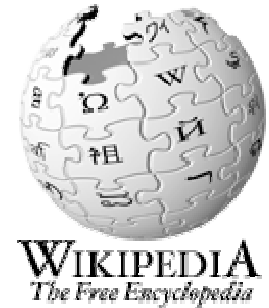


A Concurrent Calculus with Atomic Transactions

Avec Lucia Acciai (LIF) et
Michele Boreale (Univ. Firenze)

Software Transactional Memory

- **software transactional memory** (STM) is a [concurrency control](#) mechanism analogous to [database transactions](#) for controlling access to [shared memory](#) in [concurrent computing](#).
- It functions as an alternative to [lock-based synchronization](#), and is typically implemented in a [lock-free](#) way.
- A transaction in this context is a piece of code that executes a series of reads and writes to shared memory. These reads and writes logically occur at a single instant in time; intermediate states are not visible to other (successful) transactions.



Software Transactional Memory

- The idea of providing hardware support for transactions originated in 1993 by Herlihy and Moss.
- This is a model for multithreaded programs running on multiprocessors.
- In 1995 Nir Shavit and Dan Touitou extended this idea to software-only transactional memory (STM). STM has recently been the focus of intense research and support for practical implementations is growing.

Software Transactional Memory

- The conceptual simplicity of STMs enable them to be exposed to the programmer using relatively simple language syntax.
 - Tim Harris and Keir Fraser's "Language Support for Lightweight Transactions" proposed the idea of using the classical conditional critical region (CCR) to represent transactions.
- In 2005, Tim Harris, Simon Marlow, Simon Peyton Jones, and Maurice Herlihy described an STM system built on Concurrent Haskell
 - It enables arbitrary atomic operations to be composed into larger atomic operations, a useful concept impossible with lock-based programming.

A-CCS Processes

$P, Q ::= \mathbf{0}$	nil
$ \bar{a}$	(asynchronous) output action
$ a.P$	input
$ *a.P$	replicated input
$ \text{atomic}(M)$	atomic block
$ P P$	parallel composition
$ P \setminus^n a$	hiding
$ \{A\}_M$	ongoing atomic block (partial execution of M)

Atomic Expressions

Actions $\alpha, \beta ::= \text{rd}(a)$ read a from memory
| $\text{wt}(a)$ write a into the memory

Expressions $M, N ::= \text{end}$ termination
| retry abort and retry the current atomic block
| $\alpha.M$ action prefix
| $M \text{ orElse } M$ alternative

Ongoing Atomic Expressions

$A, B ::= (M)_{\sigma; \delta}$ execution of M with state σ and log δ
| A orElse A ongoing alternative

- A **state** σ is a multiset of names (output actions)
- A **log** δ is a sequence of (read / write) actions

Actions	$\alpha, \beta ::= \text{rd}(a)$	read a from memory
	$ \text{wt}(a)$	write a into the memory
(Atomic) Expressions	$M, N ::= \text{end}$	termination
	$ \text{retry}$	abort and retry the current atomic block
	$ \alpha.M$	action prefix
	$ M \text{ orElse } M$	alternative
Ongoing atomic block	$A, B ::= (M)_{\sigma, \delta}$	execution of M with state σ and log δ
	$ A \text{ orElse } A$	ongoing alternative
Processes	$P, Q ::= \mathbf{0}$	nil
	$ \bar{a}$	(asynchronous) output action
	$ a.P$	input
	$ *a.P$	replicated input
	$ \text{atomic}(M)$	atomic block
	$ P P$	parallel composition
	$ P \setminus^n a$	hiding
	$ \{A\}_M$	ongoing atomic block (partial execution of M)

Example

$p_1!(0) \parallel p_2!(0)$

$\parallel p_1?(x) \cdot p_2?(y) \cdot \text{if } x \neq y \text{ then boom!}$
 else OK!

$\parallel p_1?(x) \cdot (p_1!(x+1) \parallel p_2?(y) \cdot p_2!(x+1))$

Operational Semantics

$$\text{(OUT)} \quad \bar{a}; \sigma \rightarrow \mathbf{0}; \sigma \uplus a$$

$$\text{(IN)} \quad a.P; \sigma \uplus a \rightarrow P; \sigma$$

$$\text{(PAR-L)} \quad \frac{P; \sigma \rightarrow P'; \sigma'}{P|Q; \sigma \rightarrow P'|Q; \sigma'}$$

Operational Semantics (cont.)

$$\text{(COM)} \quad \frac{P; \sigma \rightarrow P'; \sigma \uplus \{a\} \quad Q; \sigma \uplus \{a\} \rightarrow Q'; \sigma}{P | Q \rightarrow P' | Q'}$$

$$\text{(AT-ST)} \quad \text{atomic}(M); \sigma \rightarrow \{(M)_{\sigma; \varepsilon}\} M; \sigma$$

$$\text{(AT-PASS)} \quad \frac{A \rightarrow A'}{\{A\} M; \sigma \rightarrow \{A'\} M; \sigma}$$

$$\text{(AT-RE)} \quad \{(\text{retry})_{\sigma; \delta}\} M; \sigma \rightarrow \text{atomic}(M); \sigma$$

Operational Semantics (end)

$$\text{(AT-FAIL)} \quad \frac{\delta^r \not\subseteq \sigma}{\{(\text{end})_{\sigma;\delta}\}M;\sigma \rightarrow \text{atomic}(M);\sigma}$$

$$\text{(AT-OK)} \quad \frac{\delta^r \subseteq \sigma \quad \delta^w = \{a_1, \dots, a_n\}}{\{(\text{end})_{\sigma;\delta}\}M;\sigma \rightarrow \overline{a_1} \mid \dots \mid \overline{a_n}; (\sigma - \delta^r)}$$

- The notation δ^r is for the read actions of δ
- The notation δ^w is for the write actions

Reduction on Atomic Blocks

$$(A\text{-RD-OK}) \frac{(\delta.\text{rd}(a))^r \subseteq \sigma}{(\text{rd}(a).M)_{\sigma;\delta} \rightarrow (M)_{\sigma;\delta.\text{rd}(a)}}$$

$$(A\text{-WR}) \quad (\text{wt}(a).M)_{\sigma;\delta} \rightarrow (M)_{\sigma;\delta.\text{wt}(a)}$$

$$(A\text{-OE-I}) \quad (M_1 \text{ orElse } M_2)_{\sigma;\delta} \rightarrow (M_1)_{\sigma;\delta} \text{ orElse } (M_2)_{\sigma;\delta}$$

$$(A\text{-OE-F}) \quad (\text{retry})_{\sigma;\delta} \text{ orElse } B \rightarrow B$$

$$(A\text{-OE-E}) \quad (\text{end})_{\sigma;\delta} \text{ orElse } B \rightarrow (\text{end})_{\sigma;\delta}$$

Example: hiding operator

$$(\text{atomic}(\text{rd}(a).\text{wt}(b).\text{end}) \mid \bar{a}) \setminus^0 a \rightarrow \text{atomic}(\text{rd}(a).\text{wt}(b).\text{end}) \setminus^1 a$$

Example: Encoding Choice

$$a_1 .P_1 + \cdots + a_n .P_n$$

An (input guarded) choice operation can be encoded by the At-CCS process

$$\begin{aligned} &(\text{atomic}(\text{rd}(a_1).\text{wt}(k_1).\text{end}) \text{ orElse } \cdots \text{ orElse } \text{rd}(a_n).\text{wt}(k_n).\text{end}) \\ &| k_1 .P_1 | \cdots | k_n .P_n) \setminus^0 \{k_1 \cdots k_n\} \end{aligned}$$

Example: Encoding Join

$$a_1 \times \cdots \times a_n . P$$

A multi-synchronization (a la join-calculus) can be encoded by the At-CCS process

$$(\text{atomic}(\text{rd}(a_1) \cdots \text{rd}(a_n) \text{.wt}(k) \text{.end}) \mid k . P) \setminus^0 \{k\}$$

Laws of Transactions

Laws for atomic expressions:

$$\text{(COMM)} \quad \alpha.\alpha'.M \approx \alpha'.\alpha.M$$

$$\text{(DIST)} \quad \alpha.(M \text{ orElse } N) \approx (\alpha.M) \text{ orElse } (\alpha.N)$$

$$\text{(ASS)} \quad M_1 \text{ orElse } (M_2 \text{ orElse } M_3) \approx (M_1 \text{ orElse } M_2) \text{ orElse } M_3$$

$$\text{(ABS)} \quad \alpha.\text{retry} \approx \text{retry}$$

Laws for processes:

$$\text{(ASY)} \quad a.\bar{a} \approx \mathbf{0}$$

$$\text{(A-ASY)} \quad \text{atomic}(\text{rd}(a).\text{wt}(a).\text{end}) \approx \mathbf{0}$$

$$\text{(A-1)} \quad \text{atomic}(\text{rd}(a).\text{end}) \approx a.\mathbf{0}$$