

Chapter 5

Isolation: Serializability

Agenda

- Conflict serializability
- Order preservation
- View serializability
- Final-state serializability
- Commit serializability

Conflict serializability

Conflict equivalence

Remember: We study histories, and we ignore aborted transactions.

Definition 5.1 (Conflict equivalence)

(Def. 4.23 specialized)

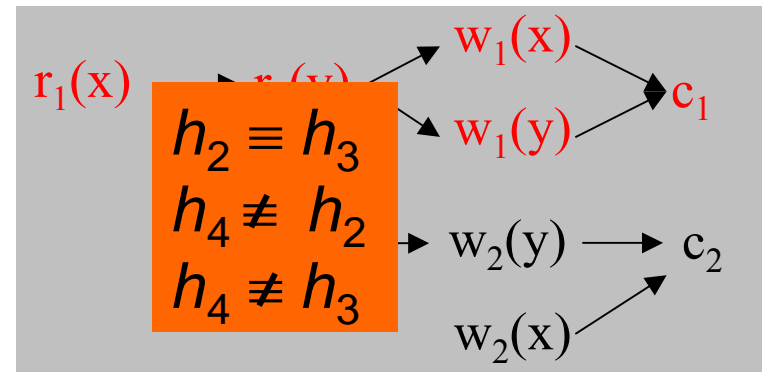
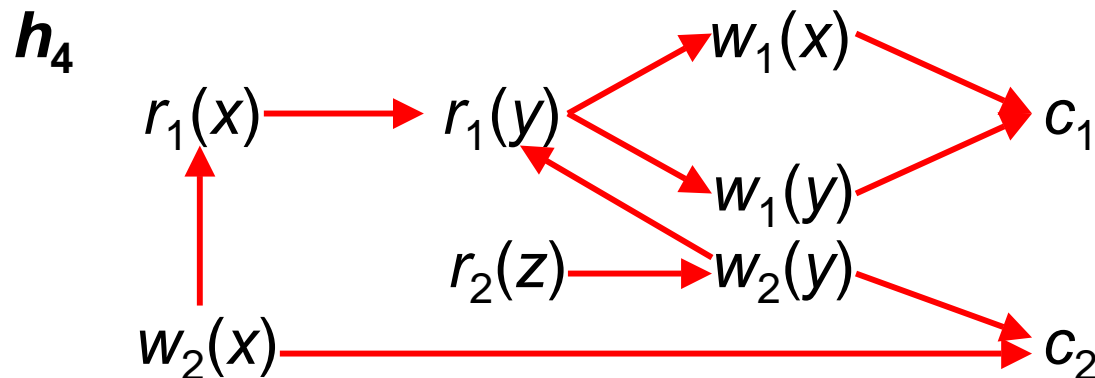
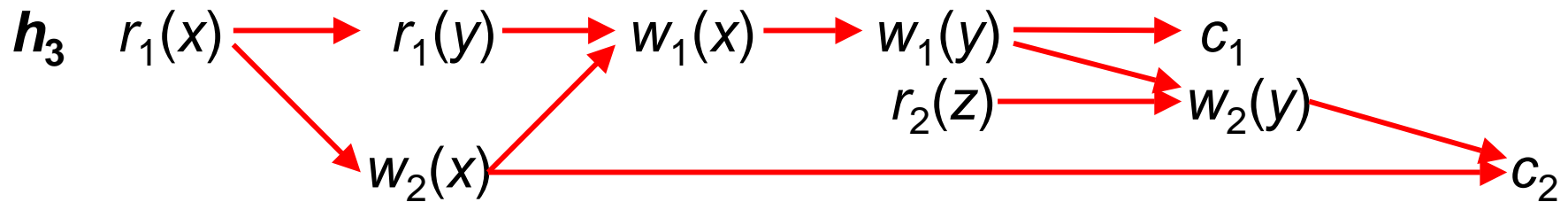
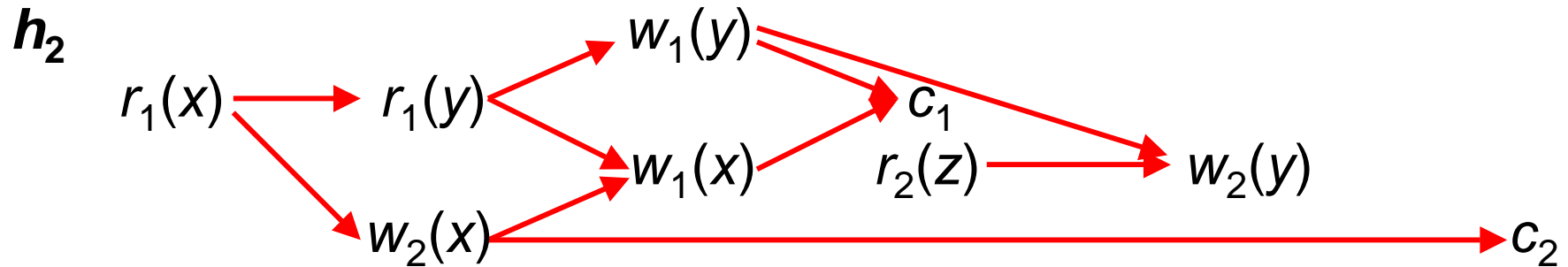
Two histories h and h' are **conflict equivalent** ($h \equiv_C h'$)

\Leftrightarrow

- $op(h) = op(h')$
- $conf(h) = conf(h')$

Conflict equivalence

Example 5.2 Histories over two transactions:



Conflict serializability

Definition 5.3

- **Conflict serializability**: A history h is conflict serializable if there exists a conflict equivalent serial history h_s : $\exists h_s$ serial: $CP(h) \equiv_C h_s$
- A schedule s is conflict serializable if there exists a serial history h_s s.t. $\exists h_s$ serial: $CP(s) \equiv_C h_s$

Corollary 5.4

- A history h is **conflict serializable** if $\exists h_s$ serial: $CP(h) \equiv_C h_s \Leftrightarrow conf(h) = conf(h_s)$.

Remember: $conf(s)$ ignores aborted transactions

Definition 5.5

$CSR ::=$ Set of all conflict serializable histories.

Proving conflict serializability

- **Idea:** Find a good characterization for $conf(h_s)$. Then, for h serializable the same characterization should hold for $conf(h)$.
- **Needed:** Efficient proof procedure.
- **Standard approach:** Try a graph.

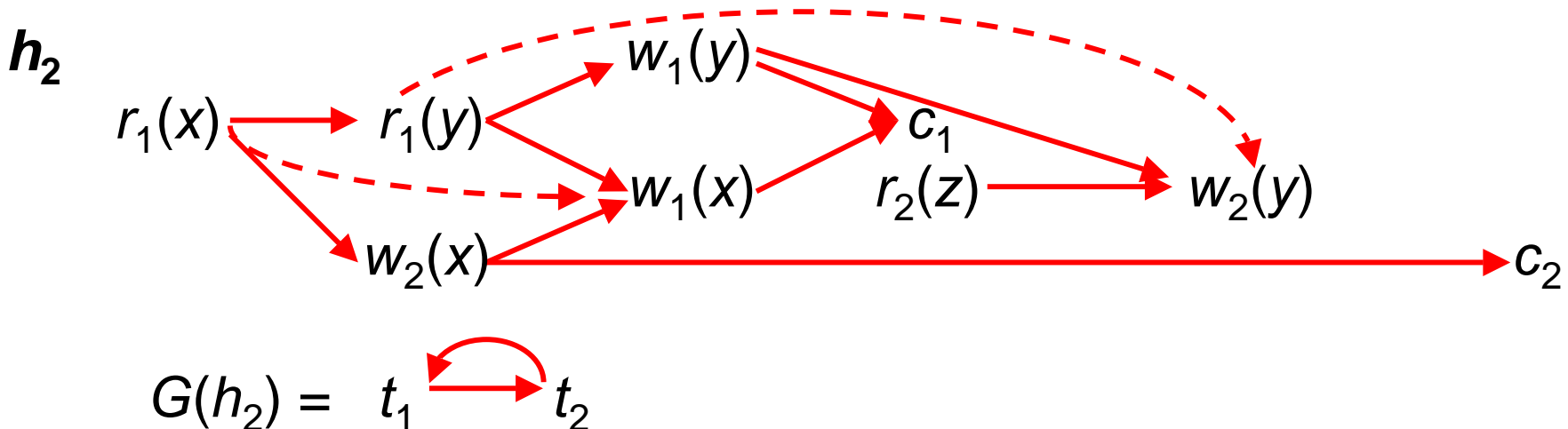
Conflict graph

Definition 5.6 (Conflict Graph):

Let h be a history. The **conflict graph** $G(h) = (V, E)$ is a directed graph with

vertices $V := \text{commit}(h)$ and

edges $E := \{(t, t') \mid t \neq t' \text{ and there are operations } p \in t, q \in t' \text{ with } (p, q) \in \text{conf}(h)\}$.



Conflict graph

Definition 5.6 (Conflict Graph):

Let h be a history. The **conflict graph** $G(h) = (V, E)$ is a directed graph with

vertices $V := \text{commit}(s)$ and

edges $E := \{(t, t') \mid t \neq t'$

and there are operations $p \in t, q \in t'$ with $(p, q) \in \text{conf}(h)\}$.

- An edge $t \rightarrow t'$ indicates that *at least one* operation in t precedes an operation in t' and the two are in conflict.
- In a serial history: All operations in t precede those in t' .
 - ◆ $G(h_s)$, h_s serial, is acyclic.
 - Failure if $G(h)$ contains a cycle.

Correctness proof

From intuition to proof:

Theorem 5.7:

Let h be a history. Then $h \in CSR \Leftrightarrow G(h)$ is acyclic.

Correctness proof

(i) $h \in \text{CSR} \Leftrightarrow G(h)$ is acyclic

Let h be a history in CSR. So there is a serial history h' with $\text{conf}(h) = \text{conf}(h')$.

Now assume that $G(h)$ has a cycle $t_1 \rightarrow t_2 \rightarrow \dots \rightarrow t_k \rightarrow t_1$.

This implies that there are pairs $(p_1, q_2), (p_2, q_3), \dots, (p_k, q_1)$

with $p_i \in t_i, q_i \in t_i, p_i <_h q_{(i+1)}$, and p_i in conflict with $q_{(i+1)}$.

Because $h' \equiv_C h$, it also implies that $p_i <_{h'} q_{(i+1)}$.

Because h' is serial, we obtain $t_i <_{h'} t_{(i+1)}$ for $i=1, \dots, k-1$, and $t_k <_{h'} t_1$.

By transitivity we infer $t_1 <_{h'} t_2$ and $t_2 <_{h'} t_1$, which is impossible.

The initial assumption is wrong. So $G(h)$ is acyclic.

(ii) $h \in \text{CSR} \Leftrightarrow G(h)$ is acyclic

Let $G(h)$ be acyclic. So it must have at least one source node.

The following topological sort produces a total order $<$ of transactions:

- start with a source node (i.e., a node without incoming edges),
- remove this node and all its outgoing edges,
- iterate a) and b) until all nodes have been added to the sorted list.

The total transaction ordering order $<$ preserves the edges in $G(h)$;

therefore it yields a serial schedule h' for which $h' \equiv_C h$.

Efficient correctness proof

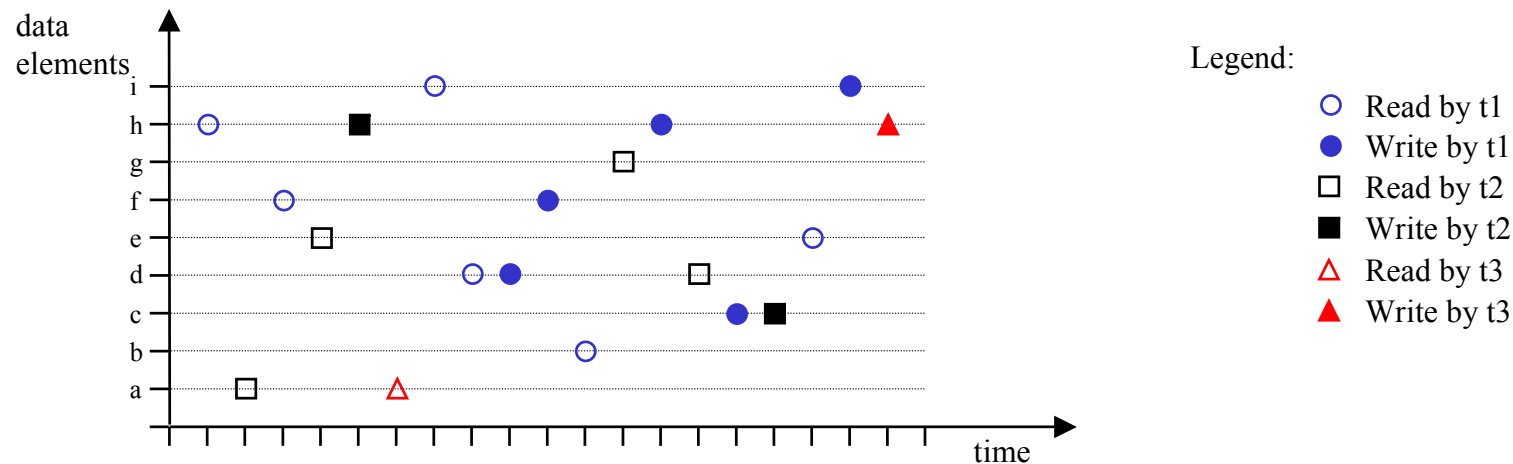
12

- From proof (ii) we conclude that if for a history h , $G(h)$ is acyclic, h is equivalent to every serial history that is a topological order of $G(h)$.
- Since in general there are several such orders, there may be several serial histories that are equivalent to h .
- Conflict serializability can be computed in polynomial time – in fact in square time because of the simple test on acyclicity.

Efficient correctness proof: Illustration

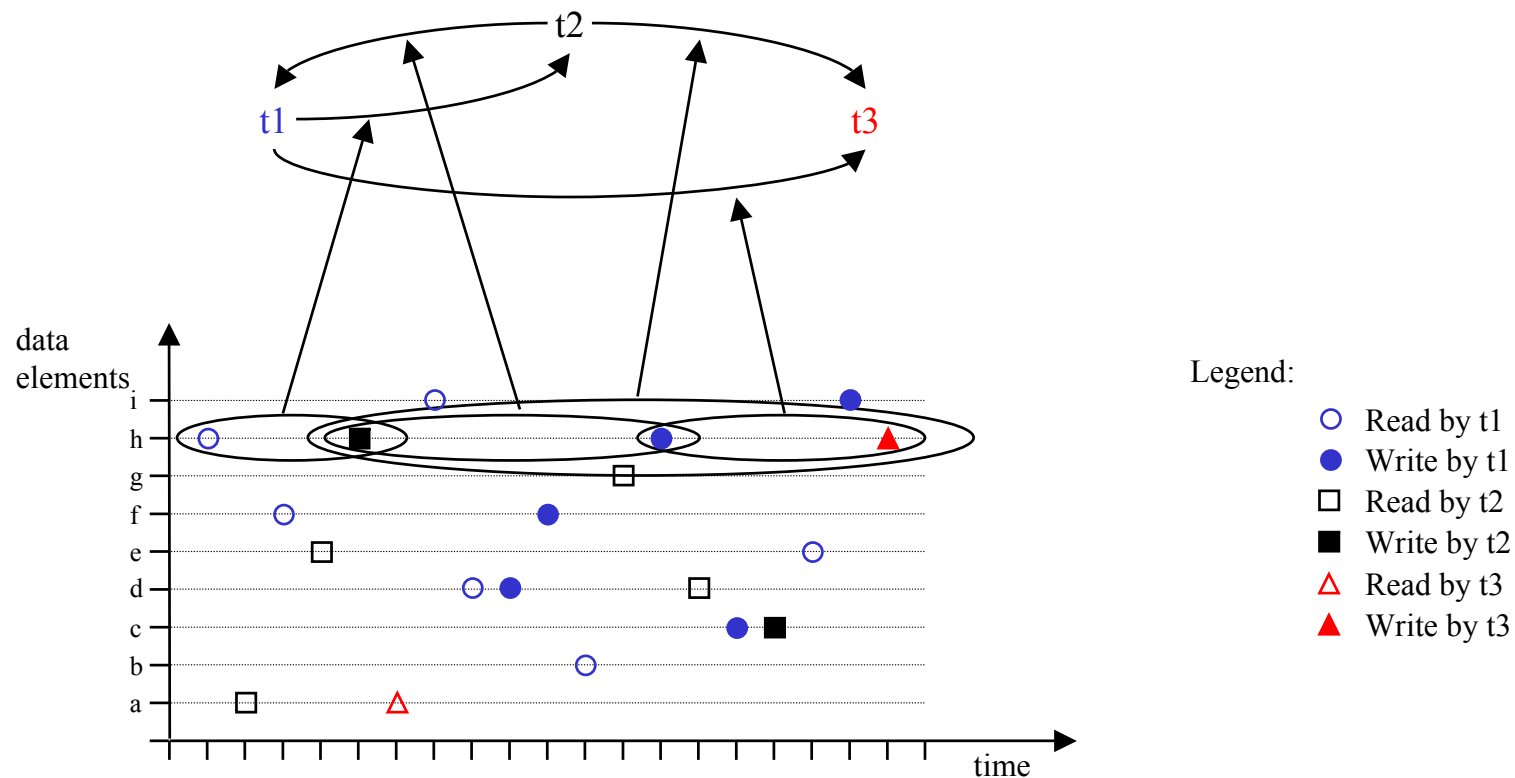
- **Example 5.8:** Take earlier history that was view serializable but not conflict serializable:

$r_1(h)$ $r_2(a)$ $r_1(f)$ $r_2(e)$ $w_2(h)$ $r_3(a)$ $r_1(i)$ $r_1(d)$ $w_1(d)$ $w_1(f)$ $r_1(b)$
 $r_2(g)$ $w_1(h)$ $r_2(d)$ $w_1(c)$ $w_2(c)$ $r_1(e)$ $w_1(i)$ c_1 $w_3(h)$ c_2 c_3 .



Efficient correctness proof: Illustration

■ Example 5.8 cont'd: Conflict graph:



Efficient correctness proof: Illustration

Example 5.9: Histories in the booking example:

h1: $r_1(B) r_2(B) r_2(T) w_2(T) w_2(B) c_2 r_1(T) c_1$ ⚡

h2: $r_2(B) r_2(T) r_1(B) r_1(T) c_1 w_2(T) w_2(B) c_2$ ✓

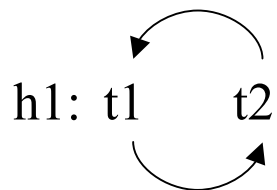
h3: $r_2(B) r_2(T) w_2(T) r_1(B) r_1(T) c_1 w_2(B) c_2$ ⚡

h4: $r_2(B) r_2(T) w_2(T) w_2(B) r_1(B) r_1(T) a_2 c_1$

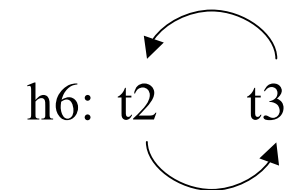
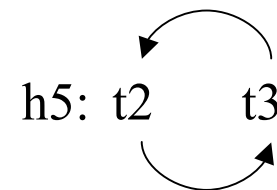
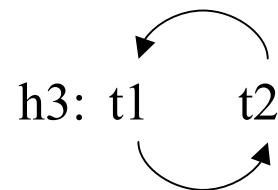
only t1 to consider!

h5: $r_3(T) w_3(T) r_2(B) r_2(T) w_2(T) w_2(B) c_2 r_3(B) w_3(B) c_3$ ⚡

h6: $r_2(B) r_2(T) r_3(T) w_3(T) r_3(B) w_3(B) c_3 w_2(T) w_2(B) c_2$ ⚡



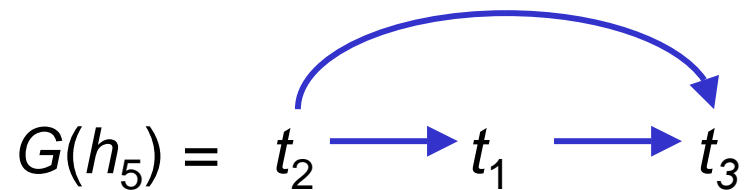
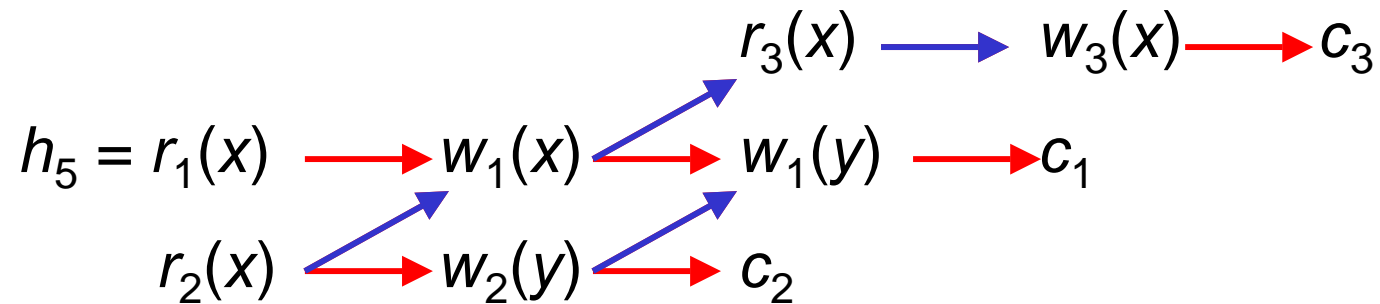
h2: t1 → t2



Efficient correctness proof: Illustration

16

Example 5.10:

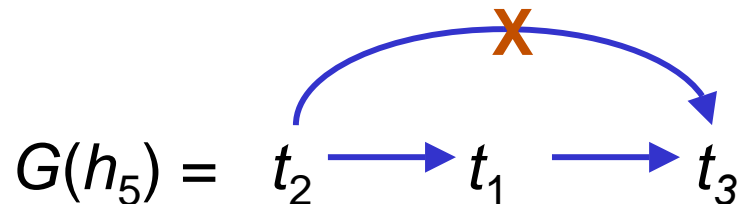
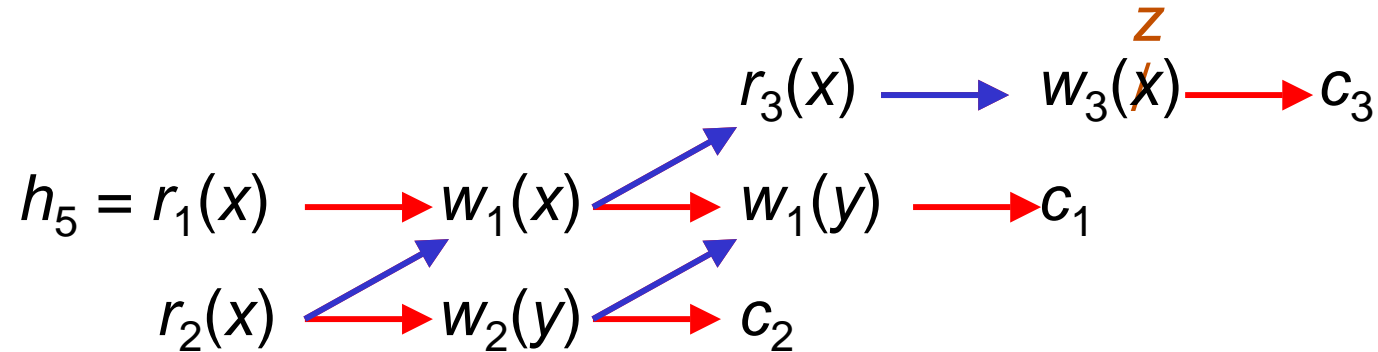


Equivalent serial history: t_2, t_1, t_3

Efficient correctness proof: Illustration

17

Example 5.10 modified:



Equivalent serial history: t_2, t_1, t_3

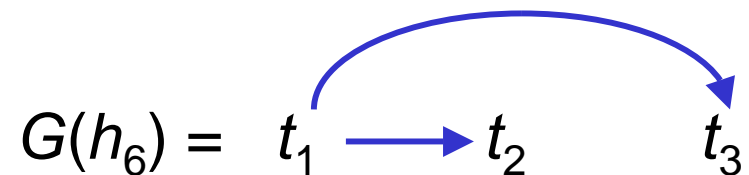
Note: In general $t_i \rightarrow t_j, t_j \rightarrow t_k \not\Rightarrow t_i \rightarrow t_k$.

Efficient correctness proof: Illustration

18

Example 5.11:

$$h_6 = w_1(x) w_1(y) c_1 r_2(x) r_3(y) w_2(x) c_2 w_3(y) c_3$$



Equivalent serial histories:

t_1, t_2, t_3

t_1, t_3, t_2

Serialization by reordering

Remember our examples with serialization by reordering of operations. **Is there a relationship between CSR and reordering?** Formalize:

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 $\sim o_i(x) p_j(y)$
if $x \neq y$ or both o and p are reads

Serialization by reordering

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 $\sim o_i(x) p_j(y)$
if $x \neq y$ or both o and p are reads

Example for transformations of schedules:

$$\begin{aligned}
 s &= w_1(x) \underbrace{r_2(x) w_1(y) w_1(z)}_{\text{C2}} \underbrace{r_3(z) w_2(y)}_{\text{C2}} w_3(y) w_3(z) \\
 \sim (C2) & w_1(x) w_1(y) \underbrace{r_2(x) w_1(z) w_2(y)}_{\text{C2}} r_3(z) w_3(y) w_3(z) \\
 \sim (C2) & w_1(x) w_1(y) w_1(z) r_2(x) w_2(y) r_3(z) w_3(y) w_3(z) \\
 &= t_1 t_2 t_3
 \end{aligned}$$

$r_2(x)$ and $r_3(z)$
can be delayed

Commutativity-based reducibility

21

Definition 5.12 (Commutativity Based Equivalence):

Schedules s and s' s.t. $op(s)=op(s')$ are **commutativity based equivalent**, denoted $s \sim^* s'$, if s can be transformed into s' by applying rules C1, C2, C3, C4 finitely many times.

Theorem 5.13:

Let s and s' be schedules s.t. $op(s)=op(s')$. Then $s \equiv_c s'$ iff $s \sim^* s'$.

Definition 5.14 (Commutativity Based Reducibility):

Schedule s is **commutativity-based reducible** if there is a serial schedule s' s.t. $s \sim^* s'$.

Corollary 5.15:

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

© Weikum, Vossen, 2002

Commutativity-based reducibility

Practical value?

- Scheduler may control the sequence of operations
 - ◆ solely based on observing/recognizing conflicts
- Just tell the scheduler which conflicts can occur
 - ◆ It does not need to know any further operator details

Order preservation

Order preserving conflict serializability

Example 5.16

Take history

$$h = w_1(x) r_2(x) c_2 w_3(y) c_3 w_1(y) c_1$$

$$G(h) = t_3 \longrightarrow t_1 \longrightarrow t_2$$

Unpleasant because t_2 was committed by the time t_3 starts.

Imagine we keep a chronological record of the transactions and for some reason would have to redo the transactions!

Order preserving conflict serializability

Definition 5.17

History h (or the committed projection $h=CP(s)$ of schedule s) is **order preserving conflict serializable** if h is conflict equivalent to a serial history h_s where

$t, t' \in h$:

if t occurs completely before t' in h

then the same holds in h_s

Definition 5.18

OCSR ::= Set of all order preserving conflict serializable histories.

⇒ In OCSR histories, transactions cannot commute if they do not overlap in time

Order preserving conflict serializability

Corollary 5.19

$$OCSR \subset CSR$$

by Def. 5.17. See example 5.16 for proper inclusion :

$G(h) \notin OCSR$, because $t_3 t_1 t_2$ does not preserve the order of the non-overlapping transactions t_2, t_3 .

Commit-Order Preservation (1)

27

Definition 5.20 (Commit-Order Preservation)

History h is **commit-ordered** if:

$\forall t_i, t_j \in h, i \neq j, p \in op_i, q \in op_j: (p, q) \in conf(h) \succ c_i <_h c_j$

Definition 5.21

$COCSR ::=$ Set of all commit-ordered histories.

⇒ The order of two operations that are in conflict agrees with the order of the commit operations of their respective transactions.

Commit-Order Preservation (2)

Theorem 5.22

$$COCSR \subset CSR$$

Proof:

Let $h \in COCSR$ und $(t_i, t_j) \in G(h)$.

(definition $COCSR$) $\succ c_i <_h c_j$.

By induction: for all paths $(t_1, \dots, t_n) \in G(h)$:

$$c_i <_h c_{i+1} \text{ for } 1 \neq i < n.$$

$COCSR \subset CSR$: Let $h \notin CSR \succ G(h)$ cyclic. Let (t_1, \dots, t_n, t_1) be a cycle. $\succ c_1 <_h c_1$ (contradiction).

$CSR \not\subset COCSR$: $h = r_1(x) w_2(x) c_2 c_1$ is CSR but not $COCSR$.

Commit-Order Preservation (3)

Theorem 5.23

$COCSR \subset OCSR$

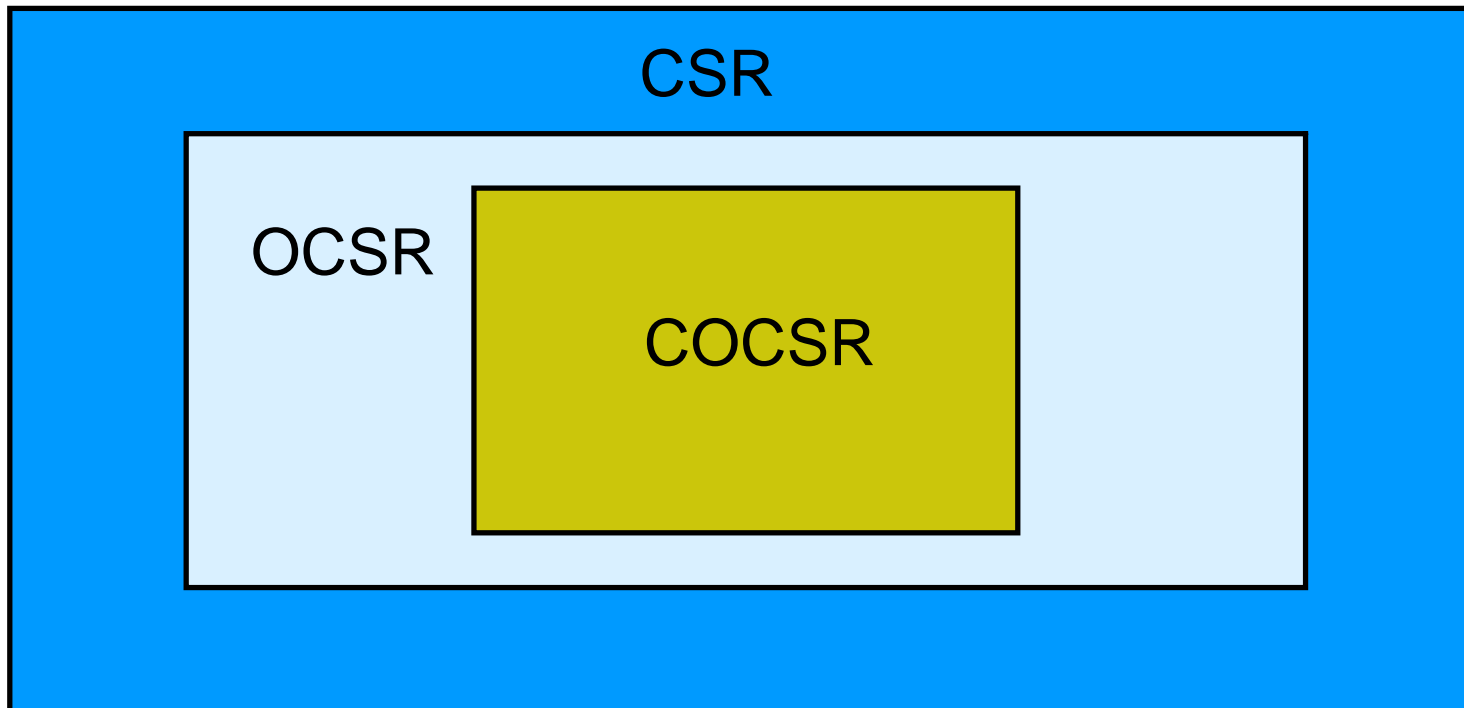
Example 5.24

- $h_1 = w_1(x) r_2(x) c_2 w_3(y) c_3 w_1(y) c_1$ CSR, not OCSR, COCSR
- $h_2 = w_3(y) c_3 w_1(x) r_2(x) c_2 w_1(y) c_1$ OCSR, not COCSR
- $h_3 = w_3(y) c_3 w_1(x) r_2(x) w_1(y) c_1 c_2$ COCSR

$$G(h) = t_3 \longrightarrow t_1 \longrightarrow t_2$$

CSR restrictions

Summary



View serializability

View equivalence (1)

Remember:

Definition 5.25

Two histories h and h' are **view equivalent** ($h \equiv_v h'$): $\Leftarrow \Leftarrow$

- $op(h) = op(h')$
- $RF(h) = RF(h')$
- $\forall x \text{ FIN}_h(x) = \text{FIN}_{h'}(x)$

$$RF(h) := \{(t_i, x, t_j) \mid t_j \triangleright_h(x) t_i\}$$

where $t_j \triangleright_h(x) t_i$ (“ t_j reads x from t_i “):

- ◆ $w_i(x) <_h r_j(x), a_i \not<_h r_j(x)$
- ◆ $\forall k \neq i, j \ w_i(x) <_h w_k(x) <_h r_j(x) \Rightarrow a_k <_h r_j(x)$

View serializability (1)

Remember:

Definition 5.26

A history h is **view serializable** if there exists a view equivalent serial history h_s : $\exists h_s \text{ serial: } CP(h) \equiv_V h_s$

Definition 5.27

$VSR ::=$ Set of all view serializable histories.

$RF(h)$ can be simplified for $CP(h)$:

$t_j \triangleright_{CP(h)}(x) t_i$:

◆ $w_i(x) <_{CP(h)} r_j(x)$ and $\nexists w_k(x) w_i(x) <_{CP(h)} w_k(x) <_{CP(h)} r_j(x)$

Sichtserialisierbarkeit

34

Satz 5.28 $CSR \subset VSR$

Beweis:

$h \in CSR \succ \exists h_s$ seriell $CP(h) \equiv_C h_s$

Zu zeigen: $CP(h) \equiv_V h_s$.

Zwei Historien h und h' sind sichtäquivalent ($h \equiv_V h'$): \Leftrightarrow

- $op(h) = op(h')$
- $RF(h) = RF(h')$
- $\forall x \text{ FIN}_h(x) = \text{FIN}_{h'}(x)$

Sichtserialisierbarkeit

Satz 5.28 *CSR*: Da in *CSR* für alle i, j $w_i(x) \not\parallel w_j(x)$ gilt und beide Historien $CP(h)$ und h_s unverträgliche Operationen in der gleichen Reihenfolge ordnen, sind ihre letzten Schreiboperationen identisch.

Beweis:

$h \in CSR \succ \exists h_s$

Zu zeigen: $CP(h) \equiv_V h_s$.

$$\begin{aligned}
 & t_i \triangleright_{CP(h)}(x) t_j \\
 & \succ w_j(x) <_{CP(h)} r_i(x) \text{ und} \\
 & \quad \exists w_k(x) w_j(x) <_{CP(h)} w_k(x) <_{CP(h)} r_i(x) \\
 & \succ_{(CSR)} w_j(x) <_{h_s} r_i(x) \text{ und} \\
 & \quad \exists w_k(x) w_j(x) <_{h_s} w_k(x) <_{h_s} r_i(x) \\
 & \succ t_i \triangleright_{h_s}(x) t_j
 \end{aligned}$$

Zwei Historien h und h' sind

- $op(h) = op(h')$ ✓
- $RF(h) = RF(h')$
- $\forall x \text{ FIN}_h(x) = \text{FIN}_{h'}(x)$

Echtheit der Teilmengenbeziehung folgt aus früherem Beispiel in Kapitel 4!

View equivalence (2)

How to use solely *RF*:

Definition 5.29 (History completion)

Let h be a history over $T = \{t_1, \dots, t_n\}$. We complete h to history \hat{h} over $t \cup \{t_0, t_\infty\}$:

- t_0 is fictitious first transaction initializing all database elements x_1, \dots, x_k that are referenced by h :

$$t_0 = w_0(x_1) \dots w_0(x_k) c_0,$$

- t_∞ is fictitious last transaction with operations $r_\infty(x)$ for all data elements x appearing in h :

$$\forall p \in h, x \in h \quad p <_h r_\infty(x)$$

View serializability (2)

Definition 5.30

Two histories h und h' are **view equivalent** ($h \equiv_v h'$): $\Leftarrow \Rightarrow$

- $RF(\hat{h})=RF(\hat{h}')$

Definition 5.31

- A history h is **view serializable** if there exists a view equivalent serial history h_s :

$$\exists h_s \text{ serial: } CP(h) \equiv_v h_s \Leftarrow \Rightarrow RF(\hat{h})=RF(\hat{h}_s)$$

Proving view serializability (1)

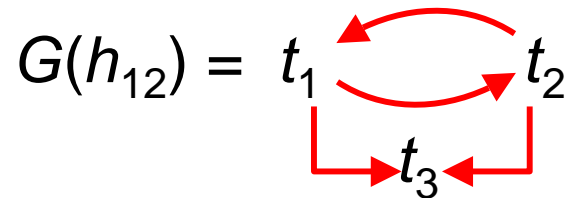
- **Idea:** Find a good characterization for $RF(h_s)$. Then, if h is serializable the same characterization should hold for $RF(h)$.
- **Needed:** Efficient proof procedure.
- **Standard approach:** Try a graph.

Proving view serializability (2)

But: Conflict graph is insufficient!

Example 5.32 Take

$$h_{12} = w_1(x) w_2(x) w_2(y) c_2 w_1(y) c_1 w_3(x) w_3(y) c_3$$



Not conflict serializable!

But view serializable: $t_1 t_2 t_3$ or $t_2 t_1 t_3$

Proving view serializability (2)

Remember:

$t_j \triangleright_{CP(h)}(x) t_i$:

- ◆ $w_i(x) <_{CP(h)} r_j(x)$ and $\nexists w_k(x) w_i(x) <_{CP(h)} w_k(x) <_{CP(h)} r_j(x)$
- The reads-from relation must not be broken up by a write operation of some other transaction. All such write operations must appear **before the write** or **after the read**.
- ⇒ We need a type of graph that reflects this condition.

Nachweis der Serialisierbarkeit (3)

Definition 5.33 (Polygraph)

Ein **Polygraph** ist ein Tripel $P = (V, E, C)$ mit

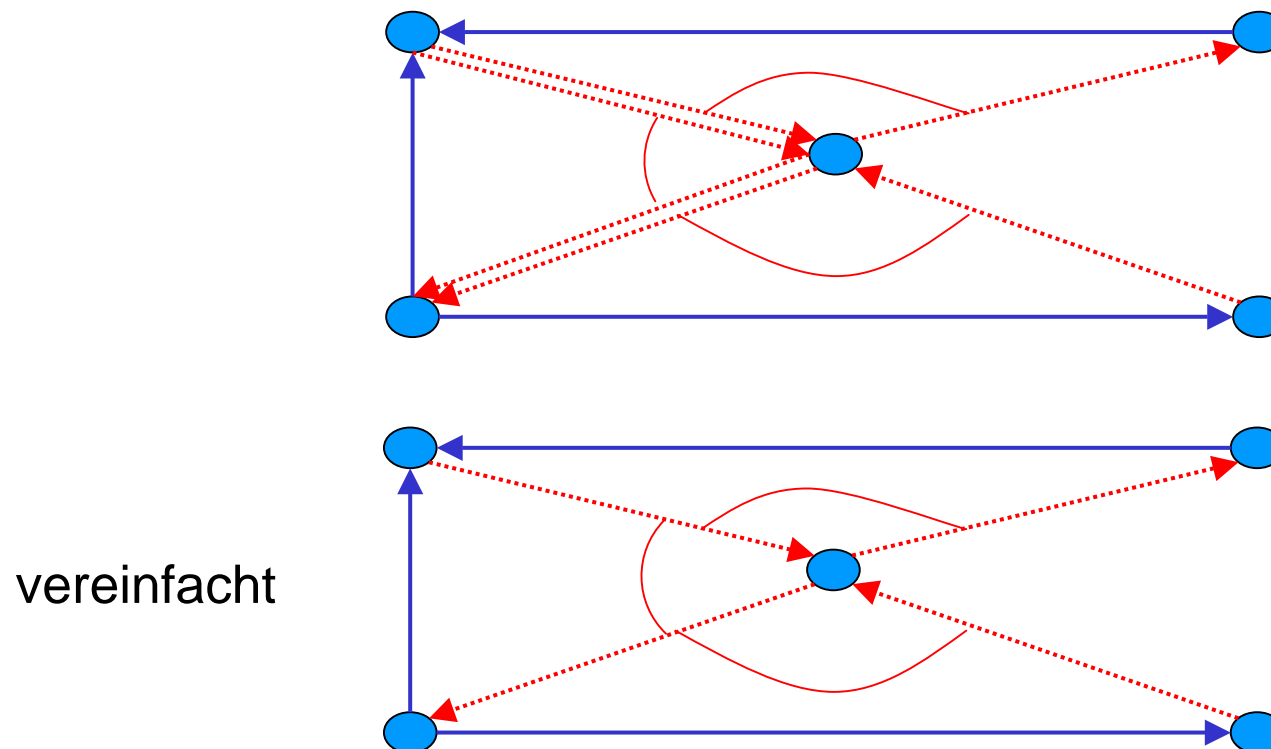
- (V, E) ist ein gerichteter Graph und
- $C \subseteq V \times V \times V$ mit $(u, v, w) \in C \Rightarrow u \neq v \neq w, (w, u) \in E$.

Veranschaulichung

42

- (V, E) ist ein gerichteter Graph und
- $C \subseteq V \times V \times V$ mit $(u, v, w) \in C \Rightarrow u \neq v \neq w, (w, u) \in E$.

Darstellung von (u, v, w) als zwei gerichtete Kanten (Bipfad) $(u, v), (v, w)$, die über Kreissegment verbunden sind.



Nachweis der Serialisierbarkeit (4)

Definition 5.34 (kompatibel)

Sei $P = (V, E, C)$ ein Polygraph und $G = (V, E')$ ein Graph mit den selben Knoten. G ist **kompatibel** zu P , wenn E' eine minimale Menge von Kanten mit den folgenden Eigenschaften ist:

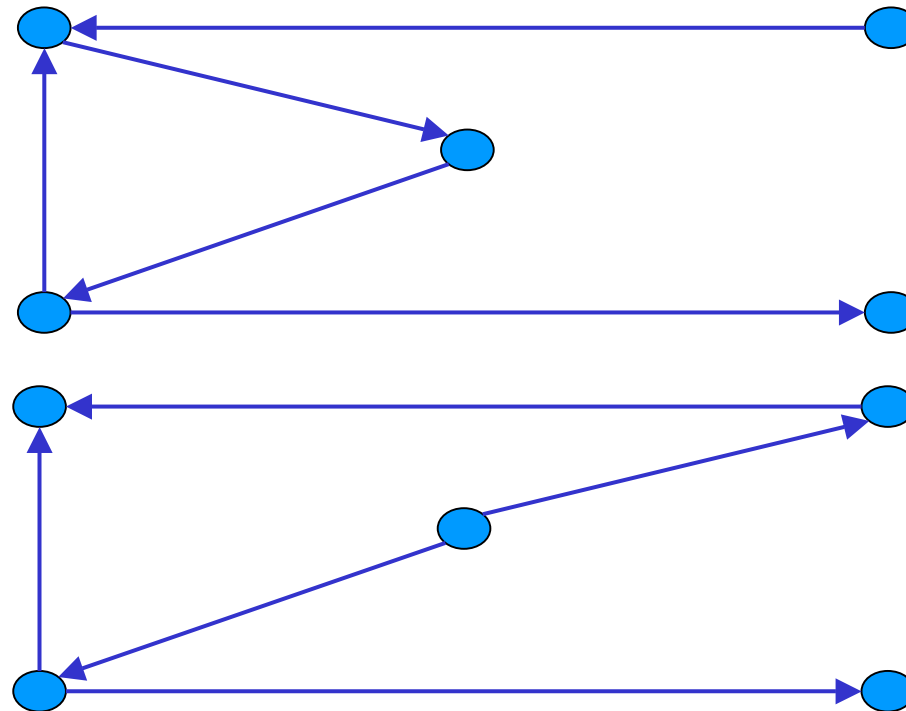
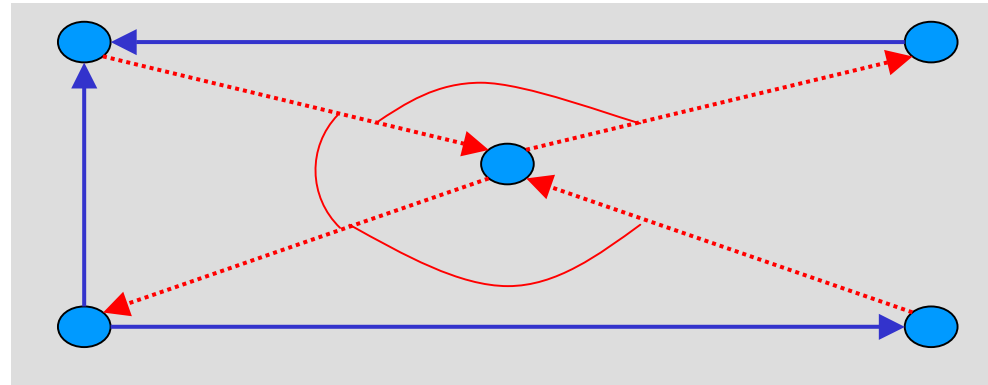
- $E \subseteq E'$
- $\forall (u, v, w) \in C: (u, v) \in E' \vee (v, w) \in E'$
- $\forall (u, v, w) \in C: (u, v) \in E' \succ (v, w) \notin E', (u, v) \notin E' \succ (v, w) \in E'$

G enthält mindestens alle Kanten aus E .

Jeder Bipfad aus P ist in G durch genau eine Kante repräsentiert

Daher: C steht für Choice \Rightarrow Ein Polygraph kann als Familie $\mathbf{D}(V, E, C)$ von gerichteten Graphen (V, E') gedeutet werden.

Beispiel zur Kompatibilität



Nachweis der Serialisierbarkeit (5)

45

Definition 5.35 ($P(h)$)

Gegeben sei eine Historie h über der Transaktionsmenge $T = \{t_1, \dots, t_n\}$. Wir definieren $P(h) = (V, E, C)$ mit

- $V = T \cup \{t_0, t_\infty\}$
- $E = \{ (t_j, t_i) \mid t_i \triangleright_{\hat{h}}(x) t_j \}$
- $C = \{ ((t_i, t_k, t_j)) \mid t_i \triangleright_{\hat{h}}(x) t_j, w_k(x) \in h \}$ mit $i \neq j \neq k$

$w_j(x) < r_i(x)$ und $\exists w_k(x) w_j(x) < w_k(x) < r_i(x)$

Eine Kante (t_j, t_i) drückt aus, dass t_j vor t_i stattfinden muss.

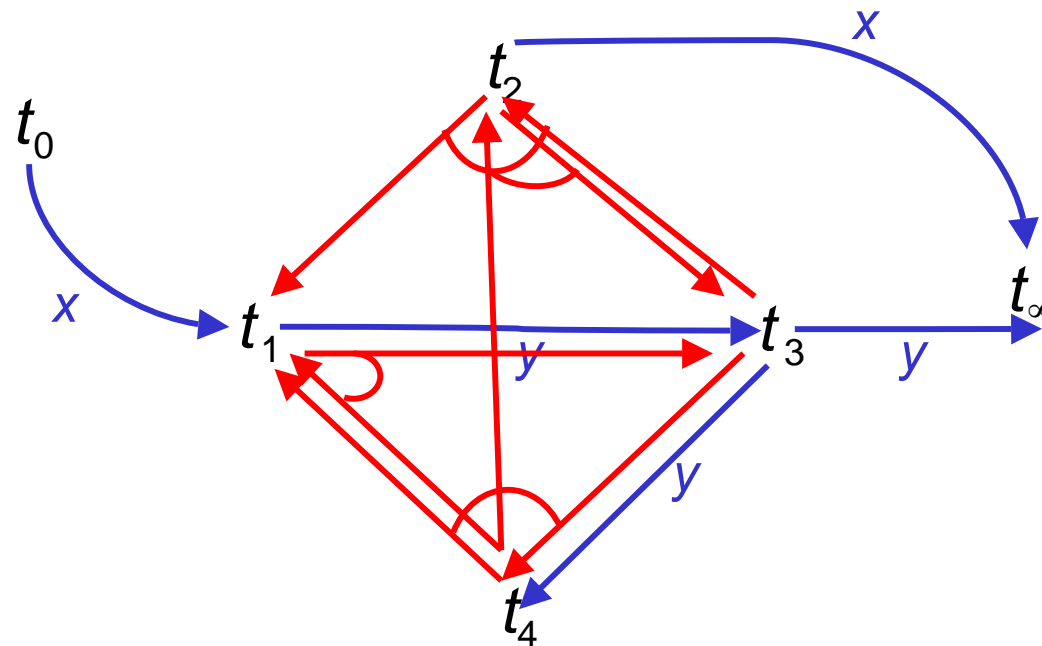
Eine Choice (t_i, t_k, t_j) drückt aus, dass t_k entweder nach t_i oder vor t_j stattfinden muss.

Nachweis der Serialisierbarkeit (6)

46

Beispiel:

$$h_1 = w_0(x) w_0(y) r_1(x) w_2(y) w_1(y) r_3(y) w_3(y) w_2(x) r_4(y) r_\infty(x) r_\infty(y)$$



Anmerkung: Für t_0 , t_∞ gibt es keine Wahl: t_0 liest nicht, t_∞ schreibt nicht.

Nachweis der Serialisierbarkeit (7)

47

Definition 5.36 (azyklischer Polygraph)

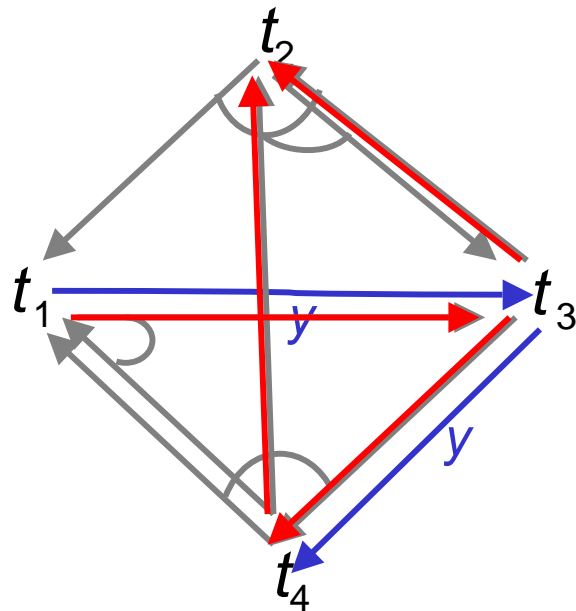
Ein Polygraph $P = (V, E, C)$ heißt **azyklisch**, falls mindestens ein azyklischer gerichteter Graph (V, E') in $\mathbf{D}(V, E, C)$ existiert.



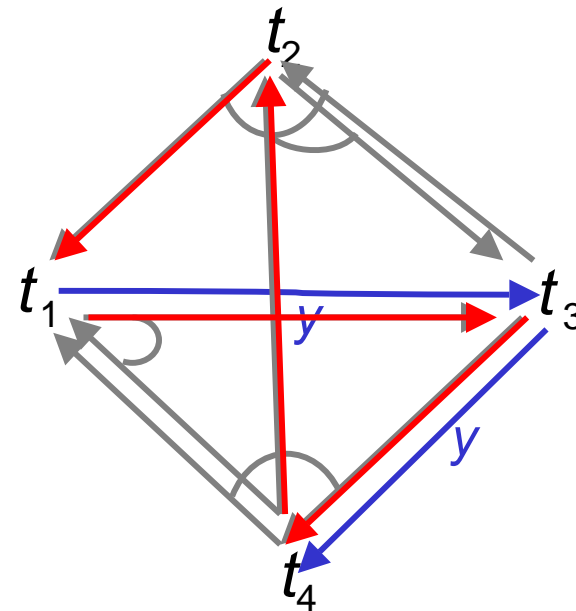
Indiz für Komplexität!

Nachweis der Serialisierbarkeit (8)

Voriges Beispiel: Zwei gerichtete Graphen aus der Familie



zyklenfrei



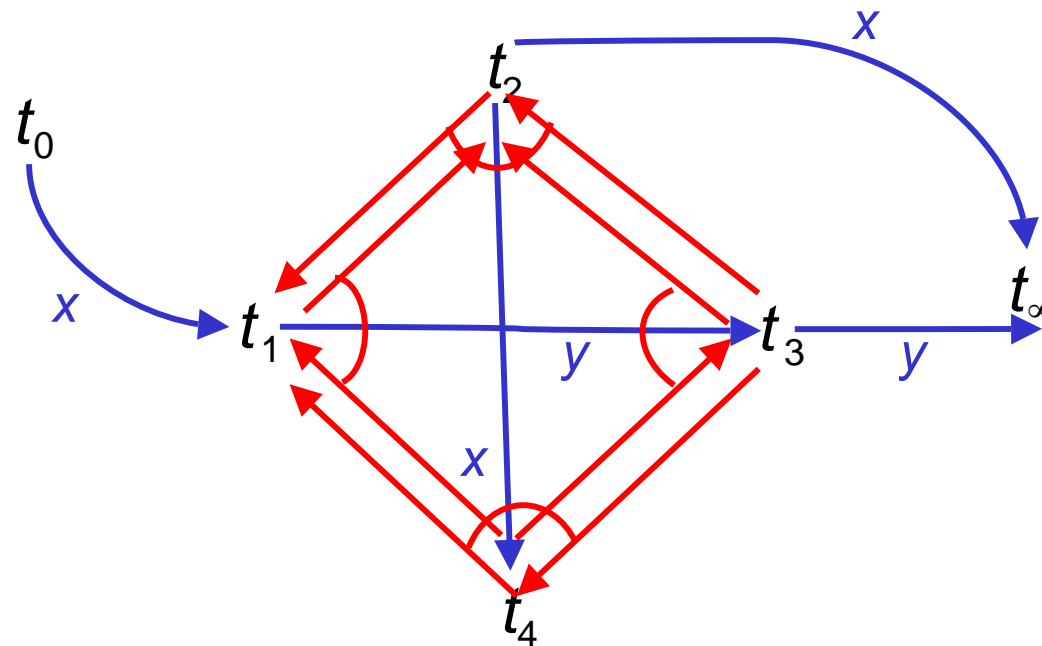
zyklenbehaftet

t_0, t_∞ können nicht an Zyklen partizipieren.

Nachweis der Serialisierbarkeit (9)

Gegenbeispiel:

$$h_1 = w_0(x) w_0(y) r_1(x) w_2(y) w_1(y) r_3(y) w_3(y) w_2(x) r_4(y) r_\infty(x) r_\infty(y)$$



Anmerkung: Für t_0 , t_∞ gibt es keine Wahl: t_0 liest nicht, t_∞ schreibt nicht.

Nachweis der Serialisierbarkeit (10)

Lemma 5.37

Seien h und h' zwei Historien. Dann gilt:

$$h \equiv_v h' \Leftrightarrow P(h) = P(h').$$

Intuitiver Beweis: Beide setzen gleiche Operationen und gleiche Relation RF voraus.

Lemma 5.38

Sei h Historie. Dann: h seriell $\Leftrightarrow P(h)$ azyklisch.

Nachweis der Serialisierbarkeit (11)

Beweis zu 5.38:

- Konstruiere $G = (V, E')$ kompatibel zu P :
 - ◆ Knoten $V = T \cup \{t_0, t_\infty\}$ übernehmen
 - ◆ Kanten: $(t, t') \in E$ übernehmen $\succ t < t'$ (sonst könnte t' nicht von t lesen)
 - ◆ Choices: $(t, t', t'') \in C$ (d.h. Operation p von Transaktion t' schreibt Element, das t von t'' liest):
 - p vor dem Schreiben von t'' : t' vor t'' : $(t', t'') \in E'$
 - p nach dem Lesen von t : t' nach t : $(t, t') \in E'$
- Somit $E' = \{(t, t') \mid t, t' \in V \wedge t < t'\}$
- (V, E') azyklisch, weil h seriell

Nachweis der Serialisierbarkeit (12)

Satz 5.39

h sichtserialisierbar $\Leftrightarrow P(h)$ azyklisch

(Analogon zum Serialisierbarkeitstheorem für Konfliktserialisierbarkeit.)

Beweis: h sichtserialisierbar $\Leftrightarrow P(h)$ azyklisch

$\supset \exists h_s$ seriell $h \equiv_V h_s$

$P(h) = P(h_s)$ nach 5.37

$P(h_s)$ azyklisch nach 5.38

$\supset P(h)$ azyklisch

Nachweis der Serialisierbarkeit (13)

Beweis: h sichtserialisierbar $\Leftrightarrow P(h)$ azyklisch

Sei G kompatibler Graph. Sei h_s eine topologische Ordnung von G .

Zu zeigen: $h_s \equiv_V h$.

Widerspruchsbeweis:

RF sei verschieden,

- ◆ d.h. es gibt $r_t(x)$ mit $r_t(x) \triangleright_h w_i(x)$, $r_t(x) \triangleright_{h_s} w_j(x)$
 - □ $(t, j, i) \in P(h)$
 - □ $(i, t) \in G$ und damit $i < t \in h_s$
 - außerdem: $i < j \in h_s$ und damit $(i, j) \in G$
 - Widerspruch zu (t, j, i)

Entscheidbarkeit der Serialisierbarkeit

54

- Zur Erinnerung: Das Entscheidungsproblem, ob eine Historie konfliktserialisierbar ist oder nicht, kann in polynomieller Zeit berechnet werden.
- Man konnte bisher nur zeigen, dass das entsprechende Entscheidungsproblem für Sichtserialisierbarkeit NP-vollständig ist.
- Siehe dazu C.H.Papadimitriou: The Serializability of Concurrent Database Updates, JACM 46(4), 631ff, Oct. 79.

Final-State Serializability

(für Interessierte)

Final-state equivalence

Remember:

Definition 5.40

Two histories h and h' are **final-state equivalent** ($h \equiv_F h'$) \Leftrightarrow

- $op(h) = op(h')$
- $\forall x \in D \text{ FIN}_h(x) = \text{FIN}_{h'}(x)$ (D database)

Compare to:

Two histories h and h' are **view equivalent** ($h \equiv_V h'$): \Leftrightarrow

- $op(h) = op(h')$
- $RF(h) = RF(h')$
- $\forall x \in D \text{ FIN}_h(x) = \text{FIN}_{h'}(x)$ (D database)

Herbrand-Semantik von Historien (1)

57

Consequence:

- Unlike before, there is no defined relationship to start from.
- All we can do is take a snapshot and characterize the current state.
- The characterization should take the past actions into account: **semantic characterization** of a history.
- Approach to semantics in mathematical logic: Construct a domain as a set of (syntactic) terms, each for a specific individual (Herbrand domain).

Herbrand-Semantik von Historien (2)

58

Interpretation of j^{th} step, p_j , of t :

- If $p_j = r(x)$,
then interpretation is assignment $v_j := x$ to local variable v_j
- If $p_j = w(x)$
then interpretation is assignment $x := f_j(v_{j_1}, \dots, v_{j_k})$
with unknown function f_j
and j_1, \dots, j_k denoting t 's prior read steps.

Beispiel

$$t = w(u) r(x) w(v) r(y) w(w)$$

Die Werte der geschriebenen Elemente sind abhängig von allen vorher gelesenen Elementen. Also:

$$u = f_1()$$

$$v = f_2(x)$$

$$w = f_3(x, y)$$

Herbrand-Semantik von Historien (3)

Definition 5.41 (Herbrand- Semantik)

Sei h eine Historie. Die **Herbrand-Semantik** H_h der Schritte $r_i(x), w_i(x) \in op(h)$ wird rekursiv wie folgt definiert:

- $H_h(r_i(x)) := H_h(w_j(x))$

wobei $r_i(x) \triangleright_h w_j(x)$ (also $j = 1, \dots, m$)

Wir machen also Gebrauch von der liest von-Beziehung, aber nur zur Berechnung der formalen Semantik!

- $H_h(w_i(x)) := f_{i,x}(H_h(r_i(y_1)), \dots, H_h(r_i(y_m)))$, wobei

- ◆ die $r_i(y_j)$ ($1 \leq j \leq m$) sämtliche Leseschritte von t_i sind, welche in h vor $w_i(x)$ stehen, und
- ◆ $f_{i,x}$ ein uninterpretiertes m -stelliges Funktionssymbol ist.

Anmerkung: Wir benötigen initiales Schreiben. Daher vervollständigen wir die Historie wie bei Sicht-Serialisierbarkeit um Transaktionen t_0, t_∞ .

Herbrand-Semantik von Historien (4)

Beispiel 5.42: Für die Historie

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

gilt für H_h :

$$H_h(w_0(x)) = f_{0,x}()$$

$$H_h(w_0(y)) = f_{0,y}()$$

$$H_h(r_1(x)) = f_{0,x}()$$

$$H_h(r_2(y)) = f_{0,y}()$$

$$H_h(w_2(x)) = f_{2,x}(f_{0,y}())$$

$$H_h(w_1(y)) = f_{1,y}(f_{0,x}())$$

Herbrand-Semantik von Historien (5)

Definition 5.43 (Herbrand-Universum)

Das **Herbrand-Universum** HU für Transaktionen t_i , $i > 0$, ist die kleinste Menge von Symbolen, für die gilt:

- $f_{0,x}() \in HU$ für jedes $x \in D$ (D Datenbasis), wobei $f_{0,x}$ ein 0-stelliges Funktionssymbol ist.
- $f_{i,x}(v_1, \dots, v_m) \in HU$ ($f_{i,x}$ ein m -stelliges Funktionssymbol) falls $w_i(x) \in op(t_i)$, mit
 - ◆ $m = | \{ r_i(y) \mid \exists y \in D \ r_i(y) <_{t_i} w_i(x) \} |$
 - ◆ $v_i \in \{ r_i(y) \mid \exists y \in D \ r_i(y) <_{t_i} w_i(x) \} \subset HU$, $1 \leq i \leq m$

Herbrand-Semantik von Historien (6)

62

Definition 5.44 (Semantik einer Historie)

Die **Semantik** einer Historie h ist die Abbildung

$$H(h) : D \rightarrow HU$$

definiert durch

$$H(h)(x) := H_h(FIN_s(x)) \quad (x \in D).$$

Die Semantik eines Schedules ist also die Menge derjenigen Werte, die von nicht-abgebrochenen Transaktionen als letzte geschrieben werden.

Herbrand-Semantik von Historien (7)

Beispiel 5.45 Für die Historie Beispiel 5.42

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

gilt für H_h

$$H_h(w_0(x)) = f_{0,x}()$$

$$H_h(w_0(y)) = f_{0,y}()$$

$$H_h(r_1(x)) = f_{0,x}()$$

$$H_h(r_2(y)) = f_{0,y}()$$

$$H_h(w_2(x)) = f_{2,x}(f_{0,y}())$$

$$H_h(w_1(y)) = f_{1,y}(f_{0,x}())$$

und somit für $H(h)$

$$H(h)(x) = H_h(w_2(x)) = f_{2,x}(f_{0,y}())$$

$$H(h)(y) = H_h(w_1(y)) = f_{1,y}(f_{0,x}())$$

Final-State-Äquivalenz (1)

Definition 5.46 (final-state-äquivalent)

Zwei Historien h und h' heißen **final-state-äquivalent** ($h \equiv_F h'$) $\square \square$

- $op(h) = op(h')$
- $H(h) = H(h')$

Zwei Schedules sind also final-state-äquivalent gdw. sie beide die gleichen Operationen umfassen und den gleichen Endzustand erzeugen.

Final-State-Äquivalenz (2)

Beispiel 5.47

Gegeben seien die folgenden Schedules:

$$h = r_1(x) r_2(y) w_1(y) r_3(z) w_3(z) r_2(x) w_2(z) w_1(x) c_1 c_2 c_3$$
$$h' = r_3(z) w_3(z) c_3 r_2(y) r_2(x) w_2(z) c_2 r_1(x) w_1(y) w_1(x) c_1$$

Dann gilt $op(h) = op(h')$ und

$$\begin{aligned} H(h)(x) &= H_h(w_1(x)) = f_{1,x}(f_{0,x}()) \\ &= H_{h'}(w_1(x)) = H(h')(x) \\ H(h)(y) &= H_h(w_1(y)) = f_{1,y}(f_{0,x}()) \\ &= H_{h'}(w_1(y)) = H(h')(y) \\ H(h)(z) &= H_h(w_2(z)) = f_{2,z}(f_{0,x}(), f_{0,y}()) \\ &= H_{h'}(w_2(z)) = H(h')(z) \end{aligned}$$

also insgesamt $h \equiv_F h'$.

Final-State-Äquivalenz (3)

Beispiel 5.48

Gegeben seien die folgenden Schedules:

$$h = r_1(x) \ r_2(y) \ w_1(y) \ w_2(y) \ c_1 \ c_2$$

$$h' = r_1(x) \ w_1(y) \ c_1 \ r_2(y) \ w_2(y) \ c_2$$

Dann gilt $op(h) = op(h')$ und

$$H(h)(y) = H_h(w_2(y)) = f_{2,y}(f_{0,y}())$$

$$\begin{aligned} H(h')(y) &= H_{h'}(w_2(y)) = f_{2,y}(H_{h'}(r_2(y))) \\ &= f_{2,y}(H_{h'}(w_1(y))) \\ &= f_{2,y}(f_{1,y}(f_{0,x}())) \end{aligned}$$

also insgesamt $h \not\equiv_F h'$.

Dieses Beispiel zeigt: Nicht allein das letzte Schreiben ist entscheidend, sondern auch die Vorgeschichte.

Final-State-Äquivalenz (4)

Aufgabe: Erfassen der „einflussreichen“ Vorgeschichte.

Definition 5.49

Eine Operation p ist in h **direkt nützlich** für eine Operation q ($p \mapsto_h q$), falls

- q ist Leseoperation und liest von fremdem Schreiber p
 $q \triangleright_h p$
- q ist Schreiboperation und p geht als eigener Leser voran
 $p = r_i(x)$, $q = w_i(y)$ und $p <_h q$.

\mapsto^* (*nützlich*) bezeichne die reflexive, transitive Hülle von \mapsto .

Final-State-Äquivalenz (5)

68

Definition 5.50

Eine Operation p heißt **lebendig** in $h \Leftarrow \triangleright$

$$\exists q \in t_\infty \quad p \mapsto^* q$$

d.h., ihr Ergebnis trägt zum Endergebnis bei.

Sie heißt **tot** sonst.

Final-State-Äquivalenz (6)

Grundgedanke: Von der Liest-von-Relation interessieren nur die Operationen mit einem Beitrag zum Endergebnis.

Definition 5.51

Die **Lebendig-Liest-von-Relation** (*live-reads-from*) von h ist definiert durch

$$LRF(h) := \{(t_i, x, t_j) \mid r_j(x) \text{ lebendig, } r_j(x) \triangleright_h w_i(x)\}$$

Anmerkung: Alle (Lese-)Operationen aus t_∞ gelten als lebendig. (Also (t_j, x, t_∞) immer in LRF).

Final-State-Äquivalenz (8)

Beispiel 5.52 Nach vervollständigtem Beispiel 5.48

$$h = w_0(x) w_0(y) c_0 r_1(x) r_2(y) w_1(y) w_2(y) c_1 c_2 r_\infty(x) r_\infty(y) c_\infty$$

$$h' = w_0(x) w_0(y) c_0 r_1(x) w_1(y) c_1 r_2(y) w_2(y) c_2 r_\infty(x) r_\infty(y) c_\infty$$

Dann gilt:

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

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

In h und h' : $w_0(x) \mapsto r_1(x) \mapsto w_1(y) \succ w_0(x) \mapsto^* w_1(y)$

$$w_0(x) \mapsto r_\infty(x) \succ w_0(x) \mapsto^* r_\infty(x)$$

$$r_2(y) \mapsto w_2(y) \mapsto r_\infty(y) \succ r_2(y) \mapsto^* r_\infty(y)$$

In h : $w_0(y) \mapsto_h r_2(y) \succ w_0(y) \mapsto_h^* r_\infty(y)$

In h' : $w_1(y) \mapsto_{h'} r_2(y) \succ$

$$w_1(y) \mapsto_{h'}^* r_\infty(y), r_1(x) \mapsto_{h'}^* r_\infty(y), w_0(x) \mapsto_{h'}^* r_\infty(y)$$

Final-State-Äquivalenz (8)

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

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

In h und h' : $w_0(x) \mapsto r_1(x) \mapsto w_1(y) \succ w_0(x) \mapsto^* w_1(y)$

$$w_0(x) \mapsto r_\infty(x) \succ w_0(x) \mapsto^* r_\infty(x)$$

$$r_2(y) \mapsto w_2(y) \mapsto r_\infty(y) \succ r_2(y) \mapsto^* r_\infty(y)$$

In h : $w_0(y) \mapsto_h r_2(y) \succ w_0(y) \mapsto_h^* r_\infty(y)$

In h' : $w_1(y) \mapsto_{h'} r_2(y) \succ$

$$w_1(y) \mapsto_{h'}^* r_\infty(y), r_1(x) \mapsto_{h'}^* r_\infty(y), w_0(x) \mapsto_{h'}^* r_\infty(y)$$

LRF wird aus RF gewonnen, indem für jedes Tupel (t_j, x, t_i) betrachtet wird, ob $r_j(x)$ lebendig, d.h. nützlich für eine Operation in t_∞ , ist. Also:

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

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

Final-State-Äquivalenz (9)

72

Die obigen Überlegungen legen den folgenden Satz nahe:

Satz 5.53

Seien h und h' zwei Historien mit $op(h) = op(h')$.

Dann gilt:

$$h \equiv_F h' \Leftrightarrow LRF(h) = LRF(h')$$

Wieder gesucht: Graphbasierte Lösung
 \Rightarrow Graph muss auf LRF aufbauen.

Final-State-Äquivalenz (10)

73

Definition 5.54 (Schrittgraph)

Der **Schrittgraph** $D(h) = (V, E)$ einer Historie h ist definiert durch

$$V := op(h)$$

$$E := \{ (p, q) \mid p \mapsto_h q \}$$

Der **reduzierte Schrittgraph** $D_1(h)$ entsteht aus $D(h)$ durch Weglassen aller toten Knoten und deren inzidenten Kanten.

Beweis-Skizze für Äquivalenz: Zeige

- $h \equiv_F h' \Leftrightarrow D_1(h) = D_1(h')$
- $D_1(h) = D_1(h') \Leftrightarrow LRF(h) = LRF(h')$

Final-State-Äquivalenz (11)

Beispiel 5.55

Betrachte die Schedules aus Beispiel 5.47 ($op(h) = op(h')$, h' seriell):

$h = w_0(x,y,z) c_0 r_1(x) r_2(y) w_1(y) r_3(z) w_3(z) r_2(x) w_2(z) w_1(x) c_1 c_2 c_3$

$h' = w_0(x,y,z) c_0 r_3(z) w_3(z) c_3 r_2(y) r_2(x) w_2(z) c_2 r_1(x) w_1(y) w_1(x) c_1$

Für diese hatten wir anhand gleicher Endzustände bereits gezeigt: $h \equiv_F h'$.

$LRF(h) =$

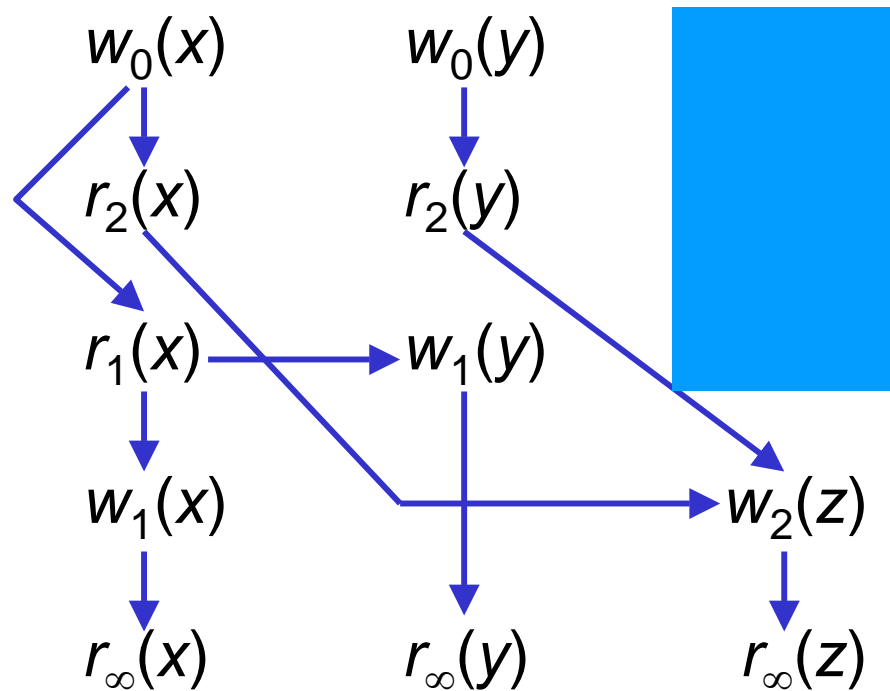
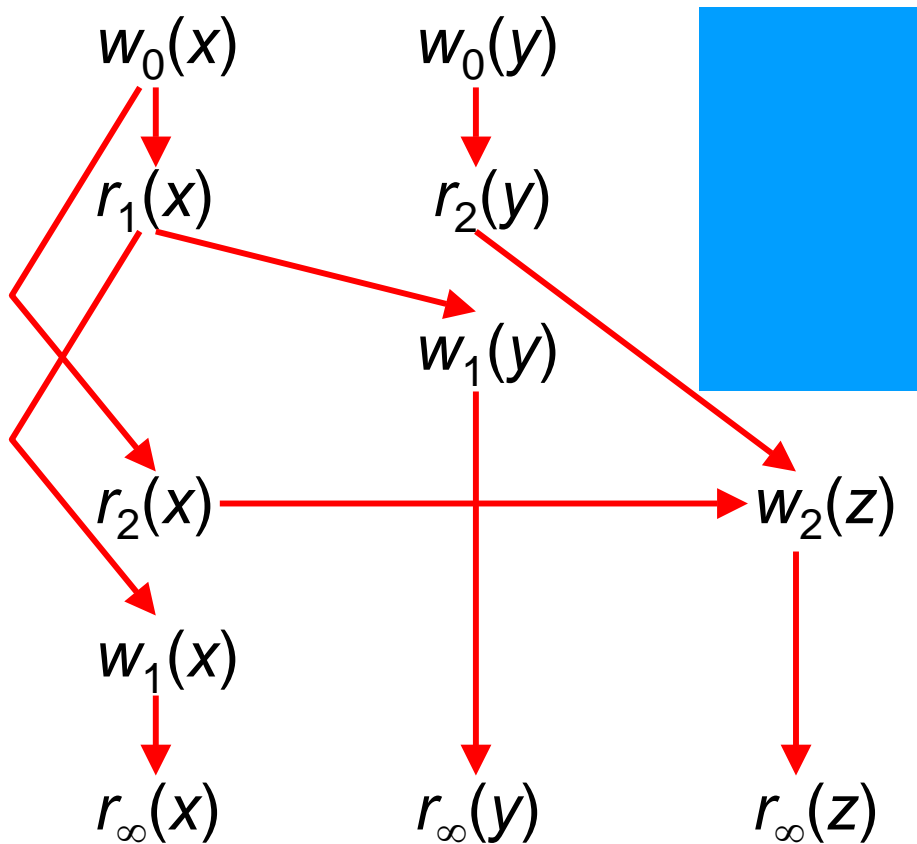
$\{(t_0, x, t_1), (t_0, x, t_2), (t_0, y, t_2), (t_1, x, t_\infty), (t_1, y, t_\infty), (t_2, z, t_\infty)\}$

$= LRF(h')$

Final-State-Äquivalenz (12)

$h = w_0(x,y,z) \ c_0 \ r_1(x) \ r_2(y) \ w_1(y) \ r_3(z) \ w_3(z) \ r_2(x) \ w_2(z) \ w_1(x) \ c_1 \ c_2 \ c_3$

$h' = w_0(x,y,z) \ c_0 \ r_3(z) \ w_3(z) \ c_3 \ r_2(y) \ r_2(x) \ w_2(z) \ c_2 \ r_1(x) \ w_1(y) \ w_1(x) \ c_1$



h und h' final-state-äquivalent

Final-State-Äquivalenz (13)

Beispiel 5.56

Betrachte die Historien aus Beispiel 5.52 (h' seriell):

$$h = w_0(x) w_0(y) c_0 r_1(x) r_2(y) w_1(y) w_2(y) c_1 c_2 r_\infty(x) r_\infty(y) c_\infty$$
$$h' = w_0(x) w_0(y) c_0 r_1(x) w_1(y) c_1 r_2(y) w_2(y) c_2 r_\infty(x) r_\infty(y) c_\infty$$

h und h' sind nicht final-state-äquivalent:

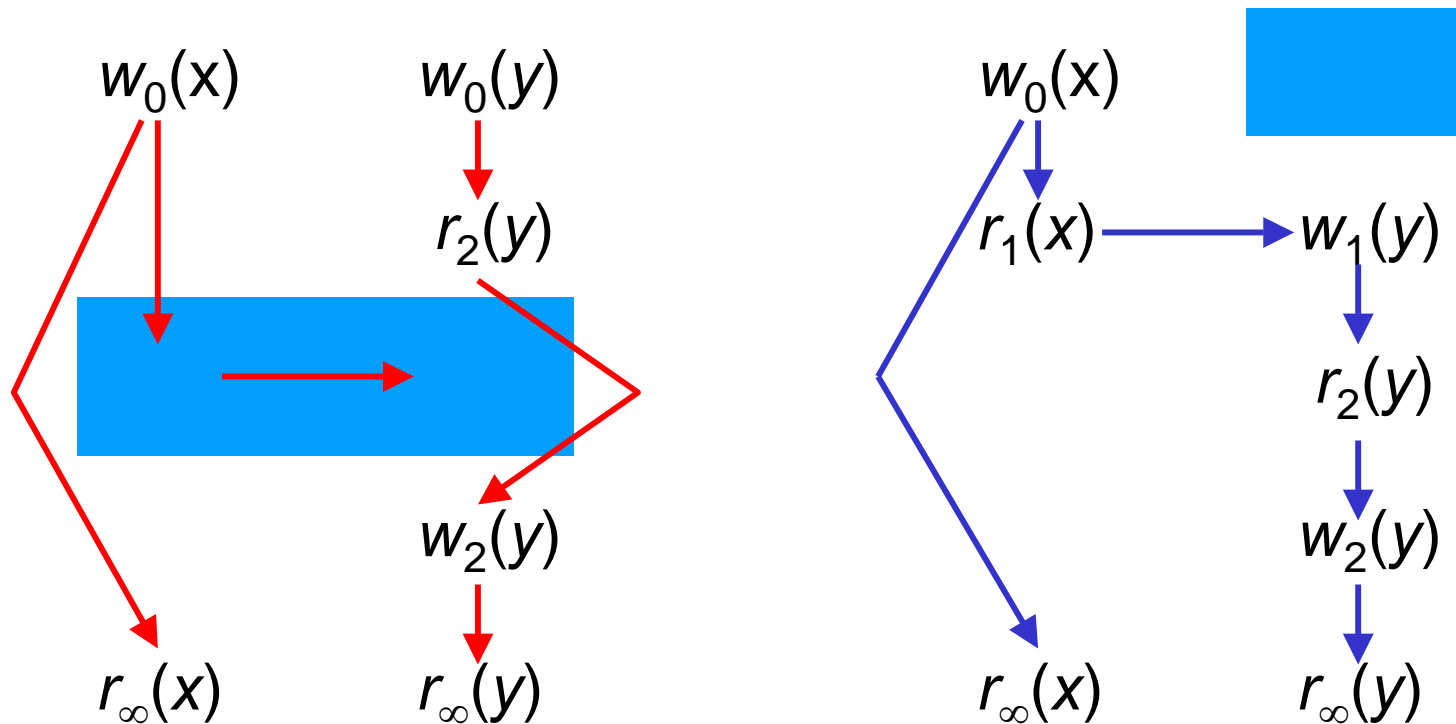
$$LRF(h) = \{(t_0, y, t_2), (t_0, x, t_\infty), (t_2, y, t_\infty)\}$$

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

Final-State-Äquivalenz (14)

$$h = w_0(x) w_0(y) c_0 r_1(x) r_2(y) w_1(y) w_2(y) c_1 c_2 r_\infty(x) r_\infty(y) c_\infty$$

$$h' = w_0(x) w_0(y) c_0 r_1(x) w_1(y) c_1 r_2(y) w_2(y) c_2 r_\infty(x) r_\infty(y) c_\infty$$



h und h' nicht final-state-äquivalent

Final-State-Serialisierbarkeit

Korollar 5.57

Die Final-State-Äquivalenz zweier Schedules h und h' kann in polynomieller Zeit entschieden werden.

Definition 5.58 (Final-State-Serialisierbar)

Eine Historie h heißt **final-state-serialisierbar**, falls es eine seriellen Historie h' gibt, mit $h \equiv_F h'$.

FSR = Menge aller final-state-serialisierbaren Historien.

Test auf Final-State-Serialisierbarkeit:

Teste für alle (seriellen) Permutationen h' von h :

$$h \equiv_F h'.$$

Vermutung: Exponentielle Komplexität.

Vermutung gestützt auf gleichartige Komplexität der bereits eingeschränkten Sichtserialisierbarkeit.

Commit Serializability

Prefix commit closure (1)

- **Remember:** A schedule s is X -serializable if there exists a serial history h_s s.t. $\exists h_s \text{ serial: } CP(s) \equiv_X h_s$.
- **Problem:** The outcome of active transactions is unknown.
- **Desirable property:** Whatever the outcome (abort or commit), it should not invalidate the correctness of already committed transactions.
- **Formal criterion:** All committed prefixes of correct schedules are correct.

Prefix commit closure (2)

- **Definition 5.59: Prefix commit closure** class of schedules:

$\forall s: s \text{ X-serializable} \succ \forall s' \leq s: \exists h_s \text{ serial: } CP(s') \equiv_X h_s.$

- ◆ **Motivation:** The **complete** past is correct! The schedule can progress without any concern for the past!

- **Definition 5.60**

A schedule s is **commit serializable (CM)** if the prefix commit closure property holds for s .

- Special classes:

$CMVSR$ = commit view serializable

$CMCSR$ = commit conflict serializable

Commit Serializability (1)

Theorem 5.61 (without proof)

- Membership in class CSR is prefix commit closed:

$$CMCSR = CSR$$

- Membership in class VSR (set of all view serializable histories) is generally not prefix commit closed.

$$CMVSR \subset VSR$$

Commit Serializability (1)

Example 5.62

$$h_{12} = w_1(x) w_2(x) w_2(y) c_2 w_1(y) c_1 w_3(x) w_3(y) c_3$$
$$CP(h_{12}) = h_{12}.$$

View equivalences (solely writes!)

$$h_{12} \equiv_V t_1 t_2 t_3 \quad \text{and} \quad h_{12} \equiv_V t_2 t_1 t_3$$

However, prefix

$$h'_{12} = w_1(x) w_2(x) w_2(y) c_2 w_1(y) c_1$$

is equivalent neither to $t_1 t_2$ nor to $t_2 t_1$ (in both cases the final state differs from that for h'_{12}).

$\Rightarrow h'_{12}$ is not in *CMVSR*.

CMVSR is characterized by:

$$\forall h' \leq h \exists h_s \text{ serial: } CP(h') \equiv_V h_s \text{ (not just: } \exists h_s \text{ serial: } CP(h) \equiv_V h_s \text{)}$$

Commit Serializability (2)

Theorem 5.63 (without proof)

$CMCSR \subset CMVSR$

Commit Serializability (3)

