in CSR. So there is a serial schedule s' with conf(s) = conf(s').
Now assume that G(s) has a cycle $t_1 \rightarrow t_2 \rightarrow ... \rightarrow t_k \rightarrow t_1$.
This implies that there are pairs $(p_1, q_2), (p_2, q_3), ... , (p_k, q_1)$
with $p_i \in t_i$, $q_i \in t_i$, $p_i <_s q_{(i+1)}$, and p_i in conflict with $q_{(i+1)}$.
Because s' $\approx_c$ s, it also implies that $p_i <_{s'} q_{(i+1)}$.
Because s' is serial, we obtain $t_i <_{s'} t_{(i+1)}$ for i=1, ..., k-1, and $t_k <_{s'} t_1$.
By transitivity we infer $t_1 <_{s'} t_2$ and $t_2 <_{s'} t_1$, which is impossible.
This contradiction shows that the initial assumption is wrong. So G(s) is acyclic.

(ii) Let G(s) be acyclic. So it must have at least one source node.
The following topological sort produces a total order < of transactions:
a) start with a source node (i.e., a node without incoming edges),
b) remove this node and all its outgoing edges,
c) iterate a) and b) until all nodes have been added to the sorted list.
The total transaction ordering order < preserves the edges in G(s);
therefore it yields a serial schedule s' for which s' $\approx_c$ s.

Commutativity and Ordering Rules

Commutativity rules:

C1: $r_i(x)\ r_j(y) \sim r_j(y)\ r_i(x)$ if $i \neq j$

C2: $r_i(x)\ w_j(y) \sim w_j(y)\ r_i(x)$ if $i \neq j$ and $x \neq y$

C3: $w_i(x)\ w_j(y) \sim w_j(y)\ w_i(x)$ if $i \neq j$ and $x \neq y$

Ordering rule:

C4: $o_i(x), p_j(y)$ unordered ~> $o_i(x)\ p_j(y)$
if $x \neq y$ or both o and p are reads

Example for transformations of schedules:

$s \quad = w_1(x)\ \underbrace{r_2(x)\ w_1(y)}\ w_1(z)\ \underbrace{r_3(z)\ w_2(y)}\ w_3(y)\ w_3(z)$

~>[C2] $\quad w_1(x)\ w_1(y)\ \underbrace{r_2(x)\ w_1(z)}\ w_2(y)\ r_3(z)\ w_3(y)\ w_3(z)$

~>[C2] $\quad w_1(x)\ w_1(y)\ w_1(z)\ r_2(x)\ w_2(y)\ r_3(z)\ w_3(y)\ w_3(z)$

$= t_1\ t_2\ t_3$

Commutativity-based Reducibility

Definition 3.16 (Commutativity Based Equivalence):
Schedules s and s' s.t. op(s)=op(s') are **commutativity based equivalent**, denoted s ~* s', if s can be transformed into s' by applying rules C1, C2, C3, C4 finitely many times.

Theorem 3.11:
Let s and s' be schedules s.t. op(s)=op(s'). Then s $\approx_c$ s' iff s ~* s'.

Commutativity-based Reducibility

Definition 3.16 (Commutativity Based Equivalence):
Schedules s and s' s.t. op(s)=op(s') are **commutativity based equivalent**, denoted s ~* s', if s can be transformed into s' by applying rules C1, C2, C3, C4 finitely many times.

Theorem 3.11:
Let s and s' be schedules s.t. op(s)=op(s'). Then s $\approx_c$ s' iff s ~* s'.

Definition 3.17 (Commutativity Based Reducibility):
Schedule s is **commutativity-based reducible** if there is a serial schedule s' s.t. s ~* s'.

Corollary 3.5:
Schedule s is commutativity-based reducible iff s $\in$ CSR.

Order Preserving Conflict Serializability

Definition 3.18 (Order Preservation):
Schedule s is **order preserving conflict serializable** if it is conflict equivalent to a serial schedule s' and
for all t, t' $\in$ trans(s): if t completely precedes t' in s, then the same holds in s'.
OCSR denotes the class of all schedules with this property.

Theorem 3.12:
OCSR $\subset$ CSR.

Example 3.13:

$s = w_1(x)\ r_2(x)\ c_2\ w_3(y)\ c_3\ w_1(y)\ c_1$ $\rightarrow \in$ CSR

$\rightarrow \notin$ OCSR

Commit-order Preserving Conflict Serializability

Definition 3.19 (Commit Order Preservation):
Schedule s is **commit order preserving conflict serializable** if
for all t_i, $t_j \in$ trans(s): if there are $p \in t_i$, $q \in t_j$ with $(p,q) \in$ conf(s) then $c_i <_s c_j$.
COCSR denotes the class of all schedules with this property.

Theorem 3.13:
COCSR $\subset$ CSR.

Commit-order Preserving Conflict Serializability

Definition 3.19 (Commit Order Preservation):
Schedule s is **commit order preserving conflict serializable** if
for all $t_i, t_j \in$ trans(s): if there are $p \in t_i$, $q \in t_j$ with $(p,q) \in$ conf(s) then $c_i <_s c_j$.
COCSR denotes the class of all schedules with this property.

Theorem 3.13:
COCSR $\subset$ CSR.

Theorem 3.14:
Schedule s is in COCSR iff there is a serial schedule s' s.t. $s \approx_c s'$ and
for all $t_i, t_j \in$ trans(s): $t_i <_{s'} t_j \Leftrightarrow c_i <_s c_j$.

Commit-order Preserving Conflict Serializability

Definition 3.19 (Commit Order Preservation):
Schedule s is **commit order preserving conflict serializable** if for all $t_i, t_j \in$ trans(s): if there are $p \in t_i$, $q \in t_j$ with $(p,q) \in$ conf(s) then $c_i <_s c_j$.
COCSR denotes the class of all schedules with this property.

Theorem 3.13:
COCSR $\subset$ CSR.

Theorem 3.14:
Schedule s is in COCSR iff there is a serial schedule s' s.t. $s \approx_c s'$ and for all $t_i, t_j \in$ trans(s): $t_i <_{s'} t_j \Leftrightarrow c_i <_s c_j$.

Theorem 3.15:
COCSR $\subset$ OCSR.

Example:

$s = w_3(y)\ c_3\ w_1(x)\ r_2(x)\ c_2\ w_1(y)\ c_1$ → $\in$ OCSR

→ $\notin$ COCSR

Chapter 3: Concurrency Control – Notions of Correctness for the Page Model

- 3.2 Canonical Synchronization Problems
- 3.3 Syntax of Histories and Schedules
- 3.4 Correctness of Histories and Schedules
- 3.5 Herbrand Semantics of Schedules
- 3.6 Final-State Serializability
- 3.7 View Serializability
- 3.8 Conflict Serializability
- **3.9 Commit Serializability**
- 3.10 An Alternative Criterion: Interleaving Specifications
- 3.11 Lessons Learned

Commit Serializability

Definition 3.20 (Closure Properties of Schedule Classes):
Let E be a class of schedules.
For schedule s let CP(s) denote the projection $\Pi_{commit(s)}$ (s).
E is **prefix-closed** if the following holds: $s \in E \Leftrightarrow p \in E$ for each prefix of s.
E is **commit-closed** if the following holds: $s \in E \Rightarrow CP(s) \in E$.

Theorem 3.16:
CSR is prefix-commit-closed, i.e., prefix-closed and commit-closed.

Commit Serializability

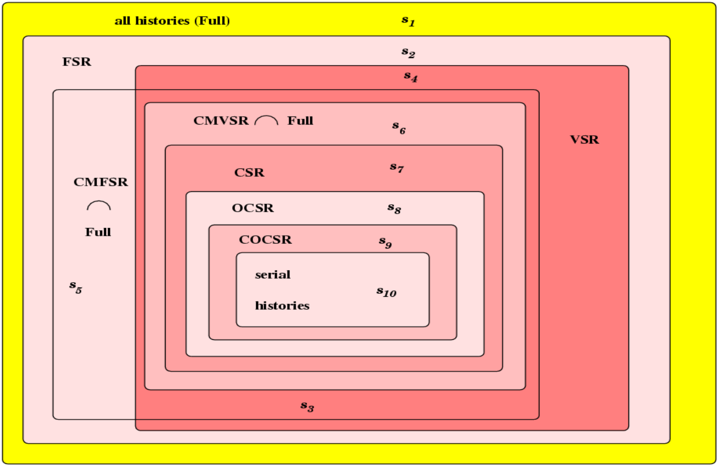
Definition 3.20 (Closure Properties of Schedule Classes):
Let E be a class of schedules.
For schedule s let CP(s) denote the projection $\Pi_{commit(s)}$ (s).
E is **prefix-closed** if the following holds: $s \in E \Leftrightarrow p \in E$ for each prefix of s.
E is **commit-closed** if the following holds: $s \in E \Rightarrow CP(s) \in E$.

Theorem 3.16:
CSR is prefix-commit-closed, i.e., prefix-closed and commit-closed.

Definition 3.21 (Commit Serializability):
Schedule s is **commit-Θ-serializable** if CP(p) is Θ-serializable for each prefix p of s, where Θ can be FSR, VSR, or CSR.
The resulting classes of commit-Θ-serializable schedules are denoted CMFSR, CMVSR, and CMCSR.

Theorem 3.17:
(i) CMFSR, CMVSR, CMCSR are prefix-commit-closed.
(ii) CMCSR $\subset$ CMVSR $\subset$ CMFSR

Landscape of History Classes



Chapter 3: Concurrency Control – Notions of Correctness for the Page Model

- 3.2 Canonical Synchronization Problems
- 3.3 Syntax of Histories and Schedules
- 3.4 Correctness of Histories and Schedules
- 3.5 Herbrand Semantics of Schedules
- 3.6 Final-State Serializability
- 3.7 View Serializability
- 3.8 Conflict Serializability
- 3.9 Commit Serializability
- **3.10 An Alternative Criterion: Interleaving Specifications**
- 3.11 Lessons Learned

Interleaving Specifications: Motivation

Example: all transactions known in advance

transfer transactions on checking accounts a and b and savings account c:

$t_1 = r_1(a)\ w_1(a)\ r_1(c)\ w_1(c)$

$t_2 = r_2(b)\ w_2(b)\ r_2(c)\ w_2(c)$

balance transaction:

$t_3 = r_3(a)\ r_3(b)\ r_3(c)$

audit transaction:

$t_4 = r_4(a)\ r_4(b)\ r_4(c)\ w_4(z)$

Possible schedules:

$$\left.\begin{array}{ll} r_1(a)\ w_1(a)\ r_2(b)\ w_2(b)\ r_2(c)\ w_2(c)\ r_1(c)\ w_1(c) & \rightarrow \in \text{CSR} \\ r_1(a)\ w_1(a)\ r_3(a)\ r_3(b)\ r_3(c)\ r_1(c)\ w_1(c) & \rightarrow \notin \text{CSR} \end{array}\right\} \text{application-tolerable interleavings}$$

$$\left.\begin{array}{ll} r_1(a)\ w_1(a)\ r_2(b)\ w_2(b)\ r_1(c)\ r_2(c)\ w_2(c)\ w_1(c) & \rightarrow \notin \text{CSR} \\ r_1(a)\ w_1(a)\ r_4(a)\ r_4(b)\ r_4(c)\ w_4(z)\ r_1(c)\ w_1(c) & \rightarrow \notin \text{CSR} \end{array}\right\} \text{non-admissable interleavings}$$

Interleaving Specifications: Motivation

Example: all transactions known in advance

transfer transactions on checking accounts a and b and savings account c:

$t_1 = r_1(a)\ w_1(a)\ r_1(c)\ w_1(c)$

$t_2 = r_2(b)\ w_2(b)\ r_2(c)\ w_2(c)$

balance transaction:

$t_3 = r_3(a)\ r_3(b)\ r_3(c)$

audit transaction:

$t_4 = r_4(a)\ r_4(b)\ r_4(c)\ w_4(z)$

Possible schedules:

$$\left.\begin{array}{ll} r_1(a)\ w_1(a)\ r_2(b)\ w_2(b)\ r_2(c)\ w_2(c)\ r_1(c)\ w_1(c) & \rightarrow \in \text{CSR} \\ r_1(a)\ w_1(a)\ r_3(a)\ r_3(b)\ r_3(c)\ r_1(c)\ w_1(c) & \rightarrow \notin \text{CSR} \end{array}\right\} \text{application-tolerable interleavings}$$

$$\left.\begin{array}{ll} r_1(a)\ w_1(a)\ r_2(b)\ w_2(b)\ r_1(c)\ r_2(c)\ w_2(c)\ w_1(c) & \rightarrow \notin \text{CSR} \\ r_1(a)\ w_1(a)\ r_4(a)\ r_4(b)\ r_4(c)\ w_4(z)\ r_1(c)\ w_1(c) & \rightarrow \notin \text{CSR} \end{array}\right\} \text{non-admissable interleavings}$$

Observations: application may tolerate non-CSR schedules

a priori knowledge of all transactions impractical

Indivisible Units

Definition 3.22 (Indivisible Units):
Let T={t_1, ..., t_n} be a set of transactions. For t_i, $t_j \in$ T, $t_i \neq t_j$, an **indivisible unit of t_i relative to t_j** is a sequence of consecutive steps of t_i s.t. no operations of t_j are allowed to interleave with this sequence.
IU(t_i, t_j) denotes the ordered sequence of indivisible units of t_i relative to t_j.
IU_k(t_i, t_j) denotes the k^{th} element of IU(t_i, t_j).

Indivisible Units

Definition 3.22 (Indivisible Units):
Let $T=\{t_1, ..., t_n\}$ be a set of transactions. For $t_i, t_j \in T$, $t_i \neq t_j$, an **indivisible unit of t_i relative to t_j** is a sequence of consecutive steps of t_i s.t. no operations of t_j are allowed to interleave with this sequence.
$IU(t_i, t_j)$ denotes the ordered sequence of indivisible units of t_i relative to t_j.
$IU_k(t_i, t_j)$ denotes the k^{th} element of $IU(t_i, t_j)$.

Example 3.14:

$t_1 = r_1(x)\ w_1(x)\ w_1(z)\ r_1(y)$
$t_2 = r_2(y)\ w_2(y)\ r_2(x)$
$t_3 = w_3(x)\ w_3(y)\ w_3(z)$

$IU(t_1, t_2) = <[r_1(x)\ w_1(x)], [\ w_1(z)\ r_1(y)]>$
$IU(t_1, t_3) = <[r_1(x)\ w_1(x)], [\ w_1(z)], [\ r_1(y)]>$
$IU(t_2, t_1) = <[r_2(y)], [\ w_2(y)\ r_2(x)]>$
$IU(t_2, t_3) = <[r_2(y)\ w_2(y)], [\ r_2(x)]>$
$IU(t_3, t_1) = <[w_3(x)\ w_3(y)], [w\ _3(z)]>$
$IU(t_3, t_2) = <[w_3(x)\ w_3(y)], [w\ _3(z)]>$

Indivisible Units

Definition 3.22 (Indivisible Units):
Let T={t_1, ..., t_n} be a set of transactions. For t_i, $t_j \in$ T, $t_i \neq t_j$, an **indivisible unit of t_i relative to t_j** is a sequence of consecutive steps of t_i s.t. no operations of t_j are allowed to interleave with this sequence.
IU(t_i, t_j) denotes the ordered sequence of indivisible units of t_i relative to t_j.
$IU_k(t_i, t_j)$ denotes the kth element of IU(t_i, t_j).

Example 3.14:

$t_1 = r_1(x)\ w_1(x)\ w_1(z)\ r_1(y)$
$t_2 = r_2(y)\ w_2(y)\ r_2(x)$
$t_3 = w_3(x)\ w_3(y)\ w_3(z)$

$IU(t_1, t_2) = < [r_1(x)\ w_1(x)], [\ w_1(z)\ r_1(y)] >$
$IU(t_1, t_3) = < [r_1(x)\ w_1(x)], [\ w_1(z)], [\ r_1(y)] >$
$IU(t_2, t_1) = < [r_2(y)], [\ w_2(y)\ r_2(x)] >$
$IU(t_2, t_3) = < [r_2(y)\ w_2(y)], [\ r_2(x)] >$
$IU(t_3, t_1) = < [w_3(x)\ w_3(y)], [w\ _3(z)] >$
$IU(t_3, t_2) = < [w_3(x)\ w_3(y)], [w\ _3(z)] >$

Example 3.15:

$s_1 = r_2(y)\ r_1(x)\ w_1(x)\ w_2(y)\ r_2(x)\ w_1(z)\ w_3(x)\ w_3(y)\ r_1(y)\ w_3(z)$ → respects all IUs

$s_2 = r_1(x)\ r_2(y)\ w_2(y)\ w_1(x)\ r_2(x)\ w_1(z)\ r_1(y)$ → violates $IU_1(t_1, t_2)$ and $IU_2(t_2, t_1)$
but is conflict equivalent to
an allowed schedule

Relatively Serializable Schedules

Definition 3.23 (Dependence of Steps):
Step q directly **depends on** step p in schedule s, denoted p~>q, if $p <_s q$ and either p, q belong to the same transaction t and $p <_t q$ or p and q are in conflict.
~>* denotes the reflexive and transitive closure of ~>.

Relatively Serializable Schedules

Definition 3.23 (Dependence of Steps):
Step q directly **depends on** step p in schedule s, denoted p~>q, if p $<_s$ q and either p, q belong to the same transaction t and p $<_t$ q or p and q are in conflict. ~>* denotes the reflexive and transitive closure of ~>.

Definition 3.24 (Relatively Serial Schedule):
s is **relatively serial** if for all transactions t_i, t_j: if q $\in$ t_j is interleaved with some $IU_k(t_i, t_j)$, then there is no operation p $\in$ $IU_k(t_i, t_j)$ s.t. p~>* q or q~>* p

Example 3.16:
$s_3 = r_1(x)\ r_2(y)\ w_1(x)\ w_2(y)\ w_3(x)\ w_1(z)\ w_3(y)\ r_2(x)\ r_1(y)\ w_3(z)$

Relatively Serializable Schedules

Definition 3.23 (Dependence of Steps):
Step q directly **depends on** step p in schedule s, denoted p~>q, if p $<_s$ q and either p, q belong to the same transaction t and p $<_t$ q or p and q are in conflict.
~>* denotes the reflexive and transitive closure of ~>.

Definition 3.24 (Relatively Serial Schedule):
s is **relatively serial** if for all transactions t_i, t_j: if q $\in$ t_j is interleaved with some $IU_k(t_i, t_j)$, then there is no operation p $\in$ $IU_k(t_i, t_j)$ s.t. p~>* q or q~>* p

Example 3.16:
$s_3 = r_1(x)\ r_2(y)\ w_1(x)\ w_2(y)\ w_3(x)\ w_1(z)\ w_3(y)\ r_2(x)\ r_1(y)\ w_3(z)$

Definition 3.25 (Relatively Serializable Schedule):
s is **relatively serializable** if it is conflict equivalent to a relatively serial schedule.

Example 3.17:
$s_4 = r_1(x)\ r_2(y)\ w_2(y)\ w_1(x)\ w_3(x)\ r_2(x)\ w_1(z)\ w_3(y)\ r_1(y)\ w_3(z)$

Relative Serialization Graph

Definition 3.26 (Push Forward and Pull Backward):
Let $IU_k(t_i, t_j)$ be an IU of t_i relative to t_j. For an operation $p_i \in IU_k(t_i, t_j)$ let
(i) $F(p_i, t_j)$ be the last operation in $IU_k(t_i, t_j)$ and
(ii) $B(p_i, t_j)$ be the first operation in $IU_k(t_i, t_j)$.

Definition 3.27 (Relative Serialization Graph):
The **relative serialization graph RSG(s)** = (V,E) of schedule s is a graph with vertices $V := op(s)$ and edge set $E \subseteq V \times V$ containing four types of edges:
(i) for consecutive operations p, q of the same transaction $(p, q) \in E$ *(I-edge)*
(ii) if $p \sim>^* q$ for $p \in t_i$, $q \in t_j$, $t_i \neq t_j$, then $(p, q) \in E$ *(D-edge)*
(iii) if (p, q) is a D-edge with $p \in t_i$, $q \in t_j$, then $(F(p, t_j), q) \in E$ *(F-edge)*
(iv) if (p,q) is a D-edge with $p \in t_i$, $q \in t_j$, then $(p, B(q, t_i)) \in E$ *(B-edge)*

Theorem 3.18:
A schedule s is relatively serializable iff RSG(s) is acyclic.

RSG Example

Example 3.19:

$t_1 = w_1(x)\ r_1(z)$

$t_2 = r_2(x)\ w_2(y)$

$t_3 = r_3(z)\ r_3(y)$

$IU(t_1, t_2) = <[w_1(x)\ r_1(z)]>$

$IU(t_1, t_3) = <[w_1(x)], [\ r_1(z)]>$

$IU(t_2, t_1) = <[r_2(x)], [\ w_2(y)]>$

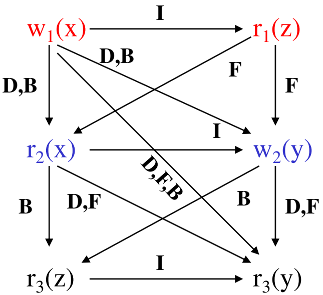
$IU(t_2, t_3) = <[r_2(x)], [\ w_2(y)]>$

$IU(t_3, t_1) = <[r_3(z)], [r\ _3(y)]>$

$IU(t_3, t_2) = <[r_3(z)\ r_3(y)]>$

$s_5 = w_1(x)\ r_2(x)\ r_3(z)\ w_2(y)\ r_3(y)\ r_1(z)$

RSG(s_5):



Chapter 3: Concurrency Control – Notions of Correctness for the Page Model

- 3.2 Canonical Synchronization Problems
- 3.3 Syntax of Histories and Schedules
- 3.4 Correctness of Histories and Schedules
- 3.5 Herbrand Semantics of Schedules
- 3.6 Final-State Serializability
- 3.7 View Serializability
- 3.8 Conflict Serializability
- 3.9 Commit Serializability
- 3.10 An Alternative Criterion: Interleaving Specifications
- **3.11 Lessons Learned**

Lessons Learned

- Equivalence to serial history is a natural correctness criterion
- CSR, albeit less general than VSR,

  is most appropriate for

  - complexity reasons
  - its monotonicity property
  - its generalizability to semantically rich operations
- OCSR and COCSR have additional beneficial properties