Transaction Correctness

Enterprise Scale Data Management
Divy Agrawal
Department of Computer Science
University of California at Santa Barbara

Transaction Correctness

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 \( t_i = (\text{op}_i, <_i) \) with \( \text{op}_i \) denoting the operations of \( t_i \) and \( <_i \) their ordering.

(i) A history for \( T \) is a pair \( s = (\text{op}(s), <_s) \) s.t.
   - (a) \( \text{op}(s) \subseteq \bigcup_{i=1}^{n} \text{op}_i \cup \bigcup_{i=1}^{n} \{a_i, c_i\} \)
   - (b) for all \( i \), \( 1 \leq i \leq n \): \( c_i \in \text{op}(s) \iff a_i \notin \text{op}(s) \)
   - (c) \( \bigcup_{i=1}^{n} <_i \subseteq <_s \)
   - (d) for all \( i \), \( 1 \leq i \leq n \), and all \( p \in \text{op}_i \): \( p <_s c_i \) or \( p <_s a_i \)
   - (e) for all \( p, q \in \text{op}(s) \) 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.

History Example

\[
\begin{align*}
&\text{r1}[x] \\
&\text{w2}[x] \\
&\text{w3}[y] \\
&\text{r1}[z] \\
&\text{r3}[z] \\
&\text{w3}[Z] \\
&\text{r1}[x] \text{r2}[x] \text{r1[z]} \text{w1}[x] \text{w2}[y] \text{r3}[z] \text{w3}[y] \text{w3}[z] \text{c1} \text{c2} \text{c3}
\end{align*}
\]
History Example

\[ R1[w] \rightarrow w1[x] \rightarrow c_1 \]
\[ R1[z] \rightarrow w2[y] \rightarrow c_2 \]
\[ R2[x] \rightarrow w3[y] \rightarrow c_3 \]
\[ R3[z] \rightarrow w3[x] \]

Histories

- Without loss of generality:
  - Examples will be total orders

- Notations:
  - \( \text{Trans}(H) \): transactions in \( H \)
  - \( \text{Commit}(H) \): committed in \( H \)
  - \( \text{Abort}(H) \): aborted in \( H \)
  - \( \text{Active}(H) \): not committed and not aborted.

Correctness

- Function \( \sigma : S \rightarrow \{0, 1\} \) such that
  \( \text{correct}(S) = \{ s \in S \mid \sigma(s) = 1 \} \)

- Pragmatic considerations:
  - \( \text{Correct}(S) \neq \emptyset \)
  - \( \text{Correct}(S) \) is efficiently decidable
  - \( \text{Correct}(S) \) is sufficiently large (WHY?)

- Goal: develop several such criteria given that semantics not known.
Correctness

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

\[
\text{Consistent} \quad \rightarrow \quad \text{Consistent}
\]

Hence, serial execution of transactions will be correct.

General Idea

- Notion of equivalence of two schedules $S_1$ and $S_2$
  - 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

- 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

$\Rightarrow$ Need a general enough and powerful notion that can capture all possible semantics of transactions.
Herbrand Semantics

- Read operation $r_i[x]$ reads the last value by the last write that occurs before $r_i[x]$.
- $w_i[x]$ writes a value that potentially depends on the value of all data items that $T_i$ has read prior to $w_i[x]$.

Abstract notion of semantics:
1. $r_i[x]$ reads the last $w_j[x]$ ($j \neq i$) before $r_i[x]$.
2. $w_i[x]$ depends on:
   1. Data from DB
   2. Transactions in $\text{ACTIVE U COMMIT}$ prior to $w_i[x]$.

Last write is well defined!!! Why?

Assumption I: No transaction Aborts
Assumption II: Initial Transactions:
   $w_0[\text{entire-database}]$, or equivalently $w_0[x, y, z, ...]

Formal Definition: H-Semantics

- $H_s(r_i[x]) = H_s(w_j[x])$ where $w_j[x]$ is the last write operation
- $H_s(w_i[x]) = \text{fix}(H_s(r_i[y_1]), ..., H_s(r_i[y_m]))$

$H_U$ (Herbrand Universe) for transaction: what is conveyed to the transaction.

$H_S$ for schedules: what is the permanent effect of the schedule of transactions.
Example

- $S = w_0[x]w_0[y]c_0r_1[x]r_2[y]w_2[x]w_1[y]c_1c_2$
- $H_s[w_0[x]] = f_0x()$
- $H_s[w_0[y]] = f_0y()$
- $H_s[r_1[x]] = f_0x()$
- $H_s[r_2[y]] = f_0y()$
- $H_s[w_2[x]] = f_2x(H_s[r_2[y]]) = f_2x(f_0y())$
- $H_s[w_1[y]] = f_1y(H_s[r_1[x]]) = f_1y(f_0x())$

Herbrand Universe

- Let $D = \{x, y, z, \ldots\}$ be a finite set of data items.
- For a transaction $T$ let $\text{op}(T)$ denote all the steps of $T$. The HU of $T_i$ is:
  - $f_0x()$ in HU for each $x$ in $D$
  - If $w_i[x]$ in $\text{op}(T_i)$ then $\text{fix}(v_1, \ldots, v_m)$ in HU where $v_i$ are the values read by $T_i$ before $w_i[x]$.

History Semantics

- $H[h] : D \rightarrow \text{HU}$
- $H[h](x) := H_s(w_i[x])$
  Where $w_i(x)$ 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?
  - 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

  - \( h = w_0[x]w_0[y]c_0r_1[x]r_2[y]w_2[x]w_1[y]c_2c_1 \)

  - \( H_s[x] = H_s[w_2[x]] = f_2x(f_0y()) \)

  - \( H_s[y] = H_s[w_1[y]] = f_1y(f_0x()) \)

+ Final State Equivalence

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

- $S = r_1[x]r_2[y]w_1[y]r_3[z]w_3[z]r_2[x]w_2[z]w_1[x]$
- $S' = r_3[z]w_3[z]r_2[y]r_2[x]w_2[z]w_1[x]w_1[y]w_1[x]$
- $H[S](x) = f_1x(f_0x()) = H[S'](x)$
- $H[S](y) = f_1y(f_0x()) = H[S'](y)$
- $H[S](z) = f_2z(f_0x(), f_0y()) = H[S'](z)$

**Another Example**

- $S = r_1[x]r_2[y]w_1[y]w_2[y]c_1c_2$
- $S' = r_1[x]w_1[y]r_2[y]w_2[y]c_1c_2$
- $H[S](y) = f_2y(f_0y())$
- $H[S'](y) = f_2y(H[S'](r_2[y])) = f_2y(f_1y(f_0x()))$

**Observations**

- 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

- \( R_j[x] \) reads-x-from \( w_i[x] \) if \( w_i[x] \) is the last write such that \( w_i[x] < r_j[x] \).
- \( RF(S) = \{ (T_i, x, T_j) \mid r_j[x] \text{ reads-x-from } w_i[x] \} \)

- Step \( p \) is directly useful for \( q \) denoted \( p \rightarrow q \) if:
  - \( q \) reads-from \( p \)
  - \( p \) is a read step and \( q \) is a subsequent write in the same transaction.
- \( \rightarrow^* \) is the transitive closure of \( \rightarrow \)

P is alive in \( S \) if it is useful for some step in \( T^\infty \):
- Exists \( q \) in \( T^\infty \) such that \( p \rightarrow^* q \)
- Otherwise \( P \) is dead in \( S \).

Live reads-from relation:
- \( LRF(S) = \{ (T_i, x, T_j) \mid r_j[x] \text{ is alive and } r_j[x] \text{ in } RF(S) \} \)

Example
- \( S = w_0[x,y] ; r_1[x] ; r_2[y] ; w_1[y] ; w_2[y] ; r_\infty[x,y] \)
- \( S' = w_0[x,y] ; r_1[x] ; w_1[y] ; r_2[y] ; w_2[y] ; r_\infty[x,y] \)

- \( RF(S) = \{ (T_0, x, T_1), (T_0, y, T_2), (T_0, x, T^\infty), (T_2, y, T^\infty) \} \)
- \( RF(S') = \{ (T_0, x, T_1), (T_0, y, T_2), (T_0, x, T^\infty), (T_2, y, T^\infty) \} \)

- \( R_2[y] \) alive in \( S \) and \( S' \) (verify)
- \( R_1[X] \) 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 \rightarrow 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**

- $S = r_1[x] r_2[y] w_1[y] w_2[y]$
- $S' = r_1[x] w_1[y] r_2[y] w_2[y]$

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

---

**FSR: Example 3.9**

- $S = r_1(x) r_2(y) w_1(y) w_2(y)$
- $S' = r_1(x) w_1(y) r_2(y) w_2(y)$

---

**Testing for FSE**

- 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 $S'$.

- $S = r_1[x] r_2[y] w_1[y] r_3[z] w_3[z] r_2[x] w_2[z] w_1[x]$

- Is equivalent to serial history $T_3-T_2-T_1$ (verify)
Testing for FSR

- How to test for FSR:
  - Try all $N!$ serializations of $N$ transactions.

- Not Efficient!!!

- More importantly: let's revisit our examples of Lost Update and Fund Transfer and see if it works from application point-of-view?

Lost Update

- History corresponding to lost update:
  - $H = r_1[x]r_2[x]w_1[x]w_2[x]$  
  - Possible serializations: $H_1 = r_1[x]w_1[x]r_2[x]w_2[x]$  
    OR $H_2 = r_2[x]w_2[x]r_1[x]w_1[x]$

- Construct $D(H)$, $D(H_1)$ and $D(H_2)$ and see if this $H$ is not FSE either to $H_1$ or $H_2$?

Fund Transfer

- Fund Transfer History:
  - $H = r_2[x]w_2[x]r_1[x]r_1[y]r_2[y]w_2[y]$
  - FSE to both $T_1-T_2$ and $T_2-T_1$.

- Even if we can develop an efficient tool to enforce FSR executions, it is not good enough for our purpose.
Key Insight

- 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.