Proving Isolation Properties for Software Transactional Memory

Annette Bieniusa, Peter Thiemann

Department of Computer Science
University of Freiburg, Germany

19. Mai 2011

UNI FREIBURG

Semantics of transactions

Transactions follow the ACI(D) principle:

- atomicity ("all-or-nothing")
- isolation
- consistency

What does this actually mean?

It should prevent ... [Berenson 1995]

Dirty reads	Reading data that was never committed to shared memory.
Non-repeatable reads	Receiving different values for multiple reads.
Read Skew	Reading data from several memory snapshots.
Lost update	Intermediate updates by other transactions are overwritten.
Dirty writes	T_1 modifies a data item. T_2 then further modifies this data item before T_1 finishes. If T_1 or T_2 then perform a rollback, it is unclear what the restored data value should be.
Write Skew	T_1 and T_2 both read data items x and y. Then, T_1 updates x and commits; T_2 updates y, and commits, possibly violating constraints between x and y.

How to prove that a TM system is correct?

- **High-level models** don't show aborting transactions. ☹
- **Low-level models** are too specific wrt one implementation. ☹

How to prove that a TM system is correct?

- **High-level models** don't show aborting transactions. ☹
- **Low-level models** are too specific wrt one implementation. ☹

Our approach

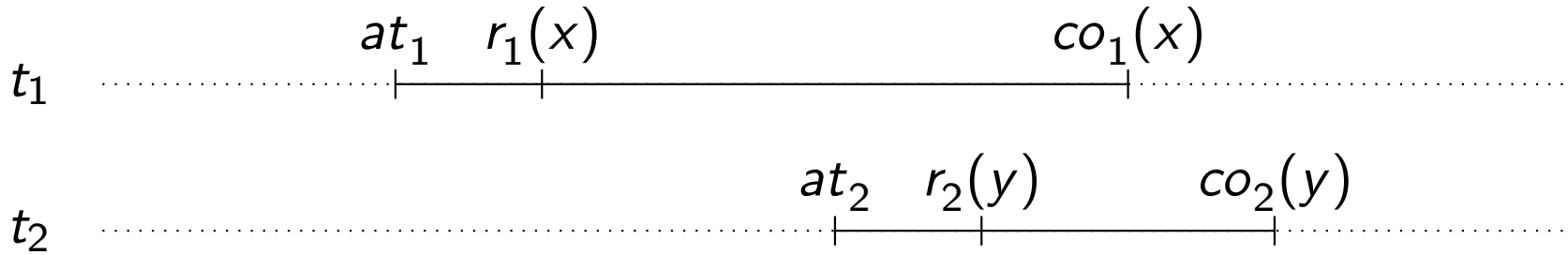
- **Small-step model** of transactions, but without synchronization details
- Operational semantics define the granularity of interleavings
- **Nondeterministic** choice of rules models the scheduling
- Evaluation steps are annotated with **effects** which abstract from the actual operation
- Every effect trace should be **serializable**

Effects

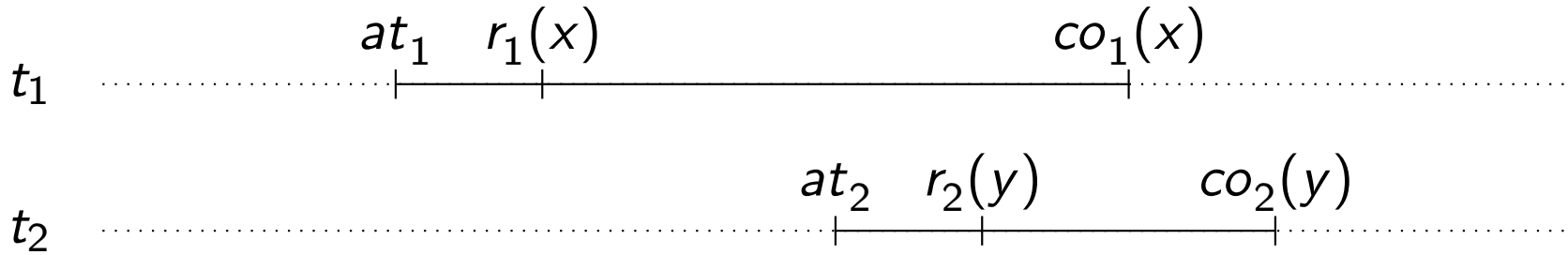
Effects by transaction i in thread t:

at_i^t	start txn
$r_i^t(l)$	read location l
$co_i^t(\bar{l})$	end txn and commit/write locations $\bar{l}$
ab_i^t	abort txn
ϵ_i^t	empty effect, administrative redex in txn
ϵ^t	empty effect
$sp^t(t')$	spawn new thread t'

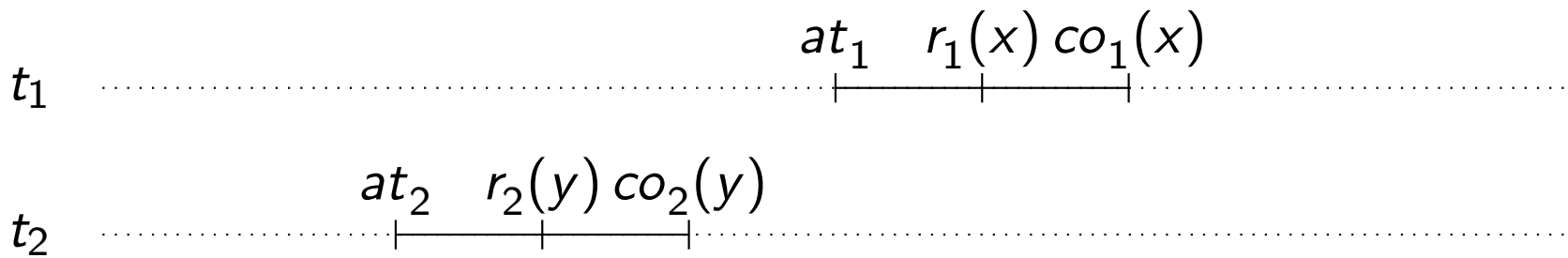
Successful commits



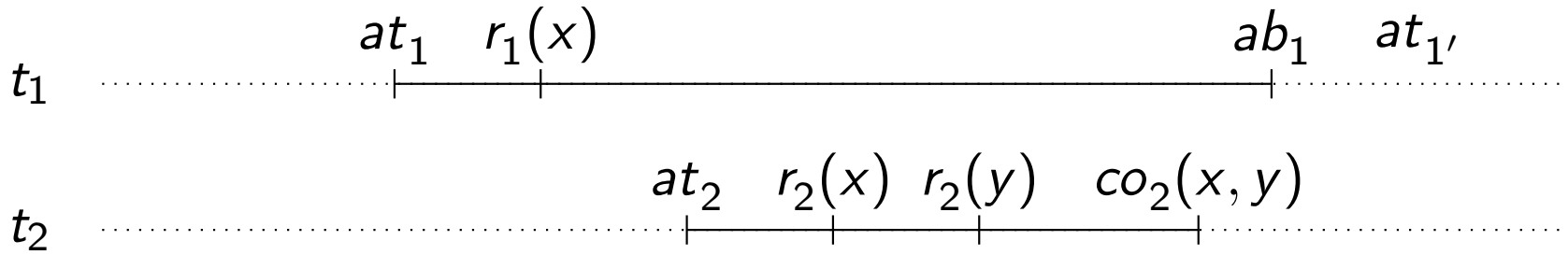
Successful commits



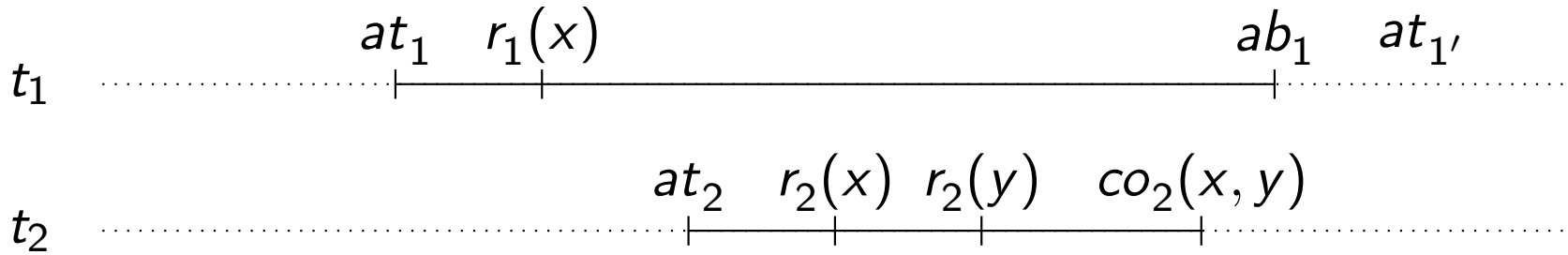
... is equivalent to:



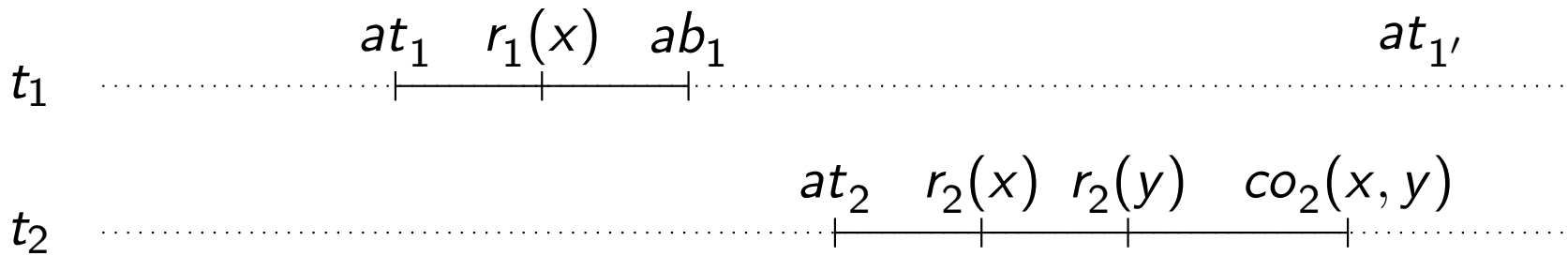
Read conflicts



Read conflicts



... is equivalent to:



Dependencies

Idea

We can re-order **independent** execution steps.

Dependencies

Idea

We can re-order **independent** execution steps.

Control Dependency

- $at_i^t \triangleright_c r_i^t(l)$
- $at_i^t \triangleright_c co_i^t(\bar{l})$
- ...

Dependencies

Idea

We can re-order **independent** execution steps.

Control Dependency

- $at_i^t \triangleright_c r_i^t(l)$
- $at_i^t \triangleright_c co_i^t(\bar{l})$
- ...

Data Dependency

- $r_i^t(l) \triangleright_d co_j^t(\bar{l})$ if $l \in \bar{l}$
- $co_i^t(\bar{l}) \triangleright_d r_j^t(l)$ if $l \in \bar{l}$
- $co_i^t(\bar{l}) \triangleright_d co_j^{t'}(\bar{l}')$ if $\bar{l} \cap \bar{l}' \neq \emptyset$

Formalization

- Monadic lambda calculus with txn operations
- Syntax of Λ_{STM}:

$$
\begin{array}{lcll}
x & \in & \mathrm{Var} & \quad l \in \mathrm{Ref} \\
v & \in & \mathrm{Val} \quad ::= & \quad l \mid \texttt{tt} \mid \texttt{ff} \mid () \mid \lambda x.e \\
e & \in & \mathrm{Exp} \quad ::= & \quad v \mid x \mid e\ e \mid \texttt{if}\ e\ e\ e \mid \texttt{return}\ e \mid e \gg\!= e \\
 & & \mid & \quad \texttt{spawn}\ e \mid \texttt{atomic}\ e \mid (e, W, R, i, e, \mathcal{H}) \\
 & & \mid & \quad \texttt{newref}\ e \mid \texttt{readref}\ e \mid \texttt{writeref}\ e\ e
\end{array}
$$

Execution traces

Execution steps:

$$\mathcal{H}, \mathcal{P} \stackrel{\alpha}{\Longrightarrow} \mathcal{H}', \mathcal{P}'$$

- Choose a thread from the thread pool nondeterministically.
- Execute one step.

$\Rightarrow$ The trace encodes the scheduler!

Operational Semantics: Transactions (selection)

IO-Atomic

$$\frac{i \text{ fresh}}{\mathcal{H}, \texttt{atomic}\ m \xrightarrow{at_i^t} \mathcal{H}, (m, \langle\rangle, \langle\rangle, i, m, \mathcal{H})}$$

Operational Semantics: Transactions (selection)

IO-Atomic

$$\frac{i \text{ fresh}}{\mathcal{H}, \texttt{atomic } m \xrightarrow{at_i^t} \mathcal{H}, (m, \langle\rangle, \langle\rangle, i, m, \mathcal{H})}$$

IO-Commit

$$\frac{check(R, \mathcal{H}) = ok \qquad \mathcal{H}'' = \mathcal{H}[W] \qquad \bar{l} = \mathrm{dom}(W)}{\mathcal{H}, (\texttt{return } v, W, R, i, m', \mathcal{H}') \xrightarrow{co_i^t(\bar{l})} \mathcal{H}'', \texttt{return } v}$$

Operational Semantics: Transactions (selection)

IO-Atomic

$$\frac{i \text{ fresh}}{\mathcal{H}, \texttt{atomic } m \xrightarrow{at_i^t} \mathcal{H}, (m, \langle\rangle, \langle\rangle, i, m, \mathcal{H})}$$

IO-Commit

$$\frac{check(R, \mathcal{H}) = ok \qquad \mathcal{H}'' = \mathcal{H}[W] \qquad \bar{l} = \mathrm{dom}(W)}{\mathcal{H}, (\texttt{return } v, W, R, i, m', \mathcal{H}') \xrightarrow{co_i^t(\bar{l})} \mathcal{H}'', \texttt{return } v}$$

IO-Rollback

$$\mathcal{H}, (m, W, R, i, m', \mathcal{H}') \xrightarrow{ab_i^t} \mathcal{H}, \texttt{atomic } m'$$

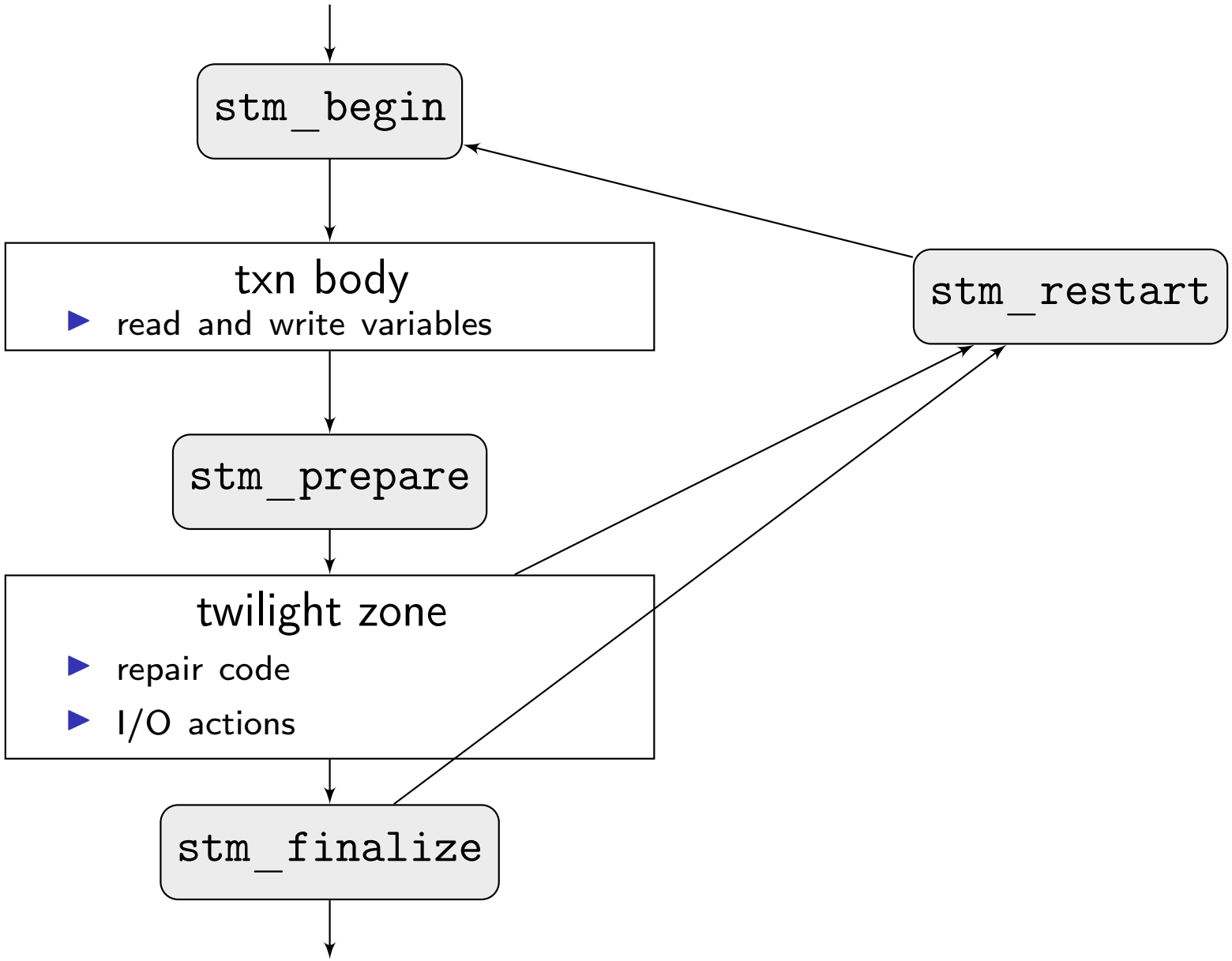
Opacity

- The type system enforces that traces all **well-formed**, but they are **not necessary serial**.
- Can we **re-order execution histories**, i.e. apply another scheduling, such that we get equivalent serial traces?
- We also require correctness for aborted transactions, therefore prove **opacity**.

Extensions

- Simple, yet powerful model
- Easily extensible for other constructs and semantics
  - Semantics of `orElse` and `retry`
  - Snapshot isolation
  - Irrevocable transactions

Twilight STM: Conflict repair and I/O integration



TwilightSTM

- extending the STM interface with repair actions, conflict detection, I/O
- application-specific tuning
- implemented and evaluated in C, Java, Haskell

TwilightSTM

- extending the STM interface with repair actions, conflict detection, I/O
- application-specific tuning
- implemented and evaluated in C, Java, Haskell

DecentSTM

- distributed STM without central components, no timestamps
- multi-versioning with GC on versions
- programmer only specifies atomic blocks, shared data, placement of nodes
- JCell: integration in distributed JVM

Summary

- A **formal semantics** of TM that is suitable for proving properties of a TM implementation
  - not high-level because interleavings of transactions are modelled explicitly
  - not low-level because actual implementation of synchronization is left abstract
- A proof that the semantics implements **opacity, SI, irrevocability**, ...
- How can we combine txns with different isolation semantics?

Summary

- A **formal semantics** of TM that is suitable for proving properties of a TM implementation
  - not high-level because interleavings of transactions are modelled explicitly
  - not low-level because actual implementation of synchronization is left abstract
- A proof that the semantics implements **opacity, SI, irrevocability**, ...
- How can we combine txns with different isolation semantics?

Questions?