Datenbanken
Wintersemester 06/07
Prof. Dr. W. May

4. Übungsblatt: Transaktionen

Besprechung am 24.1.

Aufgabe 1 (Zeitstempelbasierte Synchronisation) Zeigen Sie, daß eine zeitstempelbasierte Synchronisation nur konfliktgraph-serialisierbare Schedules erzeugt.

Annahme: es existiert ein Schedule S, dessen Konfliktgraph CG zyklisch ist, und der von einem zeitstempelbasierten Scheduler erzeugt worden ist.

CG enthält also einen Zyklus der Art $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow \ldots \rightarrow T_n \rightarrow T_1$.

$\Rightarrow$ wenn $T_i \rightarrow T_j$ in CG existiert, dann existiert ein Konflikt (also WR oder WW oder RW) zwischen T_i und T_j bzgl. einem Daten-Item.

Der Schedule enthält also Konfliktaktionen $\ldots A_{T_i} \ldots A_{T_j} \ldots$. D.h. T_i wurde vor T_j gestartet, da der Schedule sonst nicht von dem zeitstempelbasierten Scheduler erzeugt worden wäre. Damit also Zeitstempel(T_i) < Zeitstempel (T_j)

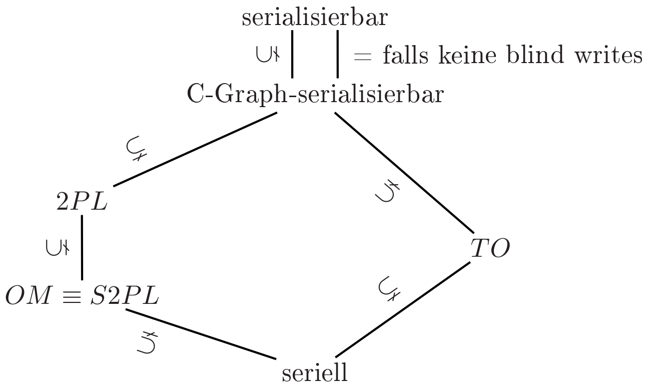
Analog für die anderen Kanten des Zylus. Also Zeitstempel(T_1) < Zeitstempel(T_2) < Zeitstempel(T_3) < … < Zeitstempel(T_n) < Zeitstempel(T_1)
$\Rightarrow$ Zeitstempel(T_1) < Zeitstempel(T_1), was ein Widerspruch ist.

Aufgabe 2 (Vergleich der Mächtigkeit von Schedulern) Ein Scheduler Φ_1 is *strenger* als ein Scheduler Φ_2 ($\Phi_1 \subseteq \Phi_2$), wenn für jeden Schedule $S \in \Phi_1$ auch $S \in \Phi_2$ gilt. Zwei Scheduler sind äquivalent, wenn $\Phi_1 \subseteq \Phi_2$ und $\Phi_2 \subseteq \Phi_1$.

Betrachten Sie die folgenden Strategien: Überwachen des Konfliktgraphen (CG), 2-Phasen-Sperren (2PL), strenges 2PL (S2PL), Zeitmarken (TO) und optimistisches Verfahren (OM). Vergleichen Sie die Mächtigkeit der einzelnen Strategien.

Gezeigt wird:

Alle Schedules, die von einem Scheduler S_{2PL}, S_{TO} oder S_{OM} akzeptiert werden, werden auch von S_{CG} akzeptiert. Es existieren Schedules, die von CG akzeptiert werden, jedoch nicht von S_{2PL}, S_{TO} oder S_{OM}. Für S_{2PL}, S_{TO} gilt: es existieren jeweils Schedules, die von einem Scheduler, jedoch nicht von dem anderen akzeptiert werden. Alle Schedules, die von S_{OM} akzeptiert werden, werden auch von S_{2PL} akzeptiert. Es gibt Schedules, die von S_{2PL} akzeptiert werden, jedoch nicht von S_{OM}. Es existieren Schedules, die von S_{TO} akzeptiert werden, jedoch nicht von S_{OM}, und umgekehrt.



(1) $2PL \subseteq CG$
Annahme: $S \in 2PL$, $S \notin CG$
$\Rightarrow CG(S)$ zyklisch $\Rightarrow$ oBdA: $T_1 \to T_2 \to \ldots \to T_k \to T_1$
Dann ex. $A_1 \ldots A_k$ so dass bzgl. S gilt:

$$\begin{array}{lcl}
S = \ldots U_1 A_1 \ldots L_2 A_1 \ldots & \Rightarrow & \text{Sperrpunkt } L_1 \text{ vor Sperrpunkt } L_2 \\
S = \ldots U_2 A_2 \ldots L_3 A_2 \ldots & \Rightarrow & \text{Sperrpunkt } L_2 \text{ vor Sperrpunkt } L_3 \\
\vdots & & \\
S = \ldots U_{k-1} A_{k-1} \ldots L_k A_{k-1} \ldots & \Rightarrow & \text{Sperrpunkt } L_{k-1} \text{ vor Sperrpunkt } L_k \\
S = \ldots U_k A_k \ldots L_1 A_k \ldots & \Rightarrow & \text{Sperrpunkt } L_k \text{ vor Sperrpunkt } L_1
\end{array}$$

$\Rightarrow$ Sperrpunkt L_1 vor Sperrpunkt $L_1 \leadsto$ Widerspruch

(2) $TO \subseteq CG$
Siehe Aufgabe 1 (analoge Technik zu (1) mit Schluss
"Zeitstempel(T_1) < Zeitstempel(T_2) < ... < Zeitstempel(T_k)< Zeitstempel(T_1)")

(3) $OM \subseteq CG$
Widerspruchsbeweis analog zu (1) und (2) mit Schluss
"EOT(T_1) < EOT(T_2) < ... < EOT(T_k) < EOT(T_1)",
da hier die EOTs, an denen die read/writesets verglichen werden, die kritischen Punkte sind.
Annahme: $S \in OM$, $S \notin CG$
$\Rightarrow CG(S)$ zyklisch o.B.d.A. $T_i \to T_j \to T_i$.
$\Rightarrow$ wenn $T_i \to T_j$ in CG existiert, dann existiert ein Konflikt (also WR oder WW oder RW) zwischen T_i und T_j bzgl. einem Daten-Item.
Der Schedule enthält also Konfliktaktionen $\ldots A_{T_i} \ldots A_{T_j} \ldots$. D.h., das Transaktionsende EOT_i von T_i muss vor A_{T_j} und somit vor EOT_j liegen, sonst hätte OM nämlich die Read/Write-Sets verglichen und gemerkt, dass entweder $rs(T_i) \cap ws(T_j) \neq \emptyset$ oder $ws(T_i) \cap ws(T_j) \neq \emptyset$ oder $ws(T_i) \cap rs(T_j) \neq \emptyset$ ist.
Analog kann man auch aus $T_j \to T_i$ in CG schliessen, dass C_j vor A_{T_i} und somit vor C_i liegt.
$\leadsto$ Widerspruch

(4) $2PL$, TO und OM sind strenger als CG:
Prinzip: man "erfindet" einen Schedule, der in CG ist, nicht aber in $2PL$, TO, OM:
- nicht in 2PL: es muss eine Transaktion geben, die erst ein Datenitem an eine andere abgibt ($\to$ unlock) und später eines von einer anderen bekommt ($\to$ lock).
- nicht in TO: relativ einfach. Jede Transaktion bekommt den Zeitstempel zugewiesen, wenn sie ihre erste Aktion ausführt. Dies ist eine realistische Annahme, da in einem realen Daten-

banksystem eine Transaktion erst mit ihrer ersten Aktion bekannt wird.
Wenn dies nicht "zufällig" dieselbe Reihenfolge wie in dem äquivalenten seriellen Schedule ist, ist der Schedule nicht in TO.

– nicht in OM: OM lehnt "gute" Schedules ab, in denen die "nachfolgende" Transaktion ein Objekt bereits "übernimmt" während die Vorgängertransaktion noch "lebt" (dirty reads, die zu Problemen führen können, falls die frühere Transaktion aborted, aber eben nicht ubedingt zu Problemen führen müssten. Damit vermeidet OM dirty reads auf Kosten von höherer Nebenläufigkeit [Dasselbe tut auch strict 2PL im Gegensatz zu 2PL].

$$S_1 \quad = \quad R_1A\ R_2A\ W_2A\ R_1B\ R_3B\ W_1B$$

$CG(S_1)$:

$$\begin{tikzcd} T_1 \arrow[r] & T_2 & T_3 \arrow[ll, bend left] \end{tikzcd}$$

– $S_1 \in CG$

– $S_1 \notin 2PL$: T_1 muss A nach der ersten oder zweiten Aktion freigeben (damit T_2 sein write-Lock bekommt), und kann B erst vor der letzten Aktion write-locken.

– $S_1 \notin TO$: $Z(T_1)$ ist der kleinste Zeitstempel, damit darf T_1 aber nach R_3B kein W_1B mehr machen.

– $S_1 \notin OM$: Nach W_2A sind beide Transaktionen T_1 und T_2 noch aktiv und $writeset(T_2) \cap readset(T_1) \supseteq \{A\}$. Wenn also eine von beiden beendet werden soll, stellt das optimistische Verfahren den Konflikt fest (Hinweis: das optimistische Verfahren prüft das erst am Ende, um sich so unter der Annahme "es wird schon gutgehen" zwischendurch die Vergleiche zu sparen).

Hinweis: der o.g. Schedule enthält nicht einmal ein blind-write (mit dem man sonst oft Gegenbeispiele konstruiert).

(5) TO und $2PL$ sind unvergleichbar:
Zu zeigen: (i) es gibt einen Schedule S_2, der in $2PL$, aber nicht in TO ist, und (ii) einen Schedule, für den das umgekehrte gilt.
(i):

$$S_2 \quad = \quad R_1A\ R_2B\ W_2B\ R_1B\ R_2C$$

$CG(S_2)$:

$$T_2 \longrightarrow T_1$$

– $S_2 \in CG$

– $S_2 \in 2PL$ (wenn T_2 C früh genug lockt –
$L_1A\ R_1A\ L_2B\ R_2B\ W_2B\ L_2C\ U_2B\ L_1B\ R_1B\ U_1A\ U_1B\ R_2C\ U_2C$)

– $S_2 \notin TO$: (Vergabe der Zeitstempel wieder bei der ersten Aktion) $Z(1)$ ist der kleinste Zeitstempel, damit darf T_1 aber nach W_2B kein R_1B mehr machen.

– $S_2 \notin OM$: Nach W_2B und R_1B ist $writeset(T_2) \cap readset(T_1) \supseteq \{B\}$. Dieser Konflikt wird festgestellt, wenn eine der Transaktionen beendet werden soll.

$$S_3 \quad = \quad R_3C\ R_1A\ R_2A\ W_2A\ R_3B\ W_1B$$

$CG(S_3)$:

$$T_3 \longrightarrow T_1 \longrightarrow T_2$$

– $S_3 \in CG$

- $S_3 \notin 2PL$: T_1 muss A nach seiner ersten Aktion freigeben, und kann B erst vor der letzten Aktion locken.
- $S_3 \in TO$: Die Zeitstempel-Vergabe stimmt mit der äquivalenten seriellen Transaktion überein. Hinweis: hieran sieht man, dass TO bereits allergisch auf das Vertauschen von unabhängigen Aktionen reagiert (vertausche R_1A und R_3C und $S_3' \notin TO$, obwohl es nicht stören würde).
- $S_3 \notin OM$: Nach R_1A und W_2A ist $writeset(T_2) \cap readset(T_1) \supseteq \{A\}$. Dieser Konflikt wird festgestellt, wenn eine der Transaktionen beendet werden soll.

(6) OM ist mindestens so streng wie $2PL$:

- Annahme: es gibt einen Schedule S, der von OM akzeptiert wird, jedoch nicht von $2PL$:
  Dann muss es in S eine Transaktion T_i geben, die nicht zweiphasig sein kann, also von der Form
  $S = \ldots R_iA \ldots W_jA \ldots R_kB \ldots W_iB \ldots$ ist (R/W oEdA verteilt, Hauptsache konfligierend).
  Damit ist nach W_jA $writeset(T_j) \cap readset(T_i) \supseteq \{A\}$ und T_i noch aktiv. Dieser Konflikt wird festgestellt, wenn eine der Transaktionen beendet werden soll. Also kann dieser Schedule nicht von OM erzeugt werden.
  Hinweis: Wenn T_j eine "spätere" Transaktion als T_i ist, ist dieser Konflikt unproblematisch - OM also zu streng.

Dass es Schedules gibt, die von 2PL akzeptiert werden, aber nicht von OM, wurde oben mit S_2 bereits gezeigt.

(7) Wenn man schon am Vergleich von $2PL$ und OM ist, kann man auch gleich $S2PL$ betrachten. Im Gegensatz zu $2PL$ hält $S2PL$ alle Sperren bis ans Ende der Transaktion. Damit kann genau das nicht passieren, womit man OM in den obigen Beispielen zur Ablehnung bringen konnte: dass eine "nachfolgende" Transaktion ein Datenitem übernimmt, während die vorhergehende noch aktiv ist. *$S2PL$ verhindert bereits zur Laufzeit*, dass sich read- und write-sets bzw write-write-sets zweier aktiver Transaktionen überschneiden. Damit ist es bezüglich der Menge der akzeptierten Schedules genau äquivalent zu OM (der Unterschied liegt darin, dass OM es erst am Transaktionsende merkt, während $S2PL$ es gleich verhindert - mit dem Nachteil, ständig überprüfen zu müssen).

(8) OM und TO sind nicht vergleichbar: Zu zeigen: (i) es gibt einen Schedule S_4, der in OM, aber nicht in TO ist, und (ii) einen Schedule, für den das umgekehrte gilt.

- Betrachte $S_4 = R_1A\ R_2B\ W_2C\ EOT_2\ R_1C$: S_4 ist in OM: Zum Zeitpunkt EOT_2 (End of Transaction) ist $writeset(T_2) = \{C\}$, und kein Problem feststellbar. S ist aber nicht in TO, da dies den kleineren Zeitstempel an T_1 vergibt, und den größeren an T_2 - somit wird R_1C abgelehnt, weil $Z(C) = Z(T_2) < Z(T_1)$ ist. Der äquivalente serielle Schedule wäre $T_2\ T_1$.
  TO akzeptiert nur Schedules, die äquivalent zu dem durch die Startzeiten der Transaktionen gegebenen seriellen Schedule sind.
- Betrachte S_3 oben: ist in TO, aber nicht in OM. OM lehnt den Schedule ab, weil T_2 bereits mit A arbeitet während T_1, das vorher auf A gearbeitet hat, noch aktiv ist (wäre aber problemlos, da T_2 "später" ist).

Aufgabe 3 (Äquivalenz von serialisierbar und konfliktserialisierbar) Sei $\mathcal{T}$ gegeben, wobei für $T \in \mathcal{T}$ gilt:

$$T = \ldots WA \ldots \Longrightarrow T = \ldots RA \ldots WA \ldots$$

(d.h., es gibt keine "blind writes").
Dann ist S serialisierbar gdw. S ist C-serialisierbar.

Beweis:

Allgemein folgt aus C-Serialisierbarkeit die Serialisierbarkeit ("$\Leftarrow$"). Bleibt zu zeigen, daß jetzt auch aus Serialisierbarkeit C-Serialisierbarkeit ("$\Rightarrow$") folgt, falls die Transaktionen keine *blind writes* enthalten.

Sei S ein serialisierbarer Schedule und S' ein serieller Schedule äquivalent zu S, also $DG(S) = DG(S')$.
Beh: $CG(S) = CG(S')$ (und damit $CG(S)$ auch zyklenfrei, also S auch C-serialisierbar).

Um dies zu zeigen, betrachten wir alle Paare von Aktionen u, v, die in Konflikt miteinander stehen. Diese Aktionen müssen in S und S' jeweils gleich geordnet sein (dann hat man nämlich denselben C-Graphen). Man betrachtet dazu Paare solcher Aktionen im seriellen Schedule S'.

Fall 1: WR Sei u eine Leseaktion (R_iA) und v die letzte vorhergehende Schreibaktion (W_jA). Dies sind also genau die Datenflusskanten im D-Graphen.

Da $DG(S) = DG(S')$ liest u A von dieser Schreibaktion v sowohl in S als auch in S'. Also liegt u nach v sowohl in S als auch in S'.

Fall 2: WW u und v sind beides Schreibaktionen $u = W_1A$, $v = W_2A$ und es existiert keine weitere Schreibaktion zwischen u und v:

$$S' = \ldots W_1A \ldots W_2A \ldots$$

Da S' seriell ist und angenommen wird, dass es keine blind writes gibt, hat S' die folgende Form:

$$S' = \ldots W_1A \ldots R_2A \ldots W_2A \ldots$$

Da W_1A die letzte Schreibaktion vor R_2A ist und R_2A vor W_2A steht, hat $DG(S')$ folgende Gestalt (siehe Definition D-Graph):

$$W_1A \longrightarrow R_2A \longrightarrow W_2A$$

Daraus folgt mittels Induktion, dass alle Schreibaktionen zu einer Entität X in $DG(S')$ total geordnet sind (und das ist der Knackpunkt, der nur beim Ausschluss von blind writes garantiert werden kann).

Da $DG(S) = DG(S')$ sind alle Schreibaktionen zu einer Entität in S und S' in gleicher Weise geordnet.

Fall 3: RW Sei u eine Leseaktion (R_iA) und v die nächste darauffolgende Schreibaktion (W_jA) in S. Da $DG(S) = DG(S')$ liest u A von der gleichen Schreibaktion w in S und S'.

Annahme: u liegt in S nach v. v muss also in S' auch vor w liegen. In (2) wurde gezeigt, dass Schreibaktionen in S und in S' in derselben Reihenfolge geordnet sind, würde dies bedeuten, dass es in S' auch so ist – dort haben wir aber nach Voraussetzung für Fall 3 $w - u - v$.

Also liegt auch in S v nach u.

Es sind also alle konflikt-induzierenden Paare von Aktionen zu einem Daten-Item in S und S' gleich geordnet, und somit ist $CG(S) = CG(S')$.

Aufgabe 4 (D-Graph, C-Graph am Beispiel) Gegeben sei der folgende, aus Aktionen dreier Transaktionen bestehende Schedule:

$$S \quad = \quad R_1A\ R_3C\ R_3B\ W_3C\ W_3B\ R_1B\ R_2C\ W_1B\ R_2B\ W_2B\ W_2C\ W_1A$$

a) Bestimmen Sie den D-Graphen für diesen Schedule.

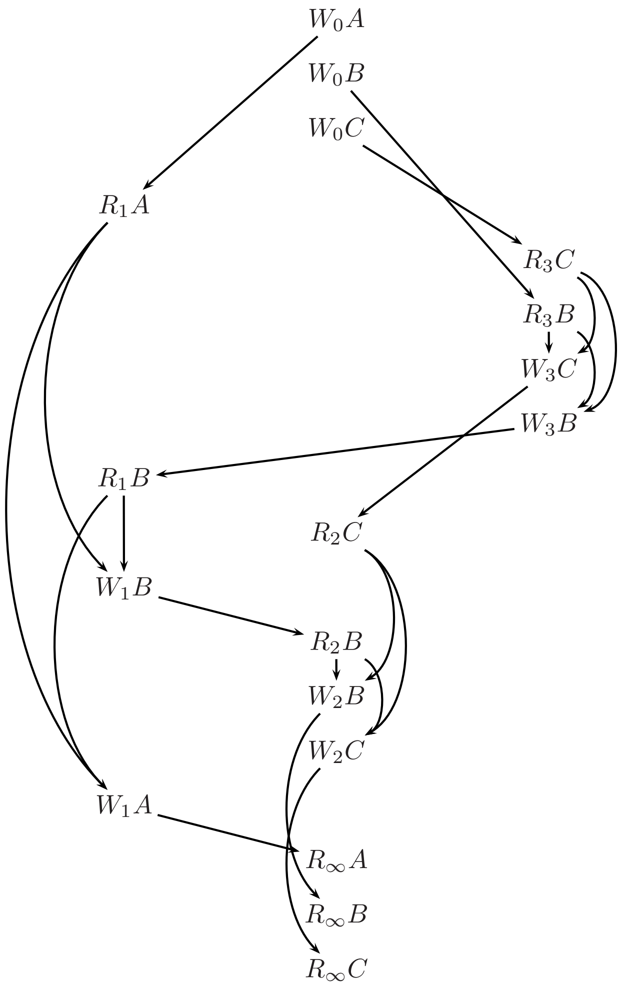
b) Überprüfen Sie mit Hilfe des C-Graphen, ob S serialisierbar ist. Wenn ja, zu welchem seriellen Schedule S' ist S äquivalent?

c) was ändert sich, wenn T_3 zuerst R_3A ausführt?

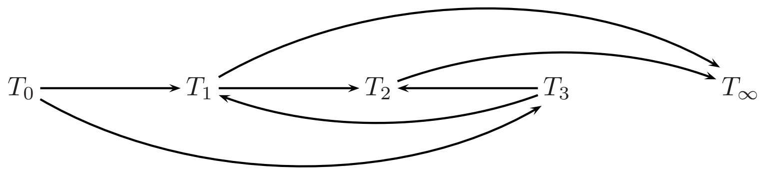
d) was ändert sich, wenn T_1 zuerst R_1C ausführt?

---

a) D-Graph:



C-Graph:



Der C-Graph enthält keinen Zyklus, damit ist der Schedule serialisierbar und äquivalent zu $T_3T_1T_2$, also:

$$R_3C\ R_3B\ W_3C\ W_3B\ R_1A\ R_1B\ W_1B\ W_1A\ R_2C\ R_2B\ W_2B\ W_2C$$

Wenn T_3 zuerst R_3A ausführt enthält der D-Graph zusätzliche Kanten von W_0A nach R_3A und von R_3A nach W_3C und W_3B. Ausserdem erhält man einen RW-Konflikt zwischen R_3A und W_1A - diese Kante ist aber im C-Graphen bereits erhalten, also keine Änderung.

Wenn T_1 zuerst R_1C ausführt enthält der D-Graph zusätzliche Kanten von W_0C nach R_1C und von R_1C nach W_1A und W_1B. Ausserdem hat man einen RW-Konflikt zwischen R_1C und W_3C und einer Kante $1 \to 3$ im C-Graphen. Damit ist der Schedule nicht serialisierbar.

Hinweis: Man kann das Problem im D-Graphen daran erkennen, dass der transaktionsinterne Datenfluss in T_1 von R_1C nach W_1A und W_1B das zwischenzeitliche Schreiben von W_3C "übersieht" – T_1 verwendet also einen alten Wert.

Aufgabe 5 (C-Graph-Serialisierbarkeit am Beispiel) Gegeben seien die folgenden drei Transaktionen:

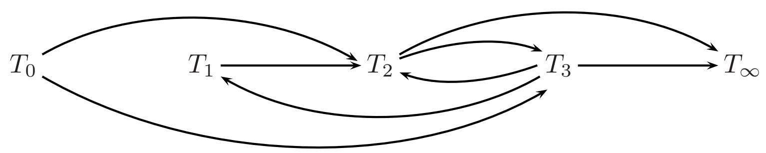
$$\begin{aligned} T_1 &= R_1A\ W_1A \\ T_2 &= W_2A\ R_2C\ W_2A \\ T_3 &= W_3A\ R_3C\ W_3C \end{aligned}$$

und der folgende zugehörige Schedule:

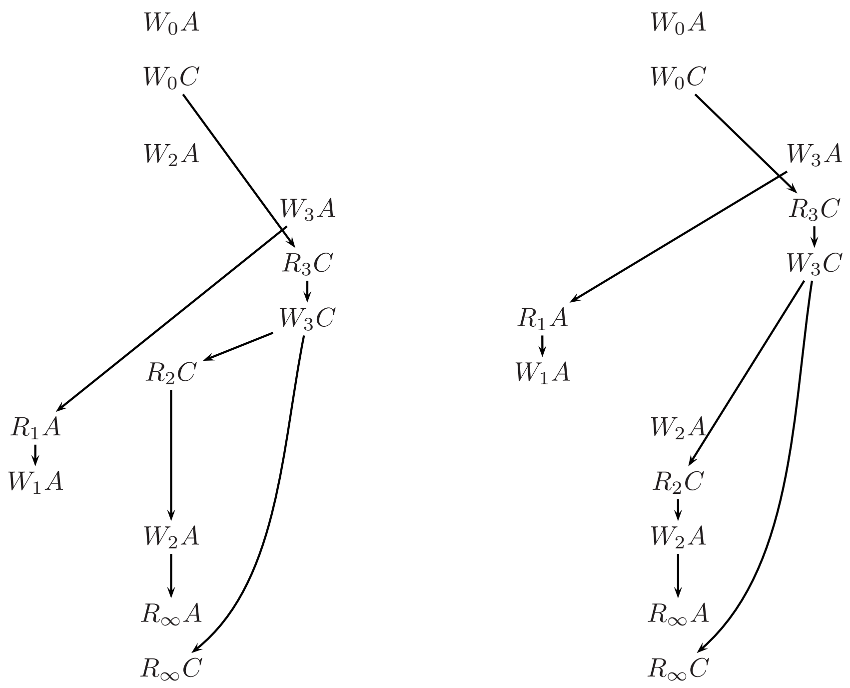
$$S = W_2A\ W_3A\ R_3C\ W_3C\ R_2C\ R_1A\ W_1A\ W_2A$$

Überprüfen Sie, ob aus dem C-Graphen dieses Schedules folgt, daß dieser Schedule serialisierbar ist.

C-Graph:



Der C-Graph enthält Zyklen, der Schedule ist jedoch serialisierbar und äquivalent zu T_3, T_1, T_2 (betrachte D-Graph).



Wie in der Vorlesung gezeigt, basiert diese Tatsache auf den verschiedenen “blind writes”, die den Datenfluss abschneiden.

Aufgabe 6 (Serialisierbarkeit (Beispiel)) Gegeben seien die Transaktionen $T_1 = RA\,WA\,WB$ und $T_2 = RA\ RB\ WA$. T_1 liest den Wert von A, erhöht diesen um 100 und schreibt den neuen Wert nach A zurück. Außerdem schreibt T_1 den alten Wert von A nach B (ohne B vorher gelesen zu haben - “blind write”). T_2 erhöht den gelesenen Wert von A um 200 und addiert den Wert von B hinzu. Der Anfangswert von A sei 10 und der von B sei 5. Betrachten Sie die folgenden Schedules:

$$\begin{aligned}
S_1 &= R_1A\ W_1A\ W_1B\ R_2A\ R_2B\ W_2A \\
S_2 &= R_2A\ R_2B\ W_2A\ R_1A\ W_1A\ W_1B \\
S_3 &= R_1A\ W_1A\ R_2A\ R_2B\ W_1B\ W_2A \\
S_4 &= R_1A\ R_2A\ R_2B\ W_1A\ W_1B\ W_2A \\
S_5 &= R_1A\ R_2A\ R_2B\ W_2A\ W_1A\ W_1B \\
S_6 &= R_1A\ W_1A\ R_2A\ W_1B\ R_2B\ W_2A
\end{aligned}$$

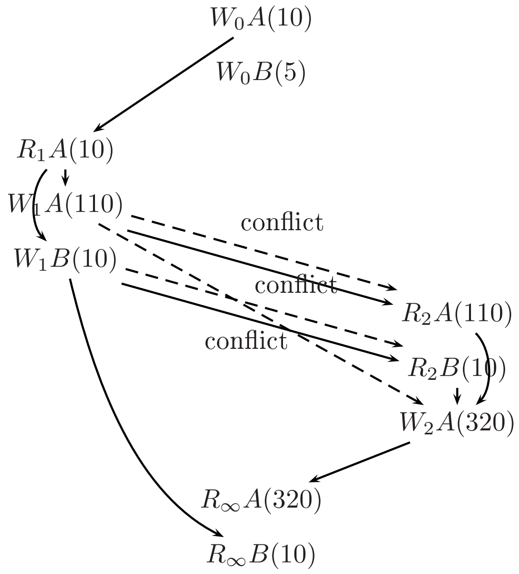
a) Ermitteln Sie für die einzelnen Schedules jeweils die Endwerte von A und B.

b) Welche Schedules sind seriell?

c) Welche Schedules sind serialisierbar?

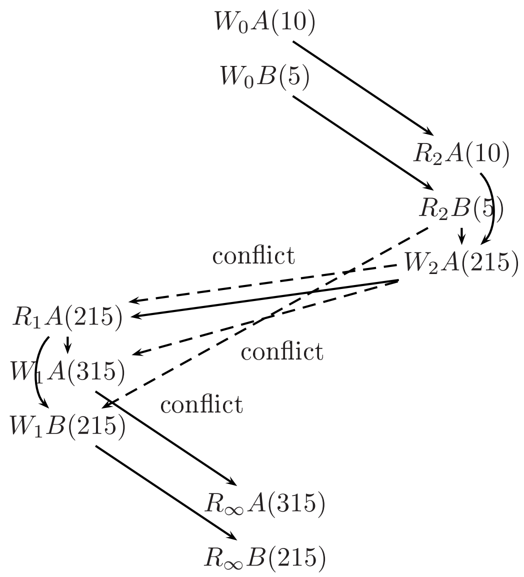
---

a) S_1 ist seriell:



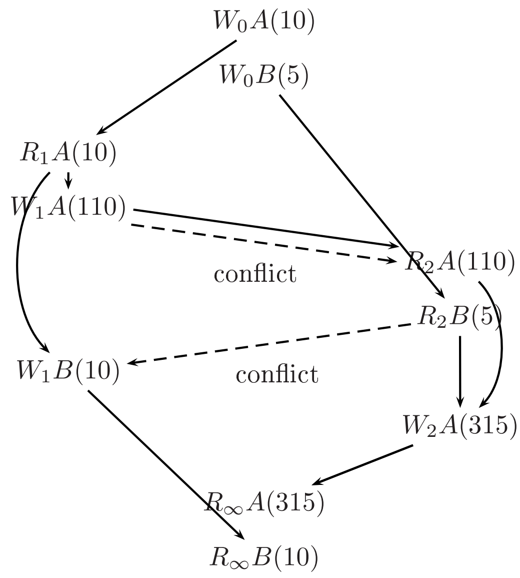
Alle Konflikte sind in dieselbe Richtung, und alle WR- und WW-Konflikte durch Datenfluss begleitet. Von W_0B geht keine Kante im D-Graphen aus, woraus zu ersehen ist, dass B durch ein “blind write” überschrieben wird.

b) S_2 ist seriell:



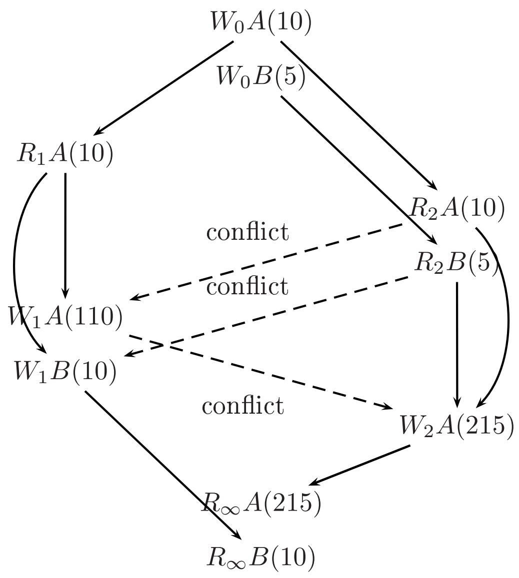
Alle Konflikte sind in dieselbe Richtung, und alle WR- und WW-Konflikte durch Datenfluss begleitet.

c) S_3 nicht serialisierbar, da T_2 bereits den neuen Wert von A, aber noch den alten Wert von B liest.
D-Graph:



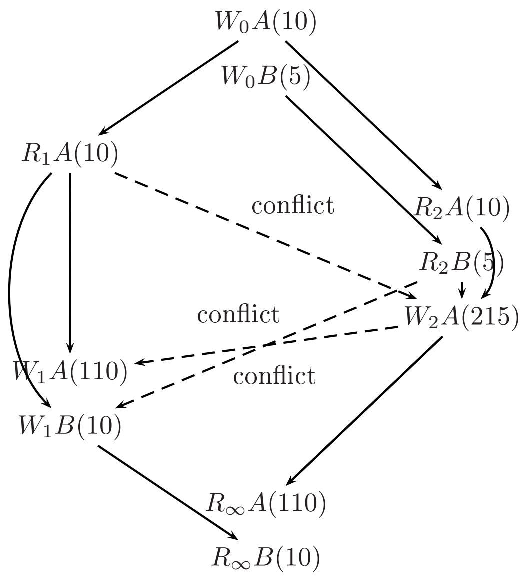
Der WR-Konflikt zwischen W_1A und R_2A wird von einer Kante im D-Graph begleitet, ist also problemlos. Der RW-Konflikt zwischen R_2B und W_1B geht in die falsche Richtung und zeigt, dass ein alter Wert gelesen wird.

d) S_4 nicht serialisierbar wegen "lost update" W_1A.
D-Graph:



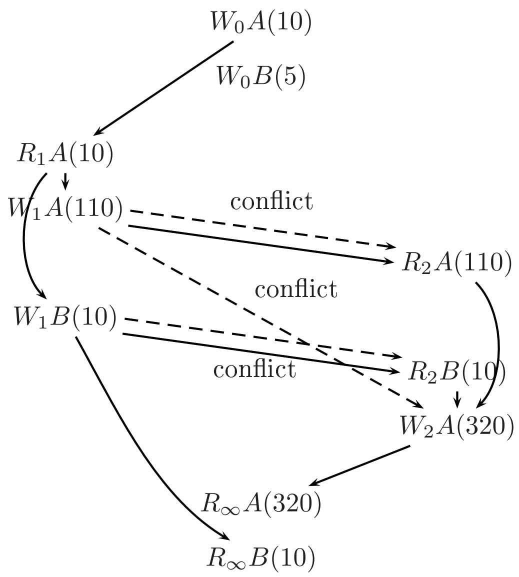
Man könnte jetzt W_2A vorziehen, und damit diesen Konflikt ebenfalls als $T_1 \to T_2$ ausrichten, fängt sich aber dann einen neuen ein (siehe S_5).

e) S_5 nicht serialisierbar wegen "lost update" W_2A.
D-Graph:



Das "lost update" W_2A äußert sich in dem WW-Konflikt zwischen W_2A und W_1A (direkt), sowie in dem RW-Konflikt zwischen R_1A und W_2A (der zeigt, dass T_1 A zu früh liest).
Wenn man jetzt auch den neuen RW-Konflikt ausrichtet, hat man den seriellen Schedule S_2.

f) S_6 ist serialisierbar (zu S_1):



Alle Konflikte sind in dieselbe Richtung, und alle WR- und WW-Konflikte durch Datenfluss begleitet.