A Programming Language Perspective on Transactional Memory Consistency

Hagit Attiya, Technion

Joint work with Sandeep Hans, Alexey Gotsman, Noam Rinetzky

Research funded by the European Union's 7th Framework Programme (FP7/2007-2013) under grant agreement n° 238639 – MC-TRANSFORM.
Transactional Memory (TM)

- **Atomic blocks** (accessing transactional variables)
  - Appear to execute atomically
  - May abort
- **Local variables** (also inside blocks)
- **Global variables** (only outside blocks)

```java
node := new(StackNode);
node.val := val;
result := abort;
while(result == abort){
    result := atomic{
        node.next = Top.read();
        node.val++ ;
        Top = node;
    }
}
g := val;
```
TM Consistency Condition

- How should the TM implementation behave?
- No single answer...
TM Consistency Conditions

Opacity
[Guerraoui & Kapalka 08]

Virtual World Consistency
[Imbs & Raynal 09]

TMS1 / TMS2
[Doherty, Groves, Luchangco, Moir 09]

What is the “right” condition?

But we are not social scientists
Observational Refinement

Preserve the observations of a client program, when an abstract library implementation is substituted with a concrete one

Our work uses observational refinement as a yardstick to evaluate TM consistency conditions
Interactions of a Program using TM

- **Local actions:** access only the local variables
- **Global actions:** interact with other client programs
- **Interface actions:** interact with TM

```plaintext
node := new(StackNode);
node.val := val;
result := abort;
while(result == abort){
    result := atomic{
        node.next =
        Top.read();
        node.val++;
        Top = node;
    }
}
g := val;
```
Interactions of a Program using TM

**History**: Finite sequence of interface actions

```
node := new(StackNode);
node.val := val;
result := abort;
while(result == abort){
    result := atomic{
        node.next = Top.read();
        node.val++ ;
        Top = node;
    }
}
g := val;
```

**Transactional System (TM)**: set of histories
Trace: includes also local and global actions

Client Program

```
node := new(StackNode);
node.val := val;
result := abort;
while(result == abort){
    result := atomic{
        node.next =
        Top.read();
        node.val++;
        Top = node;
    }
}
g := val;
```

TM

txbegin
OK call
ret / abort

txcommit
ret / abort

val:=8
val:=9
g:=7
val:=3

More than Just TM
Trace Equivalence

**Trace:** includes also local and global actions

Two traces are *observationally equivalent* $\tau \sim \tau'$ if threads see the same sequence of local values

\[\text{val}:=8 \quad \text{val}:=9 \quad \text{g}:=7 \quad \text{val}:=3\]

$\mathcal{T}_C$ *observationally refines* $\mathcal{T}_A$ if every trace $\tau$ with history in $\mathcal{T}_C$ has a trace $\tau' \sim \tau$ with history in $\mathcal{T}_A$
Why Observational Refinement?

Prove properties of $\text{TM}_A$ and deduce same properties for $\text{TM}_C$

$\text{TM}_C$ observationally refines $\text{TM}_A$ if every trace $\tau$ with history in $\text{TM}_C$ has a trace $\tau' \sim \tau$ with history in $\text{TM}_A$
Abstract System for Opacity

**Complete** history: all transactions commit / abort

Sequential history: no interleaving of transactions

Legal history: read from committed transactions

\[ \text{TM}_{\text{ATOMIC}} : \text{all sequential and legal histories} \]
Opacity

History $H$ is **opaque** if we can

- Complete $H$
- Find a permutation $S$ of $H$ that is **sequential, legal** and preserves the **real-time order** of $H$

(TM is **opaque** if every history in TM is opaque)

[Guerraoui & Kapalka 08]
Opacity Relation

\[ H \subseteq S \]

S preserves the **per-thread** and **real-time** order of H

\[ \text{TMC} \subseteq \text{TM}_A \]

for every \( H \in \text{TMC}, \ H \subseteq S, \) for some \( S \in \text{TM}_A \)

\[ \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \text{completion of } H \]

- sequential, legal
- preserves real-time order

\[ \text{TM}_C \text{ is opaque } \iff \text{TM}_C \subseteq \text{TM}_{\text{ATOMIC}} \]
Opacity Relation vs. Linearizability

Linearizability: consistency condition for library calls
  – client is suspended when waiting for a response

Observational refinement for linearizability

[Filipovic, O'Hearn, Rinetzky, Yang 09]

For opacity relation, we need to do more...
Main Result

$\text{TMC} \subseteq \text{TMA} \iff \text{TMC} \text{ observationally refines TMA}$

- global variables can be accessed, but only outside atomic blocks

```c
result := atomic{
    g := 1 ;
    read(tx) ;
}
```

(returns 1)

```c
result := atomic{
    l := g ;
    write(1,tx) ;
}
```

(returns 1)
Main Result

$\text{TMC} \subseteq \text{TM}_A \iff \text{TMC} \text{ observationally refines } \text{TM}_A$

- global variables can be accessed, but only outside atomic blocks
- finite histories
- no nesting of atomic blocks
Soundness: $\sqsubseteq$ is Sufficient

Assume $\text{TM}_C \sqsubseteq \text{TMA}$ and prove that a trace $\tau$ observed with $\text{TM}_C$ has an equivalent trace $\tau'$ observed with $\text{TMA}$.

- Consider a trace $\tau$ whose history $H$ is in $\text{TMC}$.
- $\text{TMC} \sqsubseteq \text{TMA} \Rightarrow H \sqsubseteq S$ for some history $S$ in $\text{TMA}$.

From $\tau$ and $S$, get a trace $\tau' \sim \tau$ of $\text{TMA}$ whose history is $S$. 
Soundness: $\equiv$ is Sufficient

Assume $\text{TMC} \equiv \text{TMA}$ and prove that a trace $\tau$ observed with $\text{TMC}$ has an equivalent trace $\tau'$ observed with $\text{TMA}$.

$\text{H} \equiv \text{S}$

- $\text{H}$: history $\in \text{TMC}$
- $\text{S}$: history $\in \text{TMA}$

$\tau \sim \tau'$

- Two traces are equivalent if they have the same order for same-thread actions and for global actions.
Soundness: $\equiv$ is Sufficient

Assume $\text{TMC} \equiv \text{TMA}$ and prove that a trace $\tau$ observed with $\text{TMC}$ has an equivalent trace $\tau'$ observed with $\text{TMA}$.

\[ \text{H} \quad \equiv \quad \text{S} \]

\[ \text{trace} \quad \sim \quad \text{trace} \]

💡 Inductively **permute** $\tau$ to get $\tau'$, while **preserving the order of same-thread and global actions**
Soundness: Inductive Step

Assume we have permuted a prefix of $\tau$ so its history is a prefix of a history in $\text{TMA}_A$

$\text{Locate } \phi, \text{ the next interface action in } S, \text{ and move it}$

$\text{Reordering } \phi \text{ relative to earlier actions may violate equivalence}$
Soundness: Inductive Step

Assume we have permuted a prefix of $\tau$ so its history is a prefix of a history in $\text{TM}_A$

\[
\begin{align*}
\text{history} & \in \text{TM}_C \\
\text{trace} & \sim \\
\text{Locate } \phi, \text{ the next interface action in } S, \\
\text{Reordering } \phi \text{ relative to earlier actions may violate equivalence}
\end{align*}
\]
Inductive Step: Case 1

\( \phi \neq \text{txbegin} \) by thread \( t \)

- **Only local actions by** \( t \)
- History \( S \in TM_A \)
- Actions by \( t \)
- Other actions
Inductive Step: Case 2

$$\phi = \text{txbegin} \text{ by thread } t$$
Example: \( \Box \) is Necessary

```plaintext
while (g <> 1) ;
result := atomic{
    node.next =
    Top.read();
    node.val++ ;
    Top = node;
}
}

result := atomic{
    node.next =
    Top.read();
    node.val++ ;
    Top = node;
}
g := 1 ;
```
Completeness: $\equiv$ is Necessary

$$\text{TM}_C \equiv \text{TM}_A \iff \text{TM}_C \text{ observationally refines } \text{TM}_A$$

- For every history $H$, construct a program $P_H$ ensuring the opacity relation.

- I.e., the real-time order between transactions in every trace of $P_H$ must agree with the real-time order of the transactions in $H$.

💡 Use global variables & leaking of local variables.
Leaking Information from Aborted Transactions

Completeness result assumes we can read local state of aborted transactions

Be careful about rollback

ScalaSTM might need to try an atomic block more than once before optimistic concurrency can succeed. Any call into the STM might potentially discover the failure and trigger the rollback and retry. Local non-Ref variables that have a lifetime longer than the atomic block won’t be rolled back, and so they should be avoided. Local variables used only inside or only outside the atomic block are fine, though.

Below, badToString is incorrect because it uses a mutable StringBuilder both outside and inside its atomic block. The return value will definitely mention all of the elements.
Weaker Observations, Weaker Consistency Conditions

• When local variables are rolled back after a transaction aborts, TMS1 may suffice
  – I/O automata based definition
  – In TMS, the validity of each response is checked against a “coherent” subset of the transactions
  – May include commit-pending transactions

3.2. Why TMS1 enables transactional programming

The purpose of TMS1 is to specify what guarantees the TM runtime must make in order to ensure that programmers who think about their programs as if only serial executions (i.e., executions in which the events of each transaction appear consecutively) are possible do not receive any unpleasant surprises as a result of the concurrent execution of transactions. We explain below how TMS1’s validation conditions ensure that all responses given by the TM runtime are consistent with some serial execution of the program. In particular, for each response, we describe how to transform the actual program execution into a serial execution (i.e., one in which transactions are not interleaved with each other) such that the program cannot distinguish between the actual execution and the constructed serial execution.

First consider a commitOk or abort response that occurs when there are no other commit-pending transactions. The validation conditions of the program ensure that the transaction in question is consistent with some serial execution of the program.
What We Know about VWC

Sequence-based definition

- Each aborted transaction is checked for consistency (separately)

If atomic blocks return abort / commit (typically assumed) VWC does not preserve even weak observations.

VWC suffices if there are no return codes or just one thread and no global variables.

```c
tmp0 := commit;
tmp0 := atomic{
    read tx ;
    write ty ;
}
if (tmp0 == abort )
    gv = 1 ;
tmp1 = atomic{
    tz = 1 ;
}
tmp3 = gv ;
result = atomic{
    tmp4 = read(tz)
    if ((tmp3 == 1)
        or (tmp4 == 1))
        !!!!
}
```
Future Work

- infinite histories
- nesting
- access global variables inside atomic blocks (?)
- mixing transactional and non-transactional accesses

Possibly by considering other consistency conditions (TMS2, DU-Opacity)
Thank You