Transaction Correctness

Enterprise Scale Data Management

Divy Agrawal
Department of Computer Science
University of California at Santa Barbara

Schedules and Histories

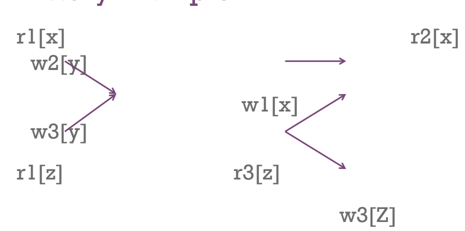
Definition 3.1 (Schedules and histories):
Let $T=\{t_1, \ldots, t_n\}$ be a set of transactions, where each $t_i \in T$ has the form $t_i=(op_i, <_i)$ with op_i denoting the operations of t_i and $<_i$ their ordering.

(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 $i \ne j$, all operations from t_i are ordered in s before all operations from t_j or vice versa.

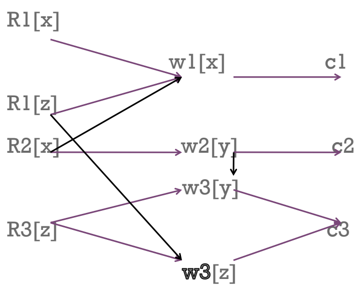
History Example



r1[x]r2[x]r1[z]w1[x]w2[y]r3[z]w3[y]w3[z]c1c2c3

History Example

4



Histories

5

- Without loss of generality:
  - Examples will be total orders
- Notations:
  - Trans(H): transactions in H
  - Commit(H): committed in H
  - Abort(H): aborted in H
  - Active(H): not committed and not aborted.

Correctness

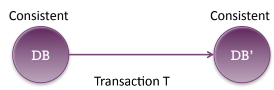
6

- Function $\sigma : S \rightarrow \{0,1\}$ such that correct(S)={s in S | σ(s)=1}
- Pragmatic considerations:
  - Correct(S)≠ϕ
  - Correct(S) is efficiently decidable
  - Correct(S) is sufficiently large (WHY?)
- Goal: develop several such criteria given that semantics not known.

Correctness

7

- Syntatctical semantics for schedules based on an intuitive notion:
  - Each transaction is a correct mapping, i.e.,



Hence, serial execution of transactions will be correct.

General Idea

8

- Notion of equivalence of two schedules S1 and S2

- Use this notion of equivalence to accept all schedules which are "equivalent" to some serial schedule as being correct.

- How to establish this equivalence notion?

Semantics

9

- Equivalence via a notion of semantics:
  - We do not know the semantics of transaction programs

  - We need a general notion that can capture all potential transaction semantics

→ Need a general enough and powerful notion that can capture all possible semantics of transactions.

+ Herbrand Semantics

10

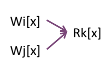
- Read operation ri[x] reads the last value by the last write that occurs before ri[x].
- Wi[x] writes a value that potentially depends on the value of all data items that Ti has read prior to wi[x].

+ Herbrand Semantics

11

- Abstract notion of semantics:
  1. ri[x] reads the last wj[x] (j≠i) before ri[x].
  2. Wi[x] depends on:
     1. Data from DB
     2. Transactions in ACTIVE U COMMIT prior to wi[x].

  Last write is well defined!!! Why?



  Assumption I: No transaction Aborts
  Assumption II: Initial Transactions:
  w0[entire-database], or equivalently
  w0[x, y, z, …]

+ Formal Definition: H-Semantics

12

- Hs(ri[x])=Hs(wj[x]) where wj[x] is the last write operation
- Hs(Wi[x])=fix(Hs(ri[y1]), …, Hs(ri[ym]))
- HU (Herbrand Universe) for transaction: what is conveyed to the transaction.
- HS for schedules: what is the permanent effect of the schedule of transactions.

Example

13

- S=w0[x]w0[y]c0r1[x]r2[y]w2[x]w1[y]c1c2
- Hs[w0[x]]=f0x()
- Hs[w0[y]]=f0y()
- Hs[r1[x]]=f0x()
- Hs[r2[y]]=f0y()
- Hs[w2[x]]=f2x(Hs[r2[y]])=f2x(f0y())
- Hs[w1[y])=f1y(Hs[r1[x]])=f1y(f0x())

Herbrand Universe

14

- Let D={x,y,z,…} be a finite set of data items. For a transaction T let op(T) denote all the steps of T. The HU of Ti is:
  - f0x() in HU for each x in D
  - If wi[x] in Op(Ti) then fix(v1, …, vm) in HU where vi are the values read by Ti before wi[x].

History Semantics

15

- H[h]: D → HU
- H[h](x) := Hs(wi[x])

Where wi(x) is the last operation in h writing x.

In other words – the semantics of a history h is the set of values that are written last in h.

Why are we doing all this?

16

- General/abstract notion of semantics.

- Can work with any interpretation of the transaction program, i.e., we do not have to worry about the program semantics of the transaction as to how they manipulate the data.

Example

17

- h=w0[x]w0[y]c0r1[x]r2[y]w2[x]w1[y]c2c1

- Hs[x]=Hs[w2[x]]=f2x(f0y())

- Hs[y]=Hs[w1[y]]=f1y(f0x())

Final State Equivalence

18

- S and S' over the same set of transactions then S is equivalent to S' if H(S)=H(S').

Example

19

- S = r1[x]r2[y]w1[y]r3[z]w3[z]r2[x]w2[z]w1[x]
- S'=r3[z]w3[z]r2[y]r2[x]w2[z]r1[x]w1[y]w1[x]
- H[S](x)=f1x(f0x())=H[S'](x)
- H[S](y)=f1y(f0x())=H[S'](y)
- H[S](z)=f2z(f0x(),f0y())=H[S'](z)

Another Example

20

- S=r1[x]r2[y]w1[y]w2[y]c1c2
- S'=r1[x]w1[y]r2[y]w2[y]c1c2
- H[S](y)=f2y(f0y())
- H[S'](y)=f2y(H[S'](r2[y]))=f2y(f1y(f0x()))

Observations

21

- Example shows that we cannot simply determine equivalence on final write operation.
- What preceded must also be taken into account.
  - In S: final value of y is based on initial value of y.
  - In S': final value of y is based on the value of y written by T1.
- Our task: can we build an efficient tool to determine equivalence ***efficiently***?

Reads-from Relation, Useful, Alive, and Dead Steps

22

- Rj[x] reads-x-from wi[x] if wi[x] is the last write such that wi[x] < rj[x].
- RF(S) = { (Ti, x, Tj) | rj[x] reads-x-from wi[x]}
- Step p is directly useful for q denoted p→q if:
  - Q reads-from P or
  - P is a read step and q is a subsequent write in the same transaction.
- →* is the transitive closure of →

Reads-from Relation, Useful, Alive, and Dead Steps

23

- P is alive in S if it is useful for some step in T∞:
  - Exists q in T∞ such that p→* q
  - Otherwise P is dead in S.
- Live reads-from relation:
  - LRF(S)={Ti, x, Tj) | rj[x] is alive and rj[x] in RF(S)}

Example

24

- S=w0[x,y]r1[x]r2[y]w1[y]w2[y]r∞[x,y]
- S'=w0[x,y]r1[x]w1[y]r2[y]w2[y]r∞[x,y]
- RF(S)={(T0,x,T1),(T0,y,T2),(T0,x,T∞),(T2,y,T∞)}
- RF(S')={(T0,x,T1),(T0,y,T2),(T0,x,T∞),(T2,y,T∞)}
- R2[y] alive in S and S' (verify)
- R1[X] dead in S but alive in S' (verify)

Example (contd.)

- LRF(S)={(T0,y,T2), (T0,x,T∞),(T2,y,T∞)}
- LRF(S')=RF(S')
- Redefine FSE: S and S' are final state equivalent if and only if LRF(S)=LRF(S') (Prove it – omitted).
- Build a tool that will allow to "efficiently" identify the LRF relations: STEP GRAPH.

Step Graph Construction

- Construct step graph D(S)=(V,E) where:
  - V=op(S)
  - E=[(p,q) | p,q in V and p → q]
- It can be shown that LRF(s)=LRF(s') iff D(s)=D(S').
- S f.s.e. S' iff D(S)=D(S') and op(S)=op(S')

Examples to check FSE using Step Graph

- S=r1[x]r2[y]w1[y]r3[z]w3[z]r2[x]w2[z]w1[x]
- S'=r3[z]w3[z]r2[y]r2[x]w3[z]r1[x]w1[y]w1[x]
- Construct D(S) and D(S') in class.

Another Example

28

- $S=r1[x]r2[y][w1[y]w2[y]$
- $S'=r1[x]w1[y]r2[y]w2[y]$

- Construct D(S) and D(S') to check FSE.

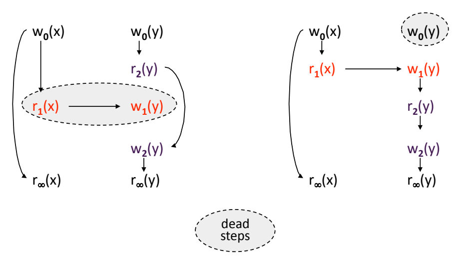
FSR: Example 3.9

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

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

D(s):

D(s'):



Transactional Information Systems 3-29 4/4/11

Testing for FSE

30

- FSE can be decided in time polynomial in the length of two schedules.

- FSR: A history is FSR if there exists a serial history S' such that S is FSE to S'.
- $S=r1[x]r2[y]w1[y]r3[z]w3[z]r2[x]w2[z]w1[x]$

Is equivalent to serial history T3-T2-T1 (verify)

Testing for FSR

31

- How to test for FSR:
  - Try all N! serialzations of N transactions.
- Not Efficient!!!
- More importantly: lets revisit our examples of Lost Update and Fund Transfer and see if it works from application point-of-view?

Lost Update

32

- History corresponding to lost update:
  - H=r1[x]r2[x]w1[x]w2[x]
  - Possible serializations: H1=r1[x]w1[x]r2[x]w2[x] OR H2=r2[x]w2[x]r1[x]w1[x]
- Construct D(H), D(H1) and D(H2) and see if this H is not FSE either to H1 or H2?

Fund Transfer

33

- Fund Transfer History:
  - H=r2[x]w2[x]r1[x]r1[y]r2[y]w2[y]
  - FSE to both T1-T2 and T2-T1.
- Even if we can develop an efficient tool to enforce FSR executions, it is not good enough for our purpose.

+ Key Insight

34

- We need to strengthen the notion of final state serializability:
  - By not only focusing on the state of the database
  - But also requiring that the "database view" observed by each transaction in the equivalent schedules is identical.

NEXT LECTURE.