Correctness of an STM Haskell Implementation

Manfred Schmidt-Schauß, David Sabel

Goethe-University, Frankfurt, Germany

ICFP '13, Boston, USA

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Software Transactional Memory (STM)

- treats **shared memory** operations as **transactions**
- provides **lock-free** and **very convenient** concurrent programming
- requires an **implementation** that **correctly executes** the transactions

STM Haskell

- STM library for Haskell
- introduced by Harris et.al, PPoPP'05
- uses Haskell's **strong type system** to distinguish between
  - software transactions,
  - functional code, and
  - IO-computations

Transactional Variables:

```
TVar a
```

Primitives to form STM-transactions `STM a`:

```
newTVar     :: a -> STM (TVar a)
readTVar    :: TVar a -> STM a
writeTVar   :: TVar a -> a -> STM ()

return      :: a -> STM a
(>>=)       :: STM a -> (a -> STM b) -> STM b

retry       :: STM ()
orElse      :: STM a -> STM a -> STM a
```

Executing an STM-transaction:

```
atomically :: STM a -> IO a
```

Semantics: the transaction-execution is

- **atomic**: all or nothing, effects are indivisible, and
- **isolated**: concurrent evaluation is not observable

Correctness of an STM-Implementation

Issues:

- Is an STM implementation **correct**?
- What does **correctness** mean?

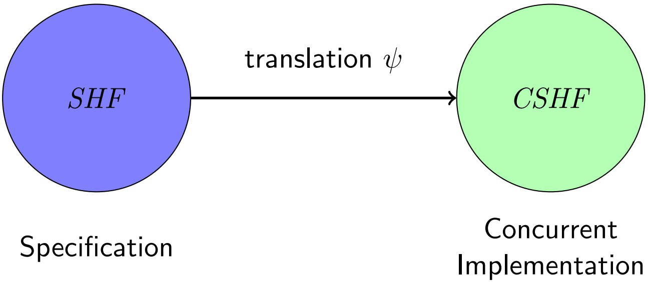
Several correctness notions have been suggested

e.g. Guerraoui & Kapalka, PPoPP'08

- linearizability, serializability, recoverability, opacity, ...
- Most of these notions are **properties on the trace** of read-/write accesses on the transactional variables.

Our approach is **different**: "**semantic** approach"

Two program calculi for STM Haskell:



Correctness: The implementation fulfills the specification

→ ψ is semantics reflecting

Adapted from the CHF-calculus (S.& Schmidt-Schauß: PPDP'11, LICS'12)

Processes:

TVar x with content e

$$P_i \in \mathit{Proc} ::= P_1 \mid P_2 \mid \nu x.P \mid \underbrace{\langle u \wr x\rangle \Leftarrow e} \mid x = e \mid \overbrace{x\,\mathbf{t}\,e}$$

Concurrent future x with identifier u evaluates e

Expressions:

$$\left.\begin{array}{l} e_i \in \mathit{Exp} ::= x \mid \lambda x.e \mid (e_1\ e_2) \mid (c\ e_1 \ldots e_{\mathrm{ar}(c)}) \\ \quad \mid \texttt{seq}\ e_1\ e_2 \mid \texttt{letrec}\ x_1 = e_1, \ldots, x_n = e_n\ \texttt{in}\ e \\ \quad \mid \texttt{case}_T\ e\ \texttt{of}\ alt_{T,1} \ldots\ alt_{T,|T|} \\ \qquad \text{where } alt_{T,i} = (c_{T,i}\ x_1 \ldots x_{\mathrm{ar}(c_{T,i})} \to e_i) \end{array}\right\} \text{extended } \lambda\text{-calculus}$$

$$\left.\begin{array}{l} \quad \mid \texttt{return}_{\texttt{IO}}\, e \mid e_1 \gg\!\!=_{\texttt{IO}} e_2 \mid \texttt{future}\, e \\ \quad \mid \texttt{atomically}\, e \mid \texttt{return}_{\texttt{STM}}\, e \mid e_1 \gg\!\!=_{\texttt{STM}} e_2 \\ \quad \mid \texttt{retry} \mid \texttt{orElse}\, e_1\, e_2 \\ \quad \mid \texttt{newTVar}\, e \mid \texttt{readTVar}\, e \mid \texttt{writeTVar}\, e \end{array}\right\} \text{IO and STM}$$

Monomorphic type system

Operational Semantics:

- **Call-by-need** "small-step" reduction $\xrightarrow{SHF}$, several rules, e.g.

(fork) $\langle u \wr y \rangle \Leftarrow \mathbb{M}[\mathtt{future}\ e] \xrightarrow{SHF} \nu z, u'.(\langle u \wr y \rangle \Leftarrow \mathbb{M}[\mathtt{return}_{\mathtt{IO}}\ z] \mid \langle u' \wr z \rangle \Leftarrow e)$

- **Big-step rule** for transactional evaluation:

$$\frac{\mathbb{D}_1[\langle u \wr y \rangle \Leftarrow \mathbb{M}[\mathtt{atomically}\ e]] \xrightarrow{SHFA,*} \mathbb{D}_1'[\langle u \wr y \rangle \Leftarrow \mathbb{M}[\mathtt{atomically}\ (\mathtt{return}_{\mathtt{STM}}\ e')]]}{\mathbb{D}[\langle u \wr y \rangle \Leftarrow \mathbb{M}[\mathtt{atomically}\ e]] \xrightarrow{SHF} \mathbb{D}'[\langle u \wr y \rangle \Leftarrow \mathbb{M}[\mathtt{return}_{\mathtt{IO}}\ e']]}$$

where $\xrightarrow{SHFA}$ are small-step rules for transactional evaluation

- Enforces **sequential evaluation** of transactions
  - ➽ atomicity and isolation obviously hold
- Rule application is **undecidable**!

Extensions w.r.t. SHF:

- **local** and **global** TVars:
  - u **tls** S = Stack of **thread-local** TVars
  - x **tg** $e\,u\,g$ = **global** TVar, where
    - u is a locking label (unlocked / locked by thread u)
    - g is a list of thread identifiers (the **notify list**)
- threads may have a **transaction log**: $\langle u \wr y \rangle \overset{T,L;K}{\Longleftarrow} e$
  T, L, K are (stacked) lists storing information about created, read, and written TVars
- ...

Stacks are necessary for rollback during **nested** `orElse`-evaluation

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Operational semantics:

- **true small-step** reduction $\xrightarrow{CSHF}$
- **concurrent** evaluation of STM transactions
- all rule applications are **decidable**

Transaction execution (informally):

- all read/writes are **logged** and performed on **local** TVars
- during the first `readTVar`-operation of thread u on TVar x: u is **added** to the **notify list** of TVar x
- commit phase
  1. **lock** global TVars
  2. **send a `retry`** to all threads in the **notify lists** of to-be-written TVars (**= conflicting threads**)
  3. write content of local TVars into global TVars
  4. remove the locks

For $calc \in \{SHF, CSHF\}$

Contextual Equivalence $\sim_{calc}$

$P_1 \sim_{calc} P_2$ iff for all contexts $\mathbb{D}$:

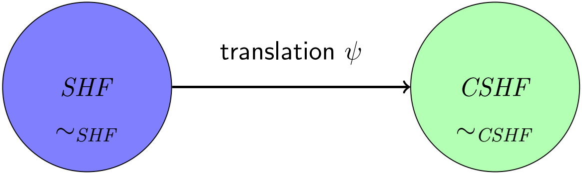
$$\mathbb{D}[P_1]{\downarrow_{calc}} \iff \mathbb{D}[P_2]{\downarrow_{calc}} \quad \land \quad \mathbb{D}[P_1]{\Downarrow_{calc}} \iff \mathbb{D}[P_2]{\Downarrow_{calc}}$$

where

- Process P is **successful** iff $P \equiv \mathbb{D}[\langle x \wr u \rangle \stackrel{\text{main}}{\Longleftarrow} \texttt{return}\ e]$
- **May-Convergence**:

  $P{\downarrow_{calc}}$ iff $\exists P' : P \xrightarrow{calc,*} P' \land P'$ is successful
- **Should-Convergence**:

  $P{\Downarrow_{calc}}$ iff $\forall P' : P \xrightarrow{calc,*} P' \implies P'{\downarrow_{calc}}$



Main Theorem

Convergence Equivalence: For any SHF-process P:

$$P{\downarrow_{SHF}} \iff \psi(P){\downarrow_{CSHF}} \text{ and } P{\Downarrow_{SHF}} \iff \psi(P){\Downarrow_{CSHF}}$$

Adequacy: For all $P_1, P_2 \in SHF$:

$$\psi(P_1) \sim_{CSHF} \psi(P_2) \implies P_1 \sim_{SHF} P_2$$

- $CSHF$ is a **correct evaluator** for SHF
- Correct **program transformations** in $CSHF$ are also correct for SHF

Conclusion and Further Work

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Conclusion

- **Semantic correctness** of an STM-Haskell implementation
- using **contextual equivalence** with may- and should-convergence

Further work

- Transfer the result to **GHC's STM implementation**
- Develop smarter strategies for the transaction manager and prove their correctness
- Language extensions: **polymorphic** types, **exceptions**,. . .

Backup Slides

Comparison GHC STM and *CSHF*

Conflict detection:

GHC STM: thread compares transaction log with content of TVars
restarts itself if a conflict occurred
(temporarily and before commit)

CSHF: the committing thread restarts conflicting threads

Pointer equality test:

GHC STM: required

CSHF : not required

Conflict requires:

GHC STM: different content

CSHF : changed content (not necessarily different)

Sketch of the Proof

- $P{\downarrow_{SHF}} \implies \psi(P){\downarrow_{CSHF}}$:
  map reductions $P \xrightarrow{SHF,*} P'$ to reductions $\psi(P) \xrightarrow{CSHF,*} \psi(P')$

- $\psi(P){\downarrow_{CSHF}} \implies P{\downarrow_{SHF}}$:
  - reorder the sequence $\psi(P) \xrightarrow{CSHF,*} P'$, s.t. reductions are **grouped per transaction**
  - **remove non-committed** transactions
  - now the sequence can be mapped to a sequence $P \xrightarrow{SHF,*} P''$

- $P{\Downarrow_{SHF}} \iff \psi(P){\Downarrow_{CSHF}}$:
  - similar, by showing equivalence of **may-divergence**:
    $P\uparrow_{SHF} \iff \psi(P)\uparrow_{CSHF}$
  - $P\uparrow = \neg(P\Downarrow) = \exists Q : P \xrightarrow{*} Q \wedge \neg(Q\downarrow)$