Run-time Analysis for Atomicity
(details of proof)

Liqiang Wang    Scott D. Stoller
{liqiang, stoller}@cs.sunysb.edu
Computer Science Dept. SUNY-Stony Brook

1. Determining Feasible Interleaving of Events

1.1 Proof of Correctness for the "No Potential for Deadlock" Method

**Definition:** Potential for Deadlock:
Two different transactions \( t \) and \( t' \) have potential for deadlock if they acquire two locks \( l_1 \) and \( l_2 \) in different order without first acquiring some other lock that would prevent their attempts to acquire \( l_1 \) and \( l_2 \) in different order.

**Theorem 1.** (Theorem 5.1 in our paper) Let \( e_1 \) and \( e_2 \) be events in a transaction \( t \), \( e_3 \) be an event in transaction \( t' \), thread\((t) \neq \text{thread\((t')\)}). \( \text{held\((e)\)} \) is the set of locks held by the transaction in which \( e \) occurs. Suppose \( t \) and \( t' \) do not have potential for deadlock, and all synchronization in \( t \) and \( t' \) are nested properly. A trace in which \( e_3 \) occurs between \( e_1 \) and \( e_2 \) is feasible iff \( h_{12} \cap \text{held\((e_3)\)} = \emptyset \), where \( h_{12} \) is the set of locks held continuously from before \( e_1 \) until after \( e_2 \).

Proof:
“\( \Rightarrow \)”: If \( h_{12} \cap \text{held\((e_3)\)} \neq \emptyset \), \( e_3 \) cannot occur between \( e_1 \) and \( e_2 \). Otherwise, i.e. if \( e_3 \) occurs between \( e_1 \) and \( e_2 \), \( \text{thread\((t)\) and \( \text{thread\((t')\)} \) would simultaneously hold same lock, obviously this is impossible.

“\( \Leftarrow \)”: Now, we prove if \( h_{12} \cap \text{held\((e_3)\)} = \emptyset \) and there is no potential for deadlock for \( t \) and \( t' \), \( e_3 \) can occur between \( e_1 \) and \( e_2 \).

If \( h_{12} = \emptyset \), or \( \text{held\((e_3)\)} = \emptyset \), so \( h_{12} \cap \text{held\((e_3)\)} = \emptyset \), there must be a trace in which \( e_3 \) occurs between \( e_1 \) and \( e_2 \). In the following proof, we assume \( h_{12} \neq \emptyset \) and \( \text{held\((e_3)\)} \neq \emptyset \).

Suppose the top level synchronization (lockset) of \( h_{12} \) is \( b_1 \), all events under the top level synchronization form a synchronization block, call it \( B_1 \). In the same way, \( i^{th} \) level synchronization is written as \( b_i \), the correspondent synchronization block is \( B_i \). Let \( M \) be the largest value of \( i \), i.e. the most-inner lock in \( h_{12} \) is \( b_M \). So \( h_{12} = \{b_1, \ldots, b_M\} \). Similarly, the \( j^{th} \) level synchronization on \( e_3 \) is written as \( b_j' \), the synchronization block is \( B_j' \), and \( \text{held\((e_3)\)} = \{b_1', \ldots, b_N'\} \), \( N \) is the largest value of \( j \), i.e. the most-inner lock in \( \text{held\((e_3)\)} \) is \( b_N' \). To determine
whether e₃ can occur between e₁ and e₂, we need to consider the interleaving of the events in B₁ and B₁’, events outside B₁ and B₁’ have no impact.

We construct a trace in which e₃ can occur between e₁ and e₂ as follows:

Firstly, find some position P between e₁ and e₂ which satisfies the following two conditions:
(i) thread(t) holds no other lock at P except for h₁₂.
(ii) Positioning e₃ at P does not create any lock conflict.
Because h₁₂ ∩ held(e₃) = ∅, this position must exist. Thus we can put e₃ to position P.

Secondly, Let \( \alpha = \{e_i | e_i \in B_1' \land e_i \text{ is before } e_3 \land \text{held}(e_i) \cap h_{12} \neq \emptyset \} \).

**Case 1:** \( \alpha = \emptyset \).
All events in t’ before e₃ can be put at the position P without any conflict. So e₁ is executed before e₃. By assumption there is no potential for deadlock, the remaining events in t and t’ can be executed in some order. Thus e₃ occurs between e₁ and e₂.

**Case 2:** \( \alpha \neq \emptyset \).
\( \forall e_i \in \alpha \), there is a sub-synchronization block Ai which satisfies the following conditions:
(i) \( e_i \in A_i \),
(ii) \( e_3 \notin A_i \),
(iii) let held_on(Aᵢ) be all locks held by Aᵢ (i.e. do not include inner locks of Aᵢ, and outer locks of Aᵢ), held_on(Aᵢ) ∩ h₁₂ ≠ ∅

Note for any \( e_i \in \alpha \), there could exist multiple such sub-synchronization blocks, and one such sub-synchronization block could include multiple elements of \( \alpha \).

Let \( \gamma \) be the set of all these sub-synchronization blocks. For all \( A_i, A_j \in \gamma \), if \( A_i \subseteq A_j \), remove such \( A_i \) from \( \gamma \); i.e. keep the outer synchronization block. Let the resulted set be \( \gamma' \). After this processing, any two sub-synchronization blocks in \( \gamma' \) are disjoint. For all elements in \( \gamma' \), according their events’ appearing sequence in t’, we rename the subscripts by continuous natural number, i.e. \( \gamma' = \{A'_1, A'_2, ..., A'_r, ..., A'_{E'}\} \). \( \forall A'_i \in \gamma' \), we have held_on(Aᵢ') ∩ h₁₂ ≠ ∅

Let held_inner(Aᵢ') be the whole locks held inside Aᵢ', held_outer(Aᵢ') be the whole locks held outside Aᵢ' (not on Aᵢ'). For all \( A'_i \in \gamma' \), let \( b_{k(i)} \) be the outmost lock in h₁₂ which is in held_on(Aᵢ') ∪ held_inner(Aᵢ'), i.e. \( b_{k(i)} \in \text{held_on}(A'_i) \cup \text{held_inner}(A'_i) \), and \( \text{held_on}(A'_i) \cup \text{held_inner}(A'_i) \cup \text{held_outer}(A'_i) \cap \{b_{k(i)} | L(i) \leq K(i)\} = \emptyset \). Such \( k(i) \) must exist because held_on(Aᵢ) ∩ h₁₂ ≠ ∅ for all \( A_i \) in \( \gamma \), and this \( A_i \) can include multiple elements in \( \gamma \). Let \( P_i \) be the position right before \( b_{k(i)} \) (i.e. the position right before acquiring lock operation, and after the previous event). Let \( \delta \) be the set of all these positions, \( \delta = \{P_1, P_2, ..., P_E\} \), and \( P_{E+1} = P \). If \( P_i \) is inside \( B_1 \), let \( P_0 \) be the position before \( B_1 \), \( \delta' = \{P_0\} \cup \delta \cup \{P_{E+1}\} \); otherwise \( \delta' = \delta \cup \{P_{E+1}\} \). Put all events in synchronization block Aᵢ' and all events between Aᵢ', Aᵢ'' of t’ at the position Pᵢ. (All events before Aᵢ' should be put at Pᵢ)

For all \( P_i \) and \( P_j \in \delta' \), and \( i < j \), let held_outer(Aᵢ', Aᵢ'') be the locks held by both Aᵢ' and Aᵢ'' (i.e. their outer locks). If there is an event e in t between \( P_i \) and \( P_j \), held(e) ∩ (held_outer(Aᵢ', Aᵢ''))
A_j') \setminus \text{held}(e_3)) \neq \emptyset$, then move all events at position $P_{i+1}, \ldots, P_j$ to position $P_i$. Thus any event of $t$ before $P$ does not conflict with any synchronization lock which is not $\text{held}(e_3)$.

So far, we get a trace in which $e_3$ can occur between $e_1$ and $e_2$. In this trace, by the construction way, it’s easy to know that any event of $t’$ before $e_3$ does not conflict with any synchronization locks in $t$. Next, we need to show any event of $t$ before $P$ does not conflict with $\text{held}(e_3)$.

By the construction way, for all $P_i, P_{i+1} \in \delta'$, any $e$ between $P_i$ and $P_{i+1}$ does not conflict with $\text{held}(e_3)$- $\text{held$\_outer\text{(}A_i\text{)}}$. We just need to consider the following two cases:

**Case 2.1** For all $P_i, P_{i+1} \in \delta'$, let $l_i$ be the locks of $\text{held}(e_3)$ outside $A_i’$, i.e. $l_i = \text{held}(e_3) \cap \text{held$\_outer\text{(}A_i\text{)}}$, any $e$ in $t$ between $P_i$ and $P_{i+1}$, $(\text{held}(e)\cap l_i \cap \text{held}(e_3)) \neq \emptyset$, i.e. no conflict occurs on all events before position $P_{i+1}$ (i.e. $P_i$). By the same reason in case 1, $e_3$ can occur between $e_1$ and $e_2$.

**Case 2.2** Otherwise, i.e. $\exists P_i, P_{i+1} \in \delta'$, let $l_i = \text{held}(e_3) \cap \text{held$\_outer\text{(}A_i\text{)}}$, there is some $e$ in $t$ between $P_i$ and $P_{i+1}$, $(\text{held}(e)\cap l_i \cap \text{held}(e_3)) \neq \emptyset$. Thus, $t$ and $t’$ would have different order in acquiring locks, i.e. $t$ first acquires $b_{K(i)}$, then $(\text{held}(e)\cap l_i \cap \text{held}(e_3))$; $t’$ first acquires $(\text{held}(e)\cap l_i \cap \text{held}(e_3))$. Further, because $\text{held$\_outer\text{(}A_i\text{)}} \cap \text{held$\_inner\text{(}A_i\text{)}} \neq \emptyset$, there is no lock which prevents such different order in acquiring locks, by the lemma deadlock will occur. But we have assumed that there is no potential for deadlock, so this case cannot exist.

So far, in the trace we constructed, for all $e$ of $t$ before $P$, and all $e’$ of $t’$ before $e_3$, they do not conflict on their current positions. So $e_3$ can occur between $e_1$ and $e_2$. □

1.2 Details and Correctness Proof for the "Depth Two" Method.

For now, we assume there are at most two-level locks held concurrently.

1. There is no synchronization operations between $e_1$ and $e_2$.

There is a top level synchronization on $e_1$ and $e_2$, all events inside this synchronization lock form a synchronization block called $B$. Similarly, the top level synchronization block on $e_3$ is $B’$. If there is no second-level synchronization on $e_1$ and $e_2$, or no second-level synchronization on $e_3$, the analysis is simpler. Now, we assume there are second-level synchronization on $e_1$, $e_2$ and $e_3$. 

3
A function \(\text{conflict}(B_i, B_j)\) is defined as:

\[
\text{conflict}(B_i, B_j) = \begin{cases} 
\text{true} & \text{if } \text{held}(B_i) \cap \text{held}(B_j) = \emptyset \\
\text{false} & \text{if } \text{held}(B_i) \cap \text{held}(B_j) \neq \emptyset 
\end{cases}
\]

Suppose \(e_1\) and \(e_2\) are in \(B_i\), \(e_3\) is in \(B_i'\):

1. If \(\text{conflict}(B, B') \lor \text{conflict}(B', B) \lor \text{conflict}(B, B') \lor \text{conflict}(B', B_i)\), then \(e_3\) cannot occur between \(e_1\) and \(e_2\).

2. There are synchronization operations between \(e_1\) and \(e_2\).
All analysis are same except that we do not need test conflict(B_i,B') and conflict(B_i, B'_i') in 1.1

2. Algorithm for Transactions that Access One Variable

2.1 Cases Analysis

Two blocks b1 and b2 are atomic iff all feasible interleaving of the four events in the two blocks do not have one or more of the following non-serializable scenarios.

A , B ∈ {b1,b2}, and A ≠ B. R is a read event, W is a write event, FW is the final write event in its transaction, MW is a middle write event (not final write) in its transaction.

Case 1: Another event of A can be before B.W1, after B.W2, or between B.W1 and B.W2 but after A.R.
A: R
B: W1 \ W2

Case 2: No requirement for another event of A
A: W
B: R1 \ R2

Case 3: No requirement for another event of A
A: W
B: W \ R

Case 4: No requirement for another event of A, except for satisfying A.FW is the final write in the transaction which A belongs
A: FW
B: R \ FW

Case 5: No requirement for another event of A, except for satisfying A.FW is the final write in the transaction which A belongs
A: FW
B: R \ MW

2.2 Prove the correctness of the algorithm

**Theorem 2.** (Theorem 5.2 in our paper) Let t and t' be transactions with thread(t) ≠ thread(t') and that access only one common variable. \{t, t’\} is atomic iff, for all blocks b for t and all blocks b’ for t’, b and b’ are atomic.

**Proof:**
“=>”: We will prove this direction by proving its contrapositive holds, i.e. if some blocks are not atomic, transactions $t$ and $t'$ are not atomic.

If some blocks are not atomic, there must be some case in 2.1 appears in a trace including these blocks. For example,

For case 5, trace $S$ is

$t$: $FW$
$t'$: $R$ $MW$

There are only two serial traces ($t$ $t'$) and ($t'$ $t$). For ($t$ $t'$), $t'.R$ reads some write of $t$, is not uninitialized as in $S$; for ($t'$ $t$), the final write is $t.FW$, not some write of $t'$ in $S$. There is no serial equivalent trace for $S$. Similar analysis exists for other non-atomic block pairs. So two transaction $t$ and $t'$ are not atomic.

“<=”:

For any non-serial trace $S$, suppose $t$ holds the final write of $S$ (if there is no write in $S$, obviously $S$ can be rescheduled into an equivalent serial trace).

(1) If $t'$ doesn’t read the final write of $t$, all events of $t'$ should be before $t$'s final write in $S$. It is like:

$t$: $...R...W...R...W$
$t'$: $W...R...$

Next, we will show $S$ is view equivalent to a serial trace ($t'$ $t$).

In $S$, any read event can be in only one of three cases: be uninitialized, reads a write in the same transaction, reads a write in another transaction.

(a) For all uninitialized reads of $t'$ under trace $S$, they are still uninitialized in trace ($t'$ $t$).

(b) For all reads of $t'$ which read write of $t$, they still read write of $t'$ in trace ($t'$ $t$).

(c) For all reads of $t'$ which read some writes of $t$, because these reads can not read $t$'s final write, there is the following scenario

\[
\begin{array}{c}
t & W & W \\
t' & R \\
\end{array}
\]

This is 2.1 case 1, there must be non-atomic block pair. So the kind of read can not exist.

By (a),(b),(c), we know ($t'$ $t$) is equivalent to $S$ for all reads of $t'$.

(d) For all uninitialized reads $t.R$ under trace $S$, if there is $t'.W$ between $t.R$ and $t.FW$ in $S$, $t'.FW$ must locate between $t.R$ and $t.FW$, because $t$ is supposed to have the final write in $S$.

$t$ $R$ $FW$
$t'$ $FW$

This is 2.1 case 4, there is non-atomic block pair. So either such read cannot exist, or such $t'.W$ cannot exist.

(e) For all reads of $t$ which read $t$'s write, they still read $t$'s write in trace ($t$ $t'$)

(f) For all reads of $t$ which read some write of $t'$, there are the following scenario,

\[
\begin{array}{c}
t & R & FW \\
\end{array}
\]
t’ W
There cannot exist another t’.W appears after t.R in S. Otherwise, there will be 2.1 case1. So all these read will read the same write in (t’ t) as in S.
By (d),(e),(f), we know (t’ t) is equivalent to S for all reads of t.

Because we have supposed that t holds the final write, so the final write is same in (t’ t) as in S. Thus, we proved that (t’ t) is equivalent to S in current condition.

(2). If t’ reads the final write of t, for example
   t: .... FW R
   t’: ... R

   t’ can not have R/W before the final write of t in trace S. Otherwise, suppose t’ has such a R which reads t.FW (t’ can not have such a W, because t holds the final write in S).
   There will be scenario:
   t: ... FW
   t’ R R
   Or
   t: ... FW
   t’ W R

   They are 2.1 case2, 2.1 case3, respectively. So (t t’) is serializable with S.

A trace equivalent to S is got. So t and t’ are atomic if all pairs of blocks are atomic. □

Theorem 3. (Theorem 5.3 in our paper) A set of T of transactions that access only one common variable is a atomic iff every subset of T with cardinality two is atomic.

Proof:
Suppose there are k transactions, T={t1, t2, …tk}

“=>”:
We need to prove if there are two non-atomic transactions in T, T is not atomic. Suppose the two non-atomic transactions form a non-serializable sub-trace S1, all the other transactions in T can be scheduled sequentially into a serial sub-trace S2, a trace S can be got by putting S2 and S1 together, i.e. (S2 S1). Obviously S is non-serializable. So T is not atomic.

“<=”:
For any non-serial trace S of T, suppose ti holds the final write of S (if there is no write in S, obviously S can be rescheduled into an equivalent serial trace), because ti is atomic with all other transactions and has the final write, we can move all operations of ti to the end. Next, the situation can be parted into two cases:
   (1). If there is no transaction which reads the final write of ti, because ti is atomic with all other transactions, the trace after moving ti to end is equivalent to S.
(2). If there are some transactions which read the final write of \( t_i \), call all these transactions set D. For any \( t_j \) of them, according to the analysis of non-atomic cases, \( t_j \) cannot have Read or Write before the final write of \( t_i \) in S. So we can move \( t_i \) and all transactions in D to the end. Thus, because all blocks of D are read and all blocks in T are atomic, so the result trace is view equivalent to S.

Next, remove D and \( t_i \) from T, remove all operations of D and \( t_i \) from S, i.e. \( T_1 = T - D - \{ t_i \} \), \( S_1 = S - \{ \text{op} \mid \text{op} \text{ in } t_i \} - \{ \text{op} \mid \text{op} \text{ in some transaction in } D \} \), continue the above procedure on \( T_1 \) and \( S_1 \), until the left operations are traced in serial way.

At final, we get a trace equivalent to S, i.e. all transactions in T are atomic. □

3. Algorithm for Two transactions that access multiple variables

3.1 Case Analysis
Two 2-block a and b come from transaction A and B respectively, the shared variables between them are \( x \) and \( y \). Two events conflict if they access the same variable and at least one of them is write.

Case 1. No conflict event, i.e all four events in two 2-blocks are reads. Obviously, any feasible interleaving is serializable.

Case 2. Only one pair of conflict events. The pair of events which don’t conflict have no effect on atomicity of a and b, we need to consider only the conflict events pair. It is easy to know that any pair of conflict events is atomic.

Case 3. Two pairs of conflict events, all of following cases are non-serializable.
Case 3.1 No read. The following interleavings are not serializable, because all of them cannot preserve the same final write in any serial trace as in current interleaving scenario.

(1)
\[
\begin{align*}
\text{a: } & W(x) \quad W(y) \\
\text{b: } & W(x) \quad W(y)
\end{align*}
\]

(2)
\[
\begin{align*}
\text{a: } & W(x) \quad W(y) \\
\text{b: } & W(y) \quad W(x)
\end{align*}
\]

(3)
\[
\begin{align*}
\text{a: } & W(x) \quad W(y) \\
\text{b: } & W(y) \quad W(x)
\end{align*}
\]
(4)
a:  W(x)   W(y)
b:  W(y)   W(x)

It’s clear that the relationship is symmetric on a & b, and x & y. So other redundant cases are not listed here.

Case 3.2 One read. All of the following interleavings are non-serializable, because a.R(x) can not read same write in current trace as in a serial trace. a.R(x) is the first uninitialized read in A. As in case 3.1, a & b, and x & y are symmetric, redundant cases are not listed here.

(1)
a:  R(x)   W(y)
b:  W(x)   W(y)

(2)
a:  R(x)   W(y)
b:  W(y)   W(x)

(3)
a:  R(x)   W(y)
b:  W(x)   W(y)

(4)
a:  R(x)   W(y)
b:  W(y)   W(x)

(5)
a:  R(x)   W(y)
b:  W(y)   W(x)

(6)
a:  R(x)   W(y)
b:  W(y)   W(x)

If the second variable in some block is read, there are six similar cases, for convenience, they are not listed here.

Case 3.3 two reads in one block, none in the other. All of the following and symmetric variants are non-serializable.

(1)
a:  R(x)       R(y)
b: \( W(x) \ W(y) \)

(2)
\begin{align*}
a: & \ R(x) \quad \ R(y) \\
b: & \ W(y) \ W(x) \\
\end{align*}

(3)
\begin{align*}
a: & \ R(x) \quad \ R(y) \\
b: & \ W(x) \quad \ W(y) \\
\end{align*}

(4)
\begin{align*}
a: & \ R(x) \quad \ R(y) \\
b: & \ W(y) \quad \ W(x) \\
\end{align*}

(5)
\begin{align*}
a: & \ R(x) \quad \ R(y) \\
b: & \ W(y) \quad \ W(x) \\
\end{align*}

(6)
\begin{align*}
a: & \ R(x) \quad \ R(y) \\
b: & \ W(y) \quad \ W(x) \\
\end{align*}

Case 3.4 One read in each block, all of the following are non-serializable.

(1)
\begin{align*}
a: & \ R(x) \quad \ W(y) \\
b: & \ W(x) \quad \ R(y) \\
\end{align*}

(2)
\begin{align*}
a: & \ R(x) \quad \ W(y) \\
b: & \ R(y) \quad \ W(x) \\
\end{align*}

(3)
\begin{align*}
a: & \ R(x) \quad \ W(y) \\
b: & \ W(x) \quad \ R(y) \\
\end{align*}

(4)
\begin{align*}
a: & \ R(x) \quad \ W(y) \\
b: & \ R(y) \quad \ W(x) \\
\end{align*}

(5)
\begin{align*}
a: & \ R(x) \quad \ W(y) \\
b: & \ W(y) \quad \ W(x) \\
\end{align*}
b: R(y)  R(x)

(6)
a: R(x)  W(y)
b: R(y)  W(x)

3.2 Correctness of the algorithm

**Theorem 4.** (Theorem 5.4 in our paper) Let A and B be transactions with thread(A) ≠ thread(B). [A, B] is atomic iff (i) for all blocks a for A and all block b for B, a and b are atomic; (ii) for all 2-blocks a for A and all 2-blocks b for B, a and b are also atomic.

**Proof:**

"=>" For 1-blocks (the block in the algorithm for transactions that access one variable), we have proved it in Theorem 1. As in Theorem 2, we prove Theorem 4 by contrapositive, i.e. if a 2-block in A is not atomic with a 2-block in B, A and B are not atomic.

*If there are two non-atomic 2-blocks, a(e1,e2) is a block in A, b(e3,e4) is a block in B. Because a and b are non-atomic, according the analysis in 3.1, there must be two pairs of conflict events, suppose e1 conflicts with e3, e2 conflicts with e4. In the non-atomic interleaving cases, the order of (e1,e3) must be reverse to the order (e2,e4). For example,

A: e1  e2
B:  e3   e4

According the definition of 2-block, all writes are the final writes for correspondent variables in its transaction. All reads have no previous write on correspondent variables in their transactions. So there is no serial equivalent trace for any non-atomic blocks interleaving in 3.1.

"<=" Let α be the set of variables accessed by both A and B. To prove any trace S has an equivalent serial trace S’, we need to prove all read events read the same values in S’ as in S, the final writes in S’ on all variables are also the final writes in S.

**Case 1:** There exists a variable x in α, such that both A and B have a write on x. Suppose B holds the final write in S. Next, we will show S is equivalent to a serial trace (A B).

(a) We need to show A can not read B’s write to any variable.

  We prove it by contradiction.

  For variable y = x, if A reads B’s write to x, we have

  A: W(x)  R(x)

* Actually, according the analysis of cases in 3.1, it’s obvious that these non-atomic block pair imply two transactions are not atomic. The proof could be regarded as a summary of the cases.
B: \ W(x)
A.W(x) must occur before B.W(x), because B.W(x) is the final write to x in S. Thus, A.(x) and A.R(x) form a 1-block that is not atomic with B.W(x).

For any variable y \in \alpha - \{x\}, suppose A.R(y) reads B.W(y). The scenario could like the following. *

A: \ \{ R(y) \ W(x) \}  
B: \ W(y) \ W(x)
So A.W(x) is before B.W(x), A.R(y) is after B.W(y). Now we show B.W(y) is the final write on y in B, there is no A.W(y) before A.R(y) in A, and A.R(y) is the first uninitialized read for y in A. Thus the four events form two 2-blocks in A and B.

If there is another write B.W'(y) after B.W(y), it must be after A.R(y) because A.R(y) reads B.W(y). So B.W(y) and B.W'(y) form a 1-block which is not atomic with A.R(y). If there is A.W(y) before A.R(y), then it must precede B.W(y), so A.W(y) and A.R(y) also form a 1-block which is not atomic with B.W(y). If there is another A.R'(y) before A.R(y), A.R'(y) can not locate before B.W(y) in S because A.R'(y) and A.R(y) would form a 1-block non-atomic with B.W(y), so A.R'(y) can only sit together with A.R(y). After replacing A.R(y) by A.R'(y), the scenario remains same. This also is reason that we use the first uninitialized read to form 2-block.

Thus for y=\neq x, 2-block (A.R(y), A.W(x)) or (A.W(x), A.R(y)), and (B.W(y),B.W(x)) or (B.W(x), B.W(y)) must exist, and the two 2-blocks are not atomic. It contradicts with the premise. So A.R(y) can not read B.W(y).

(b) Next, we show if B reads A’s write in S, B will read the same written value in S'.
(i) we show any read in B, such as R(y), if reads A.W(y), there is no B.W(y) before A.W(y) in S.
If it exists, there will be non-atomic 1-blocks.
A: \ W(y)  
B: \ W(y) \ R(y)

(ii) Then, we show any read in B, such as R(y), if reads A.W(y), this A.W(y) must be the final write for y in A. Otherwise, there will be non-atomic 1-blocks.
A: \ W(y) \ W(y)  
B: \ R(y)
So S is equivalent to (A B).

(c) Now, we show \forall y \in \alpha, if there are writes on y in both transactions A and B. the final write(y) in B must locate after the final write(y) in A in the trace S. For example,
A: \ W(x) \ W(y)  
B: \ W(x) \ W(y)
Otherwise, the scenario is like:

* This is just one of scenarios, note that A.W(x) may occur at any point before B.W(x), and A.W(x) may occur before A.R(y), B.W(x) may precede B.W(y). \{R(y) W(x)\} means either order, i.e. (R(y) W(x)) or (R(y) W(x))
A: \( W(x) \) \hspace{1cm} W(y) \\
B: \hspace{1cm} W(x) \hspace{1cm} W(y) \\

because all writes are the final writes on the correspondent variables in their transactions, two 2-blocks must be detected to be non-atomic in this interleaving.

**Case 2.** \( \neg (\exists x \in \alpha, \text{A and B both have a write on } x) \)

**Case 2.1** \( \exists x \in \alpha, \text{A has a write on } x, \text{xor B has a write on } x. \)

Suppose A has a write on x in S, i.e. A has W(x), B has R(x) (if B doesn’t have R(x), then x would not be in \( \alpha \), a contradiction occurs). Note that B.R(x) must read A.W(x) or the initial value of x.

(a) Now, we show if B.R(x) reads A.W(x), S is equivalent to (A B).

(i). Firstly, we show there cannot exist A.R(y) that reads B.W(y) for all \( y \in \alpha \).

\( y = x \) does not hold because B cannot write x by definition of case 2.1.

for \( y \neq x \), we would have the following example

A: \( W(x) \) \hspace{1cm} R(y) \\
B: \hspace{1cm} \{R(x) \hspace{0.5cm} W(y)\} \\
Or \\
A: \hspace{1cm} \{R(y) \hspace{0.5cm} W(x)\} \\
B: \hspace{1cm} W(y) \hspace{1cm} R(x) \\

If this interleaving exists, A.W(x) must be the final write to x in transaction A, and B.W(y) must be the last write to y in B, and A.R(y) and B.R(x) must be uninitialized reads in A and B. Otherwise, there will be non-atomic 1-blocks. But under this interleaving, the two 2-blocks A.(W(x), R(y)) or A.(R(y), W(x)) and B.(R(x), W(y)) or B.(W(y),R(x)) are non-atomic. This contradicts with the premise.

(ii) Then, we show any read in B, such as R(y), if it reads A.W(y), there is no B.W(y) before A.W(y) in S.

If this B.W(y) exists, there will be non-atomic 1-blocks.

A: \( W(y) \) \\
B: \hspace{1cm} W(y) \hspace{1cm} R(y) \\

(iii) Finally, we show any other read in B, such as R(y), if read A.W(y), this A.W(y) must be the final write for y in A.

If A.W(y) is not the final write, there will be non-atomic 1-block.

A: \( W(y) \) \hspace{1cm} W(y) \\
B: \hspace{1cm} R(y) \\

According to (i),(ii) and (iii), we know S is view equivalent to (A B)

(b) Now, we show if no B.R(x) read A.W(x), S == (B A)
By the same way in (a), we can prove, for any y in alpha,
(i). there is no B.R(y) to read A.W(y)
(ii). If A.R(y) read B.W(y), there is no A.W(y) before B.W(y) in S.
(iii). If A.R(y) read B.W(y), this B.W(y) is the final write for y in B.

The final writes are the same in S and in (B A), because no variable is written by both transactions. So S is view equivalent to (B A)

Case 2.2. $\neg (\exists x \in \alpha, A \text{ has a write on } x, \text{xor } B \text{ has a write on } x).
In this case, all operations on shared variables are reads, obviously the transactions are atomic. □

4. General Algorithm

**Theorem 5.** Let $T$ be a group of transaction, $\forall t_1, t_2 \in T, \text{ thread}(t_1) \neq \text{thread}(t_2)$. For any $t$ of $T$, let $t^{rd}_t$ be its reduced transaction, i.e. keep uninitialized reads and final writes, remove all other read and write events (call this URFW_Reduction). Let $T^{rd}$ be the set of all reduced transactions. $T$ is atomic iff (i) any two transactions of $T$ are atomic, and (ii) every interleaving of events in $T^{rd}$ is serializable.

**Proof:**

$\Rightarrow$: If $T$ is atomic:
(i). According to the same reason in the proof of Theorem 3, any two transactions of $T$ are atomic.
(ii). Because $T$ is atomic, any interleaving of $T$ can be rescheduled into a serial trace, obviously every interleaving of events in $T^{rd}$ is serializable.

$\Leftarrow$: We need to prove there is an equivalent serial trace for any trace $S$ of all events in $T$.
Suppose $x$ be any shared variable.
(1). If some $t_i, t_j \in T$, $t_j.R_i(x)$ reads $t_i.W_i(x)$, i.e.
   $t_i: W_i(x)$
   $t_j: R_j(x)$
Because $t_i$ and $t_j$ are atomic, No $W(x)$ of $t_j$ can locate before $t_j.R_i(x)$ in S, and no $W(x)$ in $t_i$ can locate after $t_i.W_i(x)$, so $t_j.R_i(x) \in UR(t_j), t_i.W_i(x) \in FW(t_i)$. Let $S^{rd}_s$ be the trace after executing URFW_Reduction operation on S, $t_j.R_i(x)$ also reads $t_i.W_i(x)$ in $S^{rd}$.

Because every interleaving of events in $T^{rd}$ is serializable, let $S_{sr}^{rd}$ be the equivalent serial trace of $S^{rd}$, so $t_j.R_i(x)$ reads $t_i.W_i(x)$ in both $S^{rd}$ be $S_{sr}^{rd}$. According the appearing position of each transaction in $S_{sr}^{rd}$, we can reschedule S to get a serial trace $S_{sr}$ where $t_j.R_i(x)$ also reads $t_i.W_i(x)$. Thus, $t_j.R_i(x)$ reads $t_i.W_i(x)$ in both S be $S_{sr}$.
(2). Suppose $t_k$ holds the last write $t_k.W(x)$ on $x$ in $S$, $t_k.W(x)$ is also the last write on $x$ in $S_{rd}$. Because $S_{rd}$ and $S_{sr}^{rd}$ are equivalent, $t_k.W(x)$ is the last write on $x$ in $S_{sr}^{rd}$. Since the appearing positions of transactions in $S_{sr}^{rd}$ are same as in $S_{sr}$, $t_k.W(x)$ is also the last write on $x$ in $S_{sr}$.

By the above (1) and (2), for any shared variable $x$, and any trace $S$ of events in $T$, any read($x$) in $S$ reads the same write both in $S$ and $S_{sr}$, and the final write on $x$ in $S$ is also the final write in $S_{sr}$, so $S$ is equivalent to the serial trace $S_{sr}$.

$\Box$