Towards a formal account for software transactional memory

Doriana Medić Claudio Antares Mezzina Iain Phillips Nobuko Yoshida

Reversible Computation July 9 - July 10, 2020, Oslo, Norway

online event

Introduction

- Reversible computing has been studied in several contexts ranging from quantum computing, biochemical modelling, programming, and program debugging.

- Of particular interest is its application to the study of programming abstractions for reliable systems.

- Distributed reversible actions can be seen as a building blocks for different transactional models and recovery techniques.

◦ An example showing how notions of reversible and irreversible actions in a process calculus can model a primitive form of transaction is given in [1].

◦ On the shared memory side, we recall [2], where a CCS endowed with a mechanism for software transactional memories (STMs) is presented and [3] which studies reversibility and a high-level abstraction of shared memory (tuple spaces).

[1] *V. Danos and J. Krivine. Transactions in RCCS.*

[2] *L. Acciai, M. Boreale and S. Dal-Zilio. A Concurrent Calculus with Atomic Transactions.*

[3] *E. Giachino, I. Lanese, C. A. Mezzina and F. Tiezzi: Causal-consistent rollback in a tuple-based language.*

Software Transactional Memory

- Software Transactional Memory is a way to address the problem of concurrent programming, by relieving the programmer from the burden of dealing with locks.

- Opposite to the lock-based approach, STM uses transactions, blocks of code accessing shared data which are meant to be executed **atomically**.

- It is necessary to specify the sequences of operations to be enclosed in transactions, while the system is in charge of the interleaving between the concurrent transactions.

- A transaction can either **commit** and update the system permanently or **abort** and discard all the changes done by its execution.

In this work:

- our interest is in the interplay between reversible computing and the STM approach to control the concurrent executions;

- we present a formal framework for describing STMs in a simple shared memory context;

- a transaction can access a shared variable either in **read** or in **write** mode;

- we show how it is possible to model **writer** and **reader preference** in our framework;

Example:

◦ Consider the following C-like code where two functions/threads access the same shared variables:

```
int x = 0;
int y = 5;
int z = 0;

void t1()          void t2()
 {z = y+x;  }       { x = z+1; }
```

- The possible executions of the two functions are:

t1 ; t2	t2 ; t1	t1 | t2
z = 5	z = 6	z = 5
x = 6	x = 1	x = 1

- Either the two functions are executed sequentially or are interleaved (leading to an unwanted state).

Syntax:

$$
\begin{array}{rlll}
\text{(Actions)} & \alpha, \beta & ::= & \mathtt{wr}(x) \mid \mathtt{rd}(x) \\
\text{(Processes)} & A, B & ::= & \mathbf{0} \mid \sum_i \alpha_i.A_i \\
\text{(Expressions)} & X, Y & ::= & B \mid \alpha.X \mid X; Y \mid (X \mid Y) \mid t : [\![A]\!]_\Gamma \\
\text{(Configuration)} & C & ::= & X \parallel M \\
\text{(Shared Memory)} & M & ::= & \langle x, W, R \rangle \parallel M
\end{array}
$$

Syntax:

$$
\begin{array}{rlll}
\text{(Actions)} & \alpha, \beta & ::= & \mathtt{wr}(x) \mid \mathtt{rd}(x) \\
\text{(Processes)} & A, B & ::= & \mathbf{0} \mid \sum_i \alpha_i.A_i \\
\text{(Expressions)} & X, Y & ::= & B \mid \alpha.X \mid X; Y \mid (X \mid Y) \mid t : [\![A]\!]_\Gamma \\
\text{(Configuration)} & C & ::= & X \parallel M \\
\text{(Shared Memory)} & M & ::= & \langle x, W, R \rangle \parallel M
\end{array}
$$

Syntax:

$$
\begin{array}{rlll}
\text{(Actions)} & \alpha, \beta & ::= & \mathtt{wr}(x) \mid \mathtt{rd}(x) \\
\text{(Processes)} & A, B & ::= & \mathbf{0} \mid \sum_i \alpha_i.A_i \\
\text{(Expressions)} & X, Y & ::= & B \mid \alpha.X \mid X; Y \mid (X \mid Y) \mid t : [\![A]\!]_\Gamma \\
\text{(Configuration)} & C & ::= & X \parallel M \\
\text{(Shared Memory)} & M & ::= & \langle x, W, R \rangle \parallel M
\end{array}
$$

Syntax:

$$
\begin{array}{rlll}
\text{(Actions)} & \alpha, \beta & ::= & \mathtt{wr}(x) \mid \mathtt{rd}(x) \\
\text{(Processes)} & A, B & ::= & \mathbf{0} \mid \sum_i \alpha_i.A_i \\
\text{(Expressions)} & X, Y & ::= & B \mid \alpha.X \mid X; Y \mid (X \mid Y) \mid t : [\![A]\!]_\Gamma \\
\text{(Configuration)} & C & ::= & X \parallel M \\
\text{(Shared Memory)} & M & ::= & \langle x, W, R \rangle \parallel M
\end{array}
$$

Syntax:

$$
\begin{array}{rlll}
\text{(Actions)} & \alpha, \beta & ::= & \mathtt{wr}(x) \mid \mathtt{rd}(x) \\
\text{(Processes)} & A, B & ::= & \mathbf{0} \mid \sum_i \alpha_i.A_i \\
\text{(Expressions)} & X, Y & ::= & B \mid \alpha.X \mid X; Y \mid (X \mid Y) \mid t : [\![A]\!]_\Gamma \\
\text{(Configuration)} & C & ::= & X \parallel M \\
\text{(Shared Memory)} & M & ::= & \langle x, W, R \rangle \parallel M
\end{array}
$$

History context

A history context $\mathtt{H}$ is a process with a hole $\bullet$, defined by the following grammar: $\mathtt{H} ::= \bullet \mid \alpha.\bullet + A$.

Example: $t : [\![\mathtt{wr}(x).\mathtt{rd}(x_1).\mathtt{rd}(y).A + B]\!]_\Gamma$

can be written as:

$$t : [\![\mathtt{H}[\mathtt{rd}(y).A]]\!]_\Gamma$$

where $\mathtt{H} = \mathtt{wr}(x).\mathtt{rd}(x_1).\bullet + B$.

History context

A history context $\mathtt{H}$ is a process with a hole $\bullet$, defined by the following grammar: $\mathtt{H} ::= \bullet \mid \alpha.\bullet + A$.

Example: $t : [\![\mathtt{wr}(x).\mathtt{rd}(x_1).\mathtt{rd}(y).A + B]\!]_\Gamma$

can be written as:

$$t : [\![\mathtt{H}[\mathtt{rd}(y).A]]\!]_\Gamma$$

where $\mathtt{H} = \mathtt{wr}(x).\mathtt{rd}(x_1).\bullet + B$.

◦ To be able to identify the state of the internal computation of a transaction, we mark it with symbol $^\wedge$.

Example: in transition

$$t : [\![\mathtt{rd}(x).\mathtt{rd}(y).{}^\wedge\mathtt{wr}(z).\mathtt{wr}(x')]\!]_\Gamma$$

actions $\mathtt{rd}(x)$ and $\mathtt{rd}(y)$ are already executed.

Write rule

$$(\textsc{Wr})\ \frac{(W \subseteq \{t\} \wedge R \subseteq \{t\})}{t : [\![\mathtt{H}[\hat{}\,\mathtt{wr}(x).A + B]]\!]_\Gamma \parallel \langle x, W, R\rangle \parallel M \to t : [\![\mathtt{H}[\mathtt{wr}(x).\hat{}\,A + B]]\!]_\Gamma \parallel \langle x, W \cup t, R\rangle \parallel M}$$

Write rule

$$(\textsc{Wr})\ \frac{(W \subseteq \{t\} \wedge R \subseteq \{t\})}{t : [\![\mathtt{H}[\hat{}\,\mathtt{wr}(x).A + B]]\!]_\Gamma \parallel \langle x, W, R\rangle \parallel M \to t : [\![\mathtt{H}[\mathtt{wr}(x).\hat{}\,A + B]]\!]_\Gamma \parallel \langle x, W \cup t, R\rangle \parallel M}$$

Write rule

$$(\textsc{Wr})\ \frac{(W \subseteq \{t\} \wedge R \subseteq \{t\})}{t : [\![\mathtt{H}[\hat{}\,\mathtt{wr}(x).A + B]]\!]_\Gamma \parallel \langle x, W, R\rangle \parallel M \to t : [\![\mathtt{H}[\mathtt{wr}(x).\hat{}\,A + B]]\!]_\Gamma \parallel \langle x, W \cup t, R\rangle \parallel M}$$

Write rule

$$(\textsc{Wr})\ \frac{(W \subseteq \{t\} \wedge R \subseteq \{t\})}{t : [\![\mathtt{H}[\hat{}\,\mathtt{wr}(x).A + B]]\!]_\Gamma \parallel \langle x, W, R\rangle \parallel M \to t : [\![\mathtt{H}[\mathtt{wr}(x).\hat{}\,A + B]]\!]_\Gamma \parallel \langle x, W \cup t, R\rangle \parallel M}$$

Write rule

$$(\textsc{Wr})\ \frac{(W \subseteq \{t\} \wedge R \subseteq \{t\})}{t : [\![\mathtt{H}[\hat{}\,\mathtt{wr}(x).A + B]]\!]_\Gamma \parallel \langle x, W, R\rangle \parallel M \to t : [\![\mathtt{H}[\mathtt{wr}(x).\hat{}\,A + B]]\!]_\Gamma \parallel \langle x, W \cup t, R\rangle \parallel M}$$

Example:

$$t : [\![\hat{}\,\mathtt{wr}(x).\mathtt{rd}(y)]\!]_\emptyset \parallel \langle x, \emptyset, \emptyset\rangle \parallel \langle y, \emptyset, \emptyset\rangle$$

Write rule

$$(\textsc{Wr})\ \frac{(W \subseteq \{t\} \wedge R \subseteq \{t\})}{t : [\![\mathtt{H}[\hat{}\,\mathtt{wr}(x).A + B]]\!]_\Gamma \parallel \langle x, W, R\rangle \parallel M \to t : [\![\mathtt{H}[\mathtt{wr}(x).\hat{}\,A + B]]\!]_\Gamma \parallel \langle x, W \cup t, R\rangle \parallel M}$$

Example:

$$t : [\![\hat{}\,\mathtt{wr}(x).\mathtt{rd}(y)]\!]_\emptyset \parallel \langle x, \emptyset, \emptyset\rangle \parallel \langle y, \emptyset, \emptyset\rangle$$

$$\to$$

Write rule

$$(\textsc{Wr})\ \frac{(W \subseteq \{t\} \wedge R \subseteq \{t\})}{t : [\![\mathtt{H}[\hat{}\,\mathtt{wr}(x).A + B]]\!]_\Gamma \parallel \langle x, W, R\rangle \parallel M \to t : [\![\mathtt{H}[\mathtt{wr}(x).\hat{}\,A + B]]\!]_\Gamma \parallel \langle x, W \cup t, R\rangle \parallel M}$$

Example:

$$t : [\![\hat{}\,\mathtt{wr}(x).\mathtt{rd}(y)]\!]_\emptyset \parallel \langle x, \emptyset, \emptyset\rangle \parallel \langle y, \emptyset, \emptyset\rangle$$

$$\to$$

$$t : [\![\mathtt{wr}(x).\hat{}\,\mathtt{rd}(y)]\!]_\emptyset \parallel$$

Write rule

$$(\textsc{Wr})\ \frac{(W \subseteq \{t\} \wedge R \subseteq \{t\})}{t : [\![\mathtt{H}[\hat{}\,\mathtt{wr}(x).A + B]]\!]_\Gamma \parallel \langle x, W, R\rangle \parallel M \to t : [\![\mathtt{H}[\mathtt{wr}(x).\hat{}\,A + B]]\!]_\Gamma \parallel \langle x, W \cup t, R\rangle \parallel M}$$

Example:

$$t : [\![\hat{}\,\mathtt{wr}(x).\mathtt{rd}(y)]\!]_\emptyset \parallel \langle x, \emptyset, \emptyset\rangle \parallel \langle y, \emptyset, \emptyset\rangle$$

$$\to$$

$$t : [\![\mathtt{wr}(x).\hat{}\,\mathtt{rd}(y)]\!]_\emptyset \parallel \langle x, \{t\}, \emptyset\rangle \parallel \langle y, \emptyset, \emptyset\rangle$$

Read rule

$$(\textsc{Rd})\ t : [\![\mathtt{H}[\hat{}\,\mathtt{rd}(x).A + B]]\!]_\Gamma \parallel \langle x, W, R\rangle \parallel M \to t : [\![\mathtt{H}[\mathtt{rd}(x).\hat{}\,A + B]]\!]_{\Gamma \cup (W \setminus t)} \parallel \langle x, W, R \cup t\rangle \parallel M$$

Read rule

$$(\textsc{Rd})\ t : [\![\mathtt{H}[\hat{}\,\mathtt{rd}(x).A + B]]\!]_\Gamma \parallel \langle x, W, R\rangle \parallel M \to t : [\![\mathtt{H}[\mathtt{rd}(x).\hat{}\,A + B]]\!]_{\Gamma \cup (W \setminus t)} \parallel \langle x, W, R \cup t\rangle \parallel M$$

Read rule

$$(\textsc{Rd})\ t : [\![\mathtt{H}[\hat{}\,\mathtt{rd}(x).A + B]]\!]_\Gamma \parallel \langle x, W, R\rangle \parallel M \to t : [\![\mathtt{H}[\mathtt{rd}(x).\hat{}\,A + B]]\!]_{\Gamma \cup (W \setminus t)} \parallel \langle x, W, R \cup t\rangle \parallel M$$

Read rule

$$(\textsc{Rd})\ t : [\![\mathtt{H}[\hat{}\,\mathtt{rd}(x).A + B]]\!]_\Gamma \parallel \langle x, W, R\rangle \parallel M \to t : [\![\mathtt{H}[\mathtt{rd}(x).\hat{}\,A + B]]\!]_{\Gamma \cup (W \setminus t)} \parallel \langle x, W, R \cup t\rangle \parallel M$$

Read rule

$$(\textsc{Rd})\ t : [\![\mathtt{H}[\hat{}\,\mathtt{rd}(x).A + B]]\!]_\Gamma \parallel \langle x, W, R\rangle \parallel M \to t : [\![\mathtt{H}[\mathtt{rd}(x).\hat{}\,A + B]]\!]_{\Gamma \cup (W \setminus t)} \parallel \langle x, W, R \cup t\rangle \parallel M$$

Example:

$$t : [\![\mathtt{wr}(x).\hat{}\,\mathtt{rd}(y)]\!]_\emptyset \parallel \langle x, \{t\}, \emptyset\rangle \parallel \langle y, \emptyset, \emptyset\rangle$$

Read rule

$$(\textsc{Rd})\ t : [\![\mathtt{H}[\hat{}\,\mathtt{rd}(x).A + B]]\!]_\Gamma \parallel \langle x, W, R\rangle \parallel M \to t : [\![\mathtt{H}[\mathtt{rd}(x).\hat{}\,A + B]]\!]_{\Gamma \cup (W \setminus t)} \parallel \langle x, W, R \cup t\rangle \parallel M$$

Example:

$$t : [\![\mathtt{wr}(x).\hat{}\,\mathtt{rd}(y)]\!]_\emptyset \parallel \langle x, \{t\}, \emptyset\rangle \parallel \langle y, \emptyset, \emptyset\rangle$$

$$\to$$

Read rule

$$(\textsc{Rd})\ t : [\![\mathtt{H}[{}^{\wedge}\mathtt{rd}(x).A + B]]\!]_{\Gamma} \parallel \langle x, W, R\rangle \parallel M \to t : [\![\mathtt{H}[\mathtt{rd}(x).{}^{\wedge}A + B]]\!]_{\Gamma \cup (W \setminus t)} \parallel \langle x, W, R \cup t\rangle \parallel M$$

Example:

$$t : [\![\mathtt{wr}(x).{}^{\wedge}\mathtt{rd}(y)]\!]_{\emptyset} \parallel \langle x, \{t\}, \emptyset\rangle \parallel \langle y, \emptyset, \emptyset\rangle$$

$$\to$$

$$t : [\![\mathtt{wr}(x).\mathtt{rd}(y)^{\wedge}]\!]_{\emptyset} \parallel$$

Read rule

$$(\textsc{Rd})\ t : [\![\mathtt{H}[{}^{\wedge}\mathtt{rd}(x).A + B]]\!]_{\Gamma} \parallel \langle x, W, R\rangle \parallel M \to t : [\![\mathtt{H}[\mathtt{rd}(x).{}^{\wedge}A + B]]\!]_{\Gamma \cup (W \setminus t)} \parallel \langle x, W, R \cup t\rangle \parallel M$$

Example:

$$t : [\![\mathtt{wr}(x).{}^{\wedge}\mathtt{rd}(y)]\!]_{\emptyset} \parallel \langle x, \{t\}, \emptyset\rangle \parallel \langle y, \emptyset, \emptyset\rangle$$

$$\to$$

$$t : [\![\mathtt{wr}(x).\mathtt{rd}(y)^{\wedge}]\!]_{\emptyset} \parallel \langle x, \{t\}, \emptyset\rangle \parallel$$

Read rule

$$(\textsc{Rd})\ t : [\![\mathtt{H}[{}^{\wedge}\mathtt{rd}(x).A + B]]\!]_{\Gamma} \parallel \langle x, W, R\rangle \parallel M \to t : [\![\mathtt{H}[\mathtt{rd}(x).{}^{\wedge}A + B]]\!]_{\Gamma \cup (W \setminus t)} \parallel \langle x, W, R \cup t\rangle \parallel M$$

Example:

$$t : [\![\mathtt{wr}(x).{}^{\wedge}\mathtt{rd}(y)]\!]_{\emptyset} \parallel \langle x, \{t\}, \emptyset\rangle \parallel \langle y, \emptyset, \emptyset\rangle$$

$$\to$$

$$t : [\![\mathtt{wr}(x).\mathtt{rd}(y)^{\wedge}]\!]_{\emptyset} \parallel \langle x, \{t\}, \emptyset\rangle \parallel \langle y, \emptyset, \{t\}\rangle$$

Read rule

$$(\textsc{Rd})\ t : [\![\mathtt{H}[\hat{}\,\mathtt{rd}(x).A + B]]\!]_{\Gamma} \parallel \langle x, W, R\rangle \parallel M \to t : [\![\mathtt{H}[\mathtt{rd}(x).\hat{}\,A + B]]\!]_{\Gamma \cup (W \setminus t)} \parallel \langle x, W, R \cup t\rangle \parallel M$$

Example:

$$t : [\![\mathtt{wr}(x).\hat{}\,\mathtt{rd}(y)]\!]_{\emptyset} \parallel \langle x, \{t\}, \emptyset\rangle \parallel \langle y, \emptyset, \emptyset\rangle$$

$$\to$$

$$t : [\![\mathtt{wr}(x).\mathtt{rd}(y)\hat{}\,]\!]_{\emptyset} \parallel \langle x, \{t\}, \emptyset\rangle \parallel \langle y, \emptyset, \{t\}\rangle$$

◦ Transaction t can commit since all the actions inside of the transaction are executed. When a transaction commits, it is discarded and its identifier is removed from the memory. Then we have:

$$\mathbf{0} \parallel \langle x, \emptyset, \emptyset\rangle \parallel \langle y, \emptyset, \emptyset\rangle$$

Rollback operator

- a rollback operator is implemented following the approach given in [4].

Rollback operator

The rollback operator on the transaction $t : [\![A]\!]_\Gamma$, written $\mathbf{roll}(t)$, is defined as: $\mathbf{roll}(t) = t : [\![\hat{} A]\!]_\emptyset$.

Example: consider transaction

$$t : [\![\mathtt{rd}(x).\mathtt{rd}(y).\hat{}\mathtt{wr}(z).\mathtt{wr}(x')]\!]_\Gamma$$

then rollback of transaction t is:

$$\mathbf{roll}(t) = t : [\![\hat{}\mathtt{rd}(x).\mathtt{rd}(y).\mathtt{wr}(z).\mathtt{wr}(x')]\!]_\emptyset$$

[4] *I. Lanese, C. A. Mezzina, A. Schmitt and J-B. Stefani. Controlling Reversibility in Higher-Order*

The example from introduction:

```
int x = 0;
int y = 5;
int z = 0;
```

```
void t1()
{z = y+x;   }
```

```
void t2()
{ x = z+1; }
```

- Abstracting away from the read and write values contained in variables and representing accesses of two threads to the shared memory in our framework with transactions t_1 and t_2, we obtain a system:

$$t_1 : [\![^\wedge \mathtt{rd}(y).\mathtt{rd}(x).\mathtt{wr}(z)]\!]_\emptyset \mid t_2 : [\![^\wedge \mathtt{rd}(z).\mathtt{wr}(x)]\!]_\emptyset \parallel \langle x, \emptyset, \emptyset\rangle \parallel \langle y, \emptyset, \emptyset\rangle \parallel \langle z, \emptyset, \emptyset\rangle$$

The example from introduction:

```
int x = 0;
int y = 5;
int z = 0;
```

```
void t1()
{z = y+x; }
```

```
void t2()
{ x = z+1; }
```

- Abstracting away from the read and write values contained in variables and representing accesses of two threads to the shared memory in our framework with transactions t_1 and t_2, we obtain a system:

$$t_1 : [\![\hat{}\mathtt{rd}(y).\mathtt{rd}(x).\mathtt{wr}(z)]\!]_{\emptyset} \mid t_2 : [\![\hat{}\mathtt{rd}(z).\mathtt{wr}(x)]\!]_{\emptyset} \parallel \langle x, \emptyset, \emptyset \rangle \parallel \langle y, \emptyset, \emptyset \rangle \parallel \langle z, \emptyset, \emptyset \rangle$$

- After executing all read accesses of both transaction, we have:

$$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).\hat{}\mathtt{wr}(z)]\!]_{\emptyset} \mid t_2 : [\![\mathtt{rd}(z).\hat{}\mathtt{wr}(x)]\!]_{\emptyset} \parallel \langle x, \emptyset, \{t_1\} \rangle \parallel \langle y, \emptyset, \{t_1\} \rangle \parallel \langle z, \emptyset, \{t_2\} \rangle$$

Reader preference

- With *reader preference*, we intend that no read access should be suspended.

- **Example:**

$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).^\wedge\mathtt{wr}(z)]\!]_\emptyset \mid t_2 : [\![\mathtt{rd}(z).^\wedge\mathtt{wr}(x)]\!]_\emptyset \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel \langle z, \emptyset, \{t_2\}\rangle$

.

Reader preference

- With *reader preference*, we intend that no read access should be suspended.

- **Example:**

$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).\hat{}\mathtt{wr}(z)]\!]_{\emptyset} \mid t_2 : [\![\mathtt{rd}(z).\hat{}\mathtt{wr}(x)]\!]_{\emptyset} \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel \langle z, \emptyset, \{t_2\}\rangle$

- write access to variable z, done by transaction t_1, is impossible since in the memory $\langle z, \emptyset, \{t_2\}\rangle$ we can see that some other transaction i.e. t_2 already had a read access to the same variable. Therefore, transaction t_1 will roll back.

.

Reader preference

- With *reader preference*, we intend that no read access should be suspended.

- **Example:**

$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).\hat{}\mathtt{wr}(z)]\!]_{\emptyset} \mid t_2 : [\![\mathtt{rd}(z).\hat{}\mathtt{wr}(x)]\!]_{\emptyset} \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel \langle z, \emptyset, \{t_2\}\rangle$

- write access to variable z, done by transaction t_1, is impossible since in the memory $\langle z, \emptyset, \{t_2\}\rangle$ we can see that some other transaction i.e. t_2 already had a read access to the same variable. Therefore, transaction t_1 will roll back.

$$\mathbf{roll}(t_1) \mid$$

.

Reader preference

- With *reader preference*, we intend that no read access should be suspended.

- **Example:**

$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).^{\wedge}\mathtt{wr}(z)]\!]_{\emptyset} \mid t_2 : [\![\mathtt{rd}(z).^{\wedge}\mathtt{wr}(x)]\!]_{\emptyset} \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel \langle z, \emptyset, \{t_2\}\rangle$

- write access to variable z, done by transaction t_1, is impossible since in the memory $\langle z, \emptyset, \{t_2\}\rangle$ we can see that some other transaction i.e. t_2 already had a read access to the same variable. Therefore, transaction t_1 will roll back.

$$\mathbf{roll}(t_1) \mid t_2 : [\![\mathtt{rd}(z).^{\wedge}\mathtt{wr}(x)]\!]_{\emptyset} \parallel$$

.

Reader preference

- With *reader preference*, we intend that no read access should be suspended.

- **Example:**

$$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).\hat{}\mathtt{wr}(z)]\!]_{\emptyset} \mid t_2 : [\![\mathtt{rd}(z).\hat{}\mathtt{wr}(x)]\!]_{\emptyset} \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel \langle z, \emptyset, \{t_2\}\rangle$$

- write access to variable z, done by transaction t_1, is impossible since in the memory $\langle z, \emptyset, \{t_2\}\rangle$ we can see that some other transaction i.e. t_2 already had a read access to the same variable. Therefore, transaction t_1 will roll back.

$$\mathbf{roll}(t_1) \mid t_2 : [\![\mathtt{rd}(z).\hat{}\mathtt{wr}(x)]\!]_{\emptyset} \parallel \langle x, \emptyset, \emptyset\rangle \parallel$$

.

Reader preference

- With *reader preference*, we intend that no read access should be suspended.

- **Example:**

$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).^{\wedge}\mathtt{wr}(z)]\!]_{\emptyset} \mid t_2 : [\![\mathtt{rd}(z).^{\wedge}\mathtt{wr}(x)]\!]_{\emptyset} \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel \langle z, \emptyset, \{t_2\}\rangle$

- write access to variable z, done by transaction t_1, is impossible since in the memory $\langle z, \emptyset, \{t_2\}\rangle$ we can see that some other transaction i.e. t_2 already had a read access to the same variable. Therefore, transaction t_1 will roll back.

$$\mathbf{roll}(t_1) \mid t_2 : [\![\mathtt{rd}(z).^{\wedge}\mathtt{wr}(x)]\!]_{\emptyset} \parallel \langle x, \emptyset, \emptyset\rangle \parallel \langle y, \emptyset, \emptyset\rangle \parallel$$

.

Reader preference

- With *reader preference*, we intend that no read access should be suspended.

- **Example:**

$$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).\hat{}\mathtt{wr}(z)]\!]_{\emptyset} \mid t_2 : [\![\mathtt{rd}(z).\hat{}\mathtt{wr}(x)]\!]_{\emptyset} \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel \langle z, \emptyset, \{t_2\}\rangle$$

- write access to variable z, done by transaction t_1, is impossible since in the memory $\langle z, \emptyset, \{t_2\}\rangle$ we can see that some other transaction i.e. t_2 already had a read access to the same variable. Therefore, transaction t_1 will roll back.

$$\mathbf{roll}(t_1) \mid t_2 : [\![\mathtt{rd}(z).\hat{}\mathtt{wr}(x)]\!]_{\emptyset} \parallel \langle x, \emptyset, \emptyset\rangle \parallel \langle y, \emptyset, \emptyset\rangle \parallel \langle z, \emptyset, \{t_2\}\rangle$$

.

Reader preference

- With *reader preference*, we intend that no read access should be suspended.

- **Example:**

$$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).{}^\wedge\mathtt{wr}(z)]\!]_\emptyset \mid t_2 : [\![\mathtt{rd}(z).{}^\wedge\mathtt{wr}(x)]\!]_\emptyset \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel \langle z, \emptyset, \{t_2\}\rangle$$

- write access to variable z, done by transaction t_1, is impossible since in the memory $\langle z, \emptyset, \{t_2\}\rangle$ we can see that some other transaction i.e. t_2 already had a read access to the same variable. Therefore, transaction t_1 will roll back.

$$\mathbf{roll}(t_1) \mid t_2 : [\![\mathtt{rd}(z).{}^\wedge\mathtt{wr}(x)]\!]_\emptyset \parallel \langle x, \emptyset, \emptyset\rangle \parallel \langle y, \emptyset, \emptyset\rangle \parallel \langle z, \emptyset, \{t_2\}\rangle$$

where $\mathbf{roll}(t_1) = t_1 : [\![{}^\wedge\mathtt{rd}(y).\mathtt{rd}(x).\mathtt{wr}(z)]\!]_\emptyset$.

Writer preference

- *Writer preference*, allows the write access even if some read access already took place. In this case, all the executing transactions with the read access to the same variable need to be rolled back.

- **Example:**

$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).\hat{}\mathtt{wr}(z)]\!]_{\emptyset} \mid t_2 : [\![\mathtt{rd}(z).\hat{}\mathtt{wr}(x)]\!]_{\emptyset} \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel \langle z, \emptyset, \{t_2\}\rangle$

Writer preference

- *Writer preference*, allows the write access even if some read access already took place. In this case, all the executing transactions with the read access to the same variable need to be rolled back.

- **Example:**

$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).\hat{}\mathtt{wr}(z)]\!]_{\emptyset} \mid t_2 : [\![\mathtt{rd}(z).\hat{}\mathtt{wr}(x)]\!]_{\emptyset} \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel \langle z, \emptyset, \{t_2\}\rangle$

- in this case, write access to variable z can be done and as a consequence transition t_2, having read access to variable z need to be rolled back.

Writer preference

- *Writer preference*, allows the write access even if some read access already took place. In this case, all the executing transactions with the read access to the same variable need to be rolled back.

- **Example:**

$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).\hat{}\mathtt{wr}(z)]\!]_\emptyset \mid t_2 : [\![\mathtt{rd}(z).\hat{}\mathtt{wr}(x)]\!]_\emptyset \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel \langle z, \emptyset, \{t_2\}\rangle$

- in this case, write access to variable z can be done and as a consequence transition t_2, having read access to variable z need to be rolled back.

$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).\mathtt{wr}(z)\hat{}]\!]_\emptyset \mid$

Writer preference

- *Writer preference*, allows the write access even if some read access already took place. In this case, all the executing transactions with the read access to the same variable need to be rolled back.

- **Example:**

$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).{}^{\wedge}\mathtt{wr}(z)]\!]_{\emptyset} \mid t_2 : [\![\mathtt{rd}(z).{}^{\wedge}\mathtt{wr}(x)]\!]_{\emptyset} \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel \langle z, \emptyset, \{t_2\}\rangle$

- in this case, write access to variable z can be done and as a consequence transition t_2, having read access to variable z need to be rolled back.

$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).\mathtt{wr}(z)^{\wedge}]\!]_{\emptyset} \mid \mathbf{roll}(t_2) \parallel$

Writer preference

- *Writer preference*, allows the write access even if some read access already took place. In this case, all the executing transactions with the read access to the same variable need to be rolled back.

- **Example:**

$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).\hat{}\mathtt{wr}(z)]\!]_\emptyset \mid t_2 : [\![\mathtt{rd}(z).\hat{}\mathtt{wr}(x)]\!]_\emptyset \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel \langle z, \emptyset, \{t_2\}\rangle$

- in this case, write access to variable z can be done and as a consequence transition t_2, having read access to variable z need to be rolled back.

$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).\mathtt{wr}(z)\hat{}]\!]_\emptyset \mid \mathbf{roll}(t_2) \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel$

Writer preference

- *Writer preference*, allows the write access even if some read access already took place. In this case, all the executing transactions with the read access to the same variable need to be rolled back.

- **Example:**

$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).\hat{}\mathtt{wr}(z)]\!]_\emptyset \mid t_2 : [\![\mathtt{rd}(z).\hat{}\mathtt{wr}(x)]\!]_\emptyset \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel \langle z, \emptyset, \{t_2\}\rangle$

- in this case, write access to variable z can be done and as a consequence transition t_2, having read access to variable z need to be rolled back.

$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).\mathtt{wr}(z)\hat{}]\!]_\emptyset \mid \mathbf{roll}(t_2) \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel$

Writer preference

- *Writer preference*, allows the write access even if some read access already took place. In this case, all the executing transactions with the read access to the same variable need to be rolled back.

- **Example:**

$$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).{}^{\wedge}\mathtt{wr}(z)]\!]_{\emptyset} \mid t_2 : [\![\mathtt{rd}(z).{}^{\wedge}\mathtt{wr}(x)]\!]_{\emptyset} \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel \langle z, \emptyset, \{t_2\}\rangle$$

- in this case, write access to variable z can be done and as a consequence transition t_2, having read access to variable z need to be rolled back.

$$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).\mathtt{wr}(z)^{\wedge}]\!]_{\emptyset} \mid \mathbf{roll}(t_2) \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel \langle z, \{t_1\}, \emptyset\rangle$$

Writer preference

- *Writer preference*, allows the write access even if some read access already took place. In this case, all the executing transactions with the read access to the same variable need to be rolled back.

- **Example:**

$$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).{}^{\wedge}\mathtt{wr}(z)]\!]_{\emptyset} \mid t_2 : [\![\mathtt{rd}(z).{}^{\wedge}\mathtt{wr}(x)]\!]_{\emptyset} \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel \langle z, \emptyset, \{t_2\}\rangle$$

- in this case, write access to variable z can be done and as a consequence transition t_2, having read access to variable z need to be rolled back.

$$t_1 : [\![\mathtt{rd}(y).\mathtt{rd}(x).\mathtt{wr}(z)^{\wedge}]\!]_{\emptyset} \mid \mathbf{roll}(t_2) \parallel \langle x, \emptyset, \{t_1\}\rangle \parallel \langle y, \emptyset, \{t_1\}\rangle \parallel \langle z, \{t_1\}, \emptyset\rangle$$

where $\mathbf{roll}(t_2) = t_2 : [\![{}^{\wedge}\mathtt{rd}(z).\mathtt{wr}(x)]\!]_{\emptyset}$

Conclusion and Future Work

◦ We have presented a framework to express the STM mechanism in a simple shared memory context.

◦ The framework is able to model two different policies for the execution of the concurrent transactions: writer and reader preference.

◦ Future work:

- our aim is to start from a simple calculus and then to add (in a modular way): nested transactions, data structures (e.g., C structures) and more complex scheduling policies.

- having a modular framework, our goal is to prove that it satisfies the *opacity* property, i.e that all the execution traces of our semantics, where the transactional bodies are interleaved, are equivalent to executions in which transactional blocks are executed as a whole without being interleaved with other transactions.

Thank you for attention ☺

Questions?