Weak memory models using event structures

Simon Castellan[1]

[1]LIP, ENS Lyon

April 3rd, 2016
GaLoP 2016

Unexpected behaviours

A simple concurrent and imperative program:

$$\begin{array}{c} x, y \text{ initialized to } 0 \\ \begin{array}{rcl|| rcl} x & := & 1 & y & := & 2 \\ r & \leftarrow & y & s & \leftarrow & x \end{array} \\ \text{shared variable} \cdot \text{local register} \end{array}$$

Expected outcome: $r \neq 0 \vee s \neq 0$.

Unexpected behaviours

A simple concurrent and imperative program:

$$x, y \text{ initialized to } 0$$
$$\begin{array}{rcl|rcl} x & := & 1 & y & := & 2 \\ r & \leftarrow & y & s & \leftarrow & x \end{array}$$

shared variable · local register

Expected outcome: $r \neq 0 \vee s \neq 0$.
Wrong on modern architectures (x86, ARM, ...).

Unexpected behaviours

A simple concurrent and imperative program:

$$x, y \text{ initialized to } 0$$
$$\begin{array}{rcl|l} r & \leftarrow & y & s \leftarrow x \\ x & := & 1 & y := 2 \end{array}$$

shared variable · local register

Expected outcome: $r \neq 0 \vee s \neq 0$.

Wrong on modern architectures (x86, ARM, ...).

Unexpected behaviours

Another simple program:

$$\begin{array}{lcl|l} x & := & 1 \\ s & \leftarrow & x \\ t & \leftarrow & y \end{array} \Bigg\| \begin{array}{lcl} y & := & 1 \\ s' & \leftarrow & y \\ t' & \leftarrow & x \end{array}$$

Expected outcome: $s = s' = 1 \Rightarrow t = t' = 1$

Unexpected behaviours

Another simple program:

$$\begin{array}{rcl|l} x & := & 1 & y := 1 \\ s & \leftarrow & x & s' \leftarrow y \\ t & \leftarrow & y & t' \leftarrow x \end{array}$$

Expected outcome: $s = s' = 1 \Rightarrow t = t' = 1$

Wrong even without read exchange (*Read Own Write Early*).

A need to specify the behaviour

What are the expected behaviour of a concurrent program?
$\rightarrow$ It depends on the architecture.

Architectures need to be specified:

- what instructions can be reordered?
- how are writes propagated from one thread to the other?

A need to specify the behaviour

What are the expected behaviour of a concurrent program?
$\rightarrow$ It depends on the architecture.

Architectures need to be specified:

- what instructions can be reordered?
- how are writes propagated from one thread to the other?

To that end, manufacturers provide prosaic documents, but:

- *ambiguity*: behaviours that are not specified
- *inconsistent*: some observations may not be predicted.

Some architectures:

- `SC` (*Sequential consistency*): no reordering, sequential memory.
- `ARM`: reordering of instructions targeting different variables, write caches.
- `x86`: ...

Semantics saves the day

Semantics: Formalize mathematically the vendors specifications:

- get a (possibly computer-verified) proof of non-ambiguity,
- implement the specifications and mechanically **test it** against real life architectures.

Two main types of semantics among existing models:

- *operational semantics*: executions are described by runs of an abstract machine,
- *axiomatic semantics*: the notion of valid execution is axiomatized.

Those models are called *weak memory models*.

This talk

Outline of the talk:

1. Reminder on the interpretation of shared memory concurrency in game semantics
2. How to change the interpretation of state to accomodate a concrete model: `x86-TSO`
3. Using those ideas to give a concrete model for the first-order fragment dealing with weak memory models.

Our challenge, `x86-TSO`:

- A read and a write on different memory addresses can be reordered inside a thread.
- A write need not be immediatly commited to main memory. Once it is, it is available to all threads.

I. Usual model of Idealized Parallel Algol

The good ol' IPA

The syntax and semantics of IPA

IPA: Concurrent programming language with shared variables based on the simply-typed λ-calculus.

$$A, B ::= \mathbb{B} \mid \mathbb{N} \mid \mathbf{com} \mid \mathbf{ref} \mid A \to B$$

$$\begin{aligned} M, N ::= {} & \text{...PCF constructs...} \\ & \mid\ !M \mid M := N \mid M; N \\ & \mid \text{new } x \text{ in } t \\ & \mid t \parallel u \end{aligned}$$

Existing games models:

- (Ghica-Murawski) Strategies as sets of non-alternating traces
- (C, Clairambault, Winskel) Strategies as event structures

Talk mostly agnostic about the representation.

Interpretation of the state: pure part

Intepretation of `var` as $\mathbb{N} \times \prod_{n \in \mathbb{N}} \mathtt{com}$:

$$\begin{array}{cccc} \mathtt{re} & \mathtt{wr}_0 & \mathtt{wr}_1 & \dots \\ \vdots & \vdots & \vdots & \\ n & \mathtt{ok} & \mathtt{ok} & \dots \end{array}$$

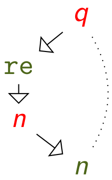
Interpretation of the state: pure part

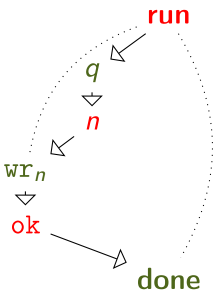
Intepretation of `var` as $\mathbb{N} \times \prod_{n \in \mathbb{N}} \texttt{com}$:

$$\begin{array}{cccc} \texttt{re} & \texttt{wr}_0 & \texttt{wr}_1 & \ldots \\ \vdots & \vdots & \vdots & \\ n & \texttt{ok} & \texttt{ok} & \ldots \end{array}$$

With the following accessors [presented as innocent strategies]:

$\texttt{re} : \quad \mathbf{ref} \to \mathbb{N} \qquad \texttt{wr} : \quad \mathbf{ref} \to \mathbb{N} \to \mathbf{com}$





Interpretation of the state: the `cell` strategy

To interpret the `new` construct (the only one to break innocence):

1. Define the set of traces `cell` as $C\ 0$ with:

$$
\begin{aligned}
C(k) ::= &\ \epsilon \\
&\mid \mathtt{re} \cdot k \cdot C(k) && \text{(Reading from the cell)} \\
&\mid \mathtt{wr}_n \cdot \mathtt{ok} \cdot C(n) && \text{(Writing on the cell)}
\end{aligned}
$$

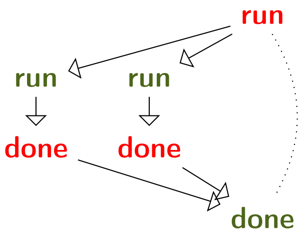
2. Define $[\![\mathrm{new}\ x\ \mathrm{in}\ t]\!] = \mathtt{cell}; [\![t]\!]$ with $x : \mathbf{ref} \vdash t : X$ (up to surgergy)

A term $x : \mathbf{ref} \vdash t : X$ sees x as *volatile* – reading on it can yield any value.
Precomposing with `cell` enforces *a particular memory discipline.*

Interpretation concurrency: $\|$

The strategy $\|$: **com** $\to$ **com** $\to$ **com** is not sequential but is still innocent in a generalized sense.
$\to$ We have now a **dag**:

$$\mathbf{com} \longrightarrow \mathbf{com} \longrightarrow \mathbf{com}$$



Those dags can be composed inside CHO.

Weakening the model

Problem: can we change the interpretation to match TSO?

Two issues:

- *Handling instruction reordering*: how to change `re` and `wr` to model reorderings?
- *Changing the memory discipline*: the memory discipline of TSO depends on the notion of *threads* absent from IPA.

$\rightarrow$ We need a new language to solve this.

II. IPA/x86: A less idealized programming language

A taste of metal in your IPA.

Syntax of IPA/x86

Reading/write to a reference is modified to handle reorderings:

$$M, N ::= \ldots\text{PCF constructs}\ldots$$
$$\mid \mathsf{let}_\iota\ r\ =\ !x \text{ in } N \mid (x :=_\iota M; N) \qquad \iota \in \mathbb{N} \text{ is the thread-id}$$
$$\mid \text{new } x \text{ in } t \mid M \parallel N$$

There is no sequential composition anymore: operations take directly their continuation:

$$\frac{\Gamma, r : \mathbb{N}, x : \mathbf{ref} \vdash N : \mathbf{com} \qquad \Gamma \vdash M : \mathbf{ref}}{\Gamma \vdash \mathsf{let}_k\ r\ =\ !x \text{ in } N : \mathbf{com}}$$

$$\frac{\Gamma, x : \mathbf{ref} \vdash M : \mathbf{ref} \qquad \Gamma \vdash N : \mathbf{com}}{\Gamma \vdash x :=_k M; N}$$

New interpretation of **ref**:

$$\begin{array}{cccc} \mathtt{re}^\iota & \mathtt{wr}_0^\iota & \mathtt{wr}_1^\iota & \ldots \\ \vdots & \vdots & \vdots & \\ n & \mathtt{ok} & \mathtt{ok} & \ldots \end{array}$$

A dirty trick

How to interpret those new construct? Naive idea:

- $\texttt{let}_\iota : \textbf{ref} \to (\mathbb{N} \to \textbf{com}) \to \textbf{com}$
- $\texttt{wr}_\iota : \textbf{ref} \to \mathbb{N} \to \textbf{com} \to \textbf{com}$

Not enough to **inspect** of the continuation.

A dirty trick

How to interpret those new construct? Naive idea:

- $\texttt{let}_\iota : \textbf{ref} \to (\mathbb{N} \to \textbf{com}) \to \textbf{com}$
- $\texttt{wr}_\iota : \textbf{ref} \to \mathbb{N} \to \textbf{com} \to \textbf{com}$

Not enough to **inspect** of the continuation.

Use instead:

- $\texttt{let}_\iota : \textbf{ref} \to (\mathbb{N} \to \textbf{ref} \to \textbf{com}) \to \textbf{com}$
- $\texttt{wr}_\iota : \textbf{ref} \to \mathbb{N} \to (\textbf{ref} \to \textbf{com}) \to \textbf{com}$

with:

- $[\![\mathrm{let}_\iota\ r = !x \text{ in } N]\!] = \mathrm{let}_\iota \odot \langle [\![x]\!], \lambda r.\lambda x.[\![N]\!]\rangle$
- $[\![x :=_\iota M; N]\!] = \mathrm{wr}_\iota \odot \langle [\![x]\!], \lambda x.[\![N]\!]\rangle$

A sequential interpretation for let

$$\mathsf{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$

A sequential interpretation for let

$$\mathrm{let}_\iota : \quad \textbf{ref} \to (\mathbb{N} \to \textbf{ref} \to \textbf{com}) \to \textbf{com}$$

run

A sequential interpretation for let

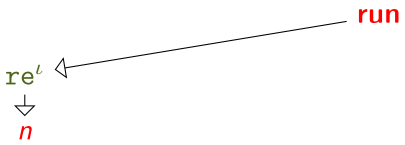
$$\mathrm{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$

run

$\mathtt{re}^\iota$

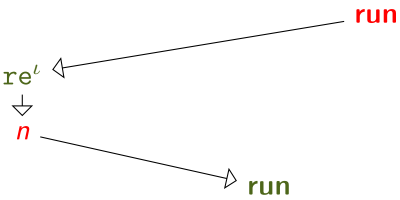
A sequential interpretation for let

$$\mathrm{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$



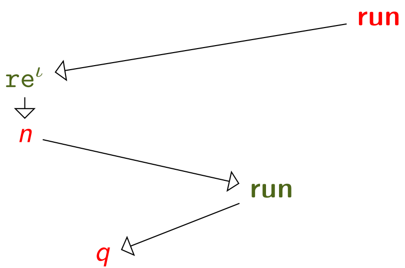
A sequential interpretation for let

$$\mathsf{let}_\iota : \quad \textbf{ref} \to (\mathbb{N} \to \textbf{ref} \to \textbf{com}) \to \textbf{com}$$



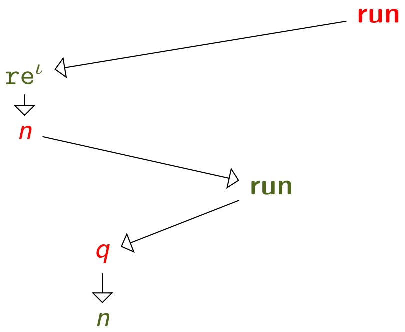
A sequential interpretation for let

$$\mathsf{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$



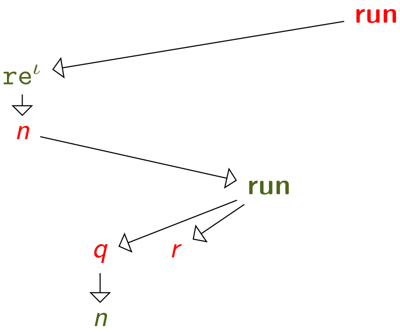
A sequential interpretation for let

$$\mathrm{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$



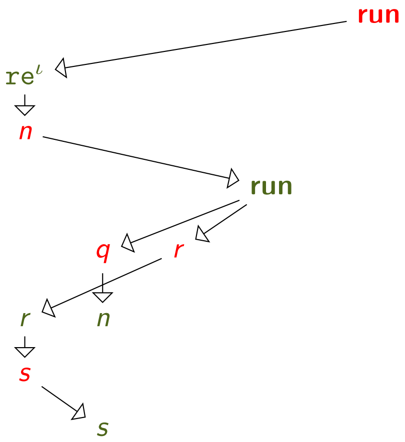
A sequential interpretation for let

$$\mathrm{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$



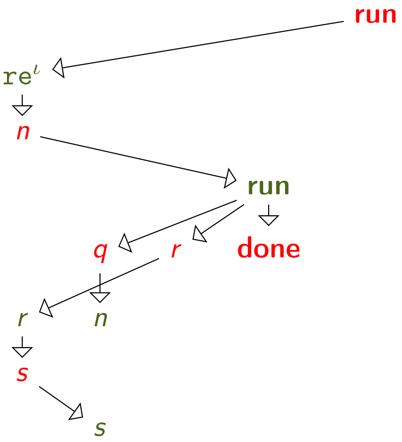
A sequential interpretation for let

$$\mathrm{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$



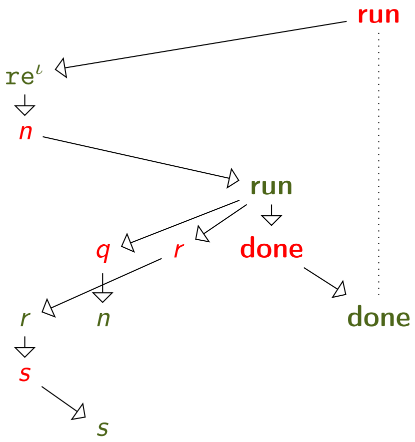
A sequential interpretation for let

$$\text{let}_\iota : \quad \textbf{ref} \to (\mathbb{N} \to \textbf{ref} \to \textbf{com}) \to \textbf{com}$$



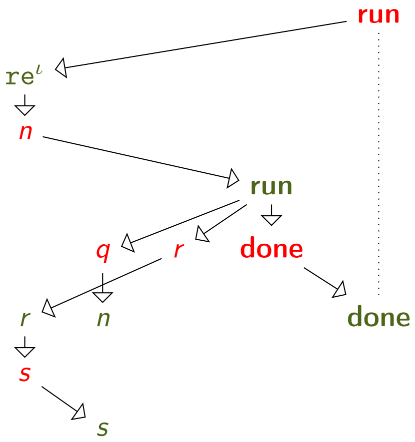
A sequential interpretation for let

$$\mathrm{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$



A sequential interpretation for let

$$\mathrm{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$



Still sequential. Can we use the space to do better?

A concurrent interpretation for let

Idea: start the evaluation of both arguments *in parallel*.

$$\mathsf{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$

A concurrent interpretation for let

Idea: start the evaluation of both arguments *in parallel*.

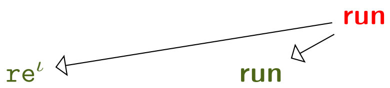
$$\mathrm{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$

run

A concurrent interpretation for let

Idea: start the evaluation of both arguments *in parallel*.

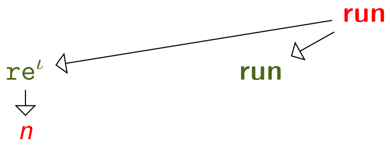
$$\mathrm{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$



A concurrent interpretation for let

Idea: start the evaluation of both arguments *in parallel*.

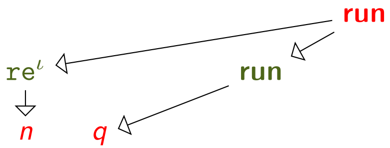
$$\mathrm{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$



A concurrent interpretation for let

Idea: start the evaluation of both arguments *in parallel*.

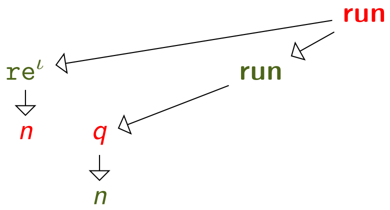
$$\mathrm{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$



A concurrent interpretation for let

Idea: start the evaluation of both arguments *in parallel*.

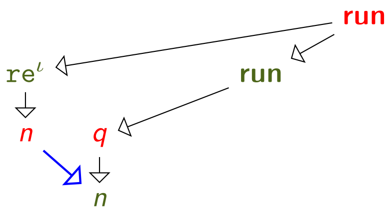
$$\mathrm{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$



A concurrent interpretation for let

Idea: start the evaluation of both arguments *in parallel*.

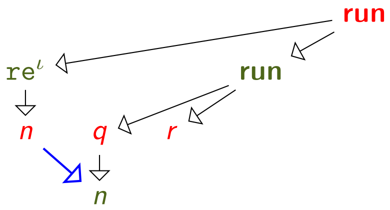
$$\text{let}_\iota : \quad \textbf{ref} \to (\mathbb{N} \to \textbf{ref} \to \textbf{com}) \to \textbf{com}$$



A concurrent interpretation for let

Idea: start the evaluation of both arguments *in parallel*.

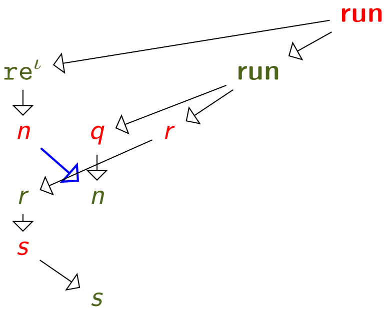
$$\mathsf{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$



A concurrent interpretation for let

Idea: start the evaluation of both arguments *in parallel*.

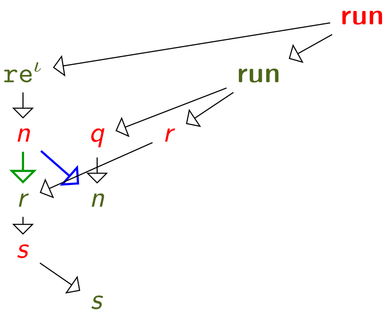
$$\mathrm{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$



A concurrent interpretation for let

Idea: start the evaluation of both arguments *in parallel*.

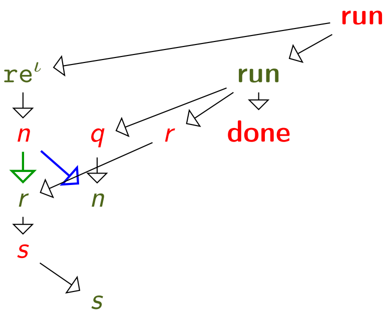
$$\mathsf{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$



A concurrent interpretation for let

Idea: start the evaluation of both arguments *in parallel*.

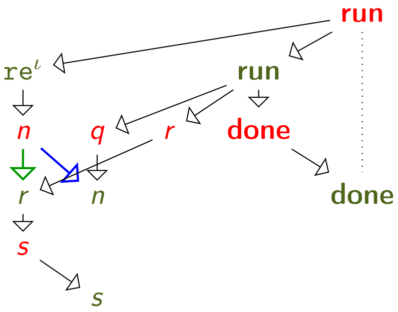
$$\mathrm{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$



A concurrent interpretation for let

Idea: start the evaluation of both arguments *in parallel*.

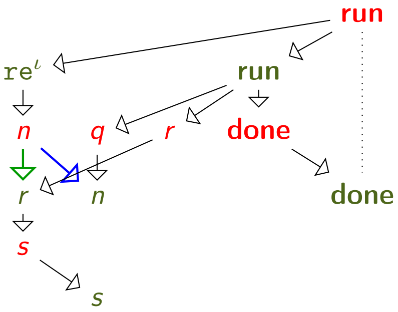
$$\mathrm{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$



A concurrent interpretation for let

Idea: start the evaluation of both arguments *in parallel*.

$$\mathrm{let}_\iota : \quad \mathbf{ref} \to (\mathbb{N} \to \mathbf{ref} \to \mathbf{com}) \to \mathbf{com}$$



Two synchronizations points:

- One for data dependency
- One to sequentialize operations on x

The strategy for `wr` is done similarly (without data dependency).

Examples

$$\text{Take } x : \mathbf{ref}, y : \mathbf{ref} \vdash M = \begin{array}{l} \mathsf{let}_0 r = !x \text{ in} \\ \mathsf{let}_0 s = !y \text{ in} \\ y :=_0 r \end{array} : \mathbf{com}.$$

Examples

$$\text{Take } x : \mathbf{ref}, y : \mathbf{ref} \vdash M = \begin{array}{l} \mathsf{let}_0\, r = !x \text{ in} \\ \mathsf{let}_0\, s = !y \text{ in} \\ y :=_0 r \end{array} : \mathbf{com}.$$

$$x : \mathbf{ref} \to y : \mathbf{ref} \to \mathbf{com}$$

run

Examples

$$\text{Take } x : \mathbf{ref}, y : \mathbf{ref} \vdash M = \begin{array}{l} \mathsf{let}_0\, r = !x \text{ in} \\ \mathsf{let}_0\, s = !y \text{ in} \\ y :=_0 r \end{array} : \mathbf{com}.$$

$$x : \mathbf{ref} \to y : \mathbf{ref} \to \mathbf{com}$$

$$\begin{tikzcd} & \mathbf{run} \arrow[dl] \\ \mathtt{re}^0 & \end{tikzcd}$$

Examples

$$\text{Take } x : \mathbf{ref}, y : \mathbf{ref} \vdash M = \begin{array}{c} \mathrm{let}_0 r = !x \text{ in} \\ \mathrm{let}_0 s = !y \text{ in} \\ y :=_0 r \end{array} : \mathbf{com}.$$

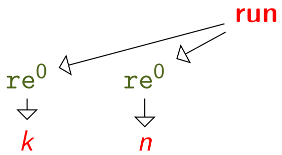
$$x : \mathbf{ref} \to y : \mathbf{ref} \to \mathbf{com}$$

$$\begin{tikzcd} & & \mathbf{run} \arrow[dll] \arrow[dl] \\ \mathtt{re}^0 & \mathtt{re}^0 & \end{tikzcd}$$

Examples

$$\text{Take } x : \mathbf{ref}, y : \mathbf{ref} \vdash M = \begin{array}{c} \mathsf{let}_0\, r = !x \text{ in} \\ \mathsf{let}_0\, s = !y \text{ in} \\ y :=_0 r \end{array} : \mathbf{com}.$$

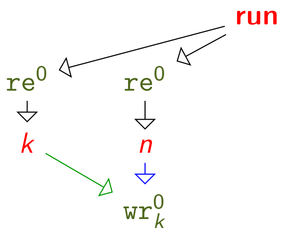
$$x : \mathbf{ref} \to y : \mathbf{ref} \to \mathbf{com}$$



Examples

$$\text{Take } x : \mathbf{ref}, y : \mathbf{ref} \vdash M = \begin{array}{l} \mathrm{let}_0\, r = !x \text{ in} \\ \mathrm{let}_0\, s = !y \text{ in} \\ y :=_0 r \end{array} : \mathbf{com}.$$

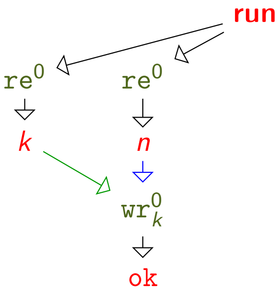
$$x : \mathbf{ref} \to y : \mathbf{ref} \to \mathbf{com}$$



Examples

$$\text{Take } x : \mathbf{ref}, y : \mathbf{ref} \vdash M = \begin{array}{c} \mathsf{let}_0\, r = !x \text{ in} \\ \mathsf{let}_0\, s = !y \text{ in} \\ y :=_0 r \end{array} : \mathbf{com}.$$

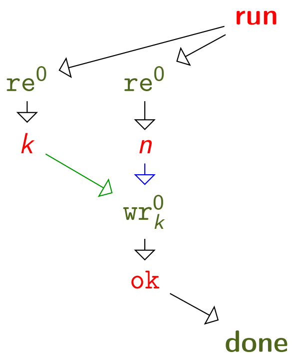
$$x : \mathbf{ref} \to y : \mathbf{ref} \to \mathbf{com}$$



Examples

$$\text{Take } x : \mathbf{ref}, y : \mathbf{ref} \vdash M = \begin{array}{c} \mathsf{let}_0\, r = !x \text{ in} \\ \mathsf{let}_0\, s = !y \text{ in} \\ y :=_0 r \end{array} : \mathbf{com}.$$

$$x : \mathbf{ref} \to y : \mathbf{ref} \to \mathbf{com}$$



A strategy `cell` that represents a TSO memory

Instruction reorderings: check. What about relaxed memory?

→ Update the grammar of valid traces. Now parametrized over:

- a *global value n*
- *thread-local* local values $\mu : \mathbb{N} \to \mathbb{N} \cup \{\star\}$

$$
\begin{aligned}
C(\mu, n) ::= {} & \epsilon \\
& \mid \mathtt{re}^{\iota} \cdot \mu(\iota) \cdot C(\mu, n) && (\mu(\iota) \neq \star) \\
& \qquad \text{(Read from local cache)} \\
& \mid \mathtt{re}^{\iota} \cdot n \cdot C(\mu, n) && (\mu(\iota) = \star) \\
& \qquad \text{(Read from the global memory)} \\
& \mid \mathtt{wr}^{\iota}_{k} \cdot ok \cdot C(\mu[\iota \leftarrow k], n) \\
& \qquad \text{(Write to local cache)} \\
& \mid C(\mu[\iota \leftarrow \star], \mu(\iota)) && (\mu(\iota) \neq \star) \\
& \qquad \text{(Committing a write)}
\end{aligned}
$$

This gives a strategy $\mathtt{cell}_{\mathtt{TSO}}$.

Example

Remember the program at the beginning:

$$\begin{array}{l} x :=_0 1; \\ \mathsf{let}_0\ s\ = !x\ \mathsf{in} \\ \mathsf{let}_0\ t\ = !y\ \mathsf{in}\ () \end{array} \;\Bigg\|\; \begin{array}{l} y :=_0 1; \\ \mathsf{let}_0\ s'\ = !y\ \mathsf{in} \\ \mathsf{let}_0\ t'\ = !x\ \mathsf{in}\ () \end{array}$$

The following is a valid trace of $\mathtt{cell}_{\mathtt{TSO}}$ (hence can describe a valid interaction on x for instance)

$$\mathtt{wr}_1^0 \cdot \mathtt{ok} \cdot \mathtt{re}^0 \cdot 1 \cdot \mathtt{re}^1 \cdot 0$$

Example

Remember the program at the beginning:

$$\begin{array}{l} x :=_0 1; \\ \mathsf{let}_0\ s\ = !x\ \mathsf{in} \\ \mathsf{let}_0\ t\ = !y\ \mathsf{in}\ () \end{array} \;\Bigg\| \; \begin{array}{l} y :=_0 1; \\ \mathsf{let}_0\ s'\ = !y\ \mathsf{in} \\ \mathsf{let}_0\ t'\ = !x\ \mathsf{in}\ () \end{array}$$

The following is a valid trace of $\mathtt{cell}_{\mathtt{TSO}}$ (hence can describe a valid interaction on x for instance)

$$\mathtt{wr}_1^0 \cdot \mathtt{ok} \cdot \mathtt{re}^0 \cdot 1 \cdot \mathtt{re}^1 \cdot 0$$

Example

Remember the program at the beginning:

$$\begin{array}{l} x :=_0 1; \\ \mathsf{let}_0\ s\ = !x\ \mathsf{in} \\ \mathsf{let}_0\ t\ = !y\ \mathsf{in}\ () \end{array} \Bigg\| \begin{array}{l} y :=_0 1; \\ \mathsf{let}_0\ s'\ = !y\ \mathsf{in} \\ \mathsf{let}_0\ t'\ = !x\ \mathsf{in}\ () \end{array}$$

The following is a valid trace of $\mathtt{cell}_{\mathtt{TSO}}$ (hence can describe a valid interaction on x for instance)

$$\mathtt{wr}_1^0 \cdot \mathtt{ok} \cdot \mathtt{re}^0 \cdot 1 \cdot \mathtt{re}^1 \cdot 0$$

Example

Remember the program at the beginning:

$$\begin{array}{l} x :=_0 1; \\ \mathsf{let}_0\ s\ = !x\ \mathsf{in} \\ \mathsf{let}_0\ t\ = !y\ \mathsf{in}\ () \end{array} \;\Big\|\; \begin{array}{l} y :=_0 1; \\ \mathsf{let}_0\ s'\ = !y\ \mathsf{in} \\ \mathsf{let}_0\ t'\ = !x\ \mathsf{in}\ () \end{array}$$

The following is a valid trace of $\texttt{cell}_{\texttt{TSO}}$ (hence can describe a valid interaction on x for instance)

$$\texttt{wr}_1^0 \cdot \texttt{ok} \cdot \texttt{re}^0 \cdot 1 \cdot \texttt{re}^1 \cdot 0$$

III. The first-order case

Filtering out the higher-order hop

Getting rid of game semantics

- To study the weak memory aspects, the first-order is enough.

  $\rightarrow$ Simpler presentation of the ideas in the first-order fragment?
  (terms of the form $x : \mathbf{ref}, y : \mathbf{ref} \vdash t : \mathbf{com}$)

- Eliminate game semantics bureaucracy:
  - Consider the projection on the left-hand side (sequence of **ref**)
  - We collapse $\mathtt{re}_x^\iota \cdot k$ into one event $\mathtt{R}_x^{(\iota,k)}$
    (Similarly writes are collapsed in $\mathtt{W}_x^{(\iota,k)}$)

Overview of the model

- The model is parametrized over an architecture $\mathscr{A} = (\rightharpoonup_{\mathscr{A}}, \mathscr{E}_{\mathscr{A}})$ where
  - $\rightharpoonup_{\mathscr{A}}$ is a relation indicating which instruction ordering *cannot* be relaxed.
  - $\mathscr{E}_{\mathscr{A}}$ is an event structure describing the memory discipline.

Overview of the model

- The model is parametrized over an architecture $\mathscr{A} = (\multimap_{\mathscr{A}}, \mathscr{E}_{\mathscr{A}})$ where
  - $\multimap_{\mathscr{A}}$ is a relation indicating which instruction ordering *cannot* be relaxed.
  - $\mathscr{E}_{\mathscr{A}}$ is an event structure describing the memory discipline.
- The semantics is in two steps:
  1. **The volatile part**: which is the pure functional part (before pre-composition with cell).
     Defined by a simple *induction* on the program syntax using $\multimap_{\mathscr{A}}$.
  2. **The closed part**: after pre-composition with `cell`.
     Defined as the *synchronized product* with $\mathscr{E}_{\mathscr{A}}$.

Examples of volatile semantics

For $t = \begin{pmatrix} \text{let } s = x \text{ in } \mathtt{x} := \mathtt{s}; \\ \text{let } r = y \text{ in } \mathtt{y} := \mathtt{r}; \\ z := s + r; () \end{pmatrix}$, we have:

$$\begin{tikzcd}
& \mathtt{R}_x^{(0)} \arrow[rrr, no head, squiggly] \arrow[d] & & & \mathtt{R}_x^{(1)} \arrow[d] & \\
& \mathtt{W}_x^{(0)} \arrow[dl] \arrow[dr] & & & \mathtt{W}_x^{(1)} \arrow[dl] \arrow[dr] & \\
\mathtt{R}_y^{(0)} \arrow[rr, no head, squiggly] \arrow[d] & & \mathtt{R}_y^{(1)} \arrow[d] & \mathtt{R}_y^{(0)} \arrow[rr, no head, squiggly] \arrow[d] & & \mathtt{R}_y^{(1)} \arrow[d] \\
\mathtt{W}_y^{(0)} \arrow[d] & & \mathtt{W}_y^{(1)} \arrow[d] & \mathtt{W}_y^{(0)} \arrow[d] & & \mathtt{W}_y^{(1)} \arrow[d] \\
\mathtt{W}_z^{(0)} & & \mathtt{W}_z^{(1)} & \mathtt{W}_z^{(1)} & & \mathtt{W}_z^{(2)}
\end{tikzcd}$$

$(\rightharpoonup_{\mathrm{SC}})$

Examples of volatile semantics

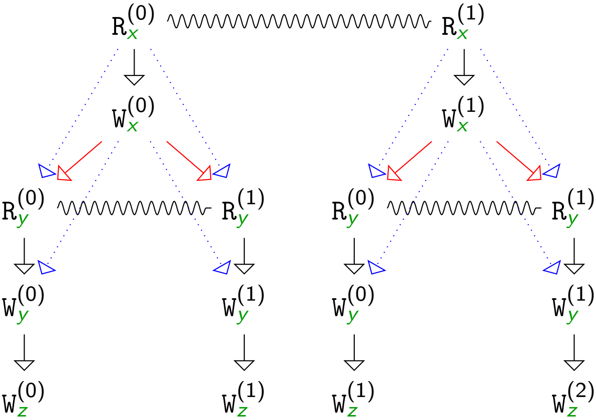
For $t = \begin{pmatrix} \text{let } s = x \text{ in } \mathtt{x} := \mathtt{s}; \\ \text{let } r = y \text{ in } \mathtt{y} := \mathtt{r}; \\ z := s + r; () \end{pmatrix}$, we have:

$$
\begin{tikzcd}
& \mathtt{R}_x^{(0)} \arrow[d] \arrow[rrr, no head, squiggly] & & & \mathtt{R}_x^{(1)} \arrow[d] & \\
& \mathtt{W}_x^{(0)} \arrow[dl, red] \arrow[dr, red] & & & \mathtt{W}_x^{(1)} \arrow[dl, red] \arrow[dr, red] & \\
\mathtt{R}_y^{(0)} \arrow[d] \arrow[rr, no head, squiggly] & & \mathtt{R}_y^{(1)} \arrow[d] & \mathtt{R}_y^{(0)} \arrow[d] \arrow[rr, no head, squiggly] & & \mathtt{R}_y^{(1)} \arrow[d] \\
\mathtt{W}_y^{(0)} \arrow[d] & & \mathtt{W}_y^{(1)} \arrow[d] & \mathtt{W}_y^{(0)} \arrow[d] & & \mathtt{W}_y^{(1)} \arrow[d] \\
\mathtt{W}_z^{(0)} & & \mathtt{W}_z^{(1)} & \mathtt{W}_z^{(1)} & & \mathtt{W}_z^{(2)}
\end{tikzcd}
$$

$(\rightarrowtriangle_{\mathsf{SC}})$

Examples of volatile semantics

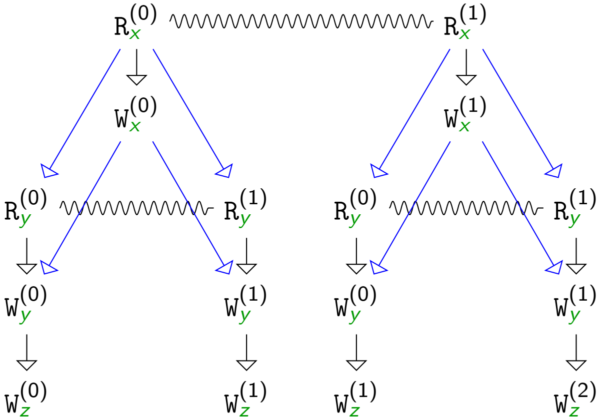
$$\text{For } t = \begin{pmatrix} \text{let } s = x \text{ in } \mathtt{x} := \mathtt{s}; \\ \text{let } r = y \text{ in } \mathtt{y} := \mathtt{r}; \\ z := s + r; () \end{pmatrix}, \text{ we have:}$$



$(\rightarrowtriangle_{\mathrm{SC}})$

Examples of volatile semantics

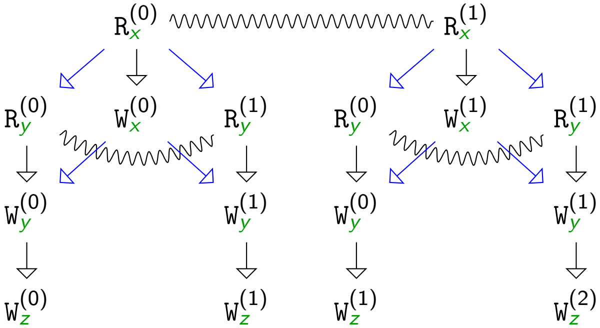
$$\text{For } t = \begin{pmatrix} \text{let } s = x \text{ in } \mathtt{x} := \mathtt{s}; \\ \text{let } r = y \text{ in } \mathtt{y} := \mathtt{r}; \\ z := s + r; () \end{pmatrix}, \text{ we have:}$$



$(\rightarrowtriangle_{\mathsf{SC}})$

Examples of volatile semantics

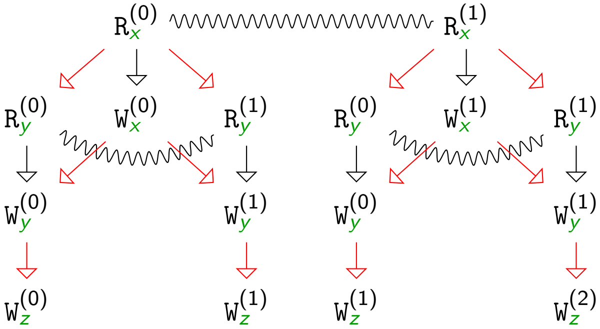
$$\text{For } t = \begin{pmatrix} \text{let } s = x \text{ in } \mathtt{x} := \mathtt{s}; \\ \text{let } r = y \text{ in } \mathtt{y} := \mathtt{r}; \\ z := s + r; () \end{pmatrix}, \text{ we have:}$$



$(\rightarrowtriangle_{\text{x86}})$

Examples of volatile semantics

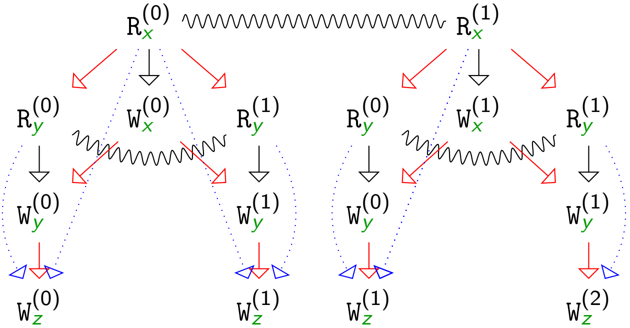
For $t = \begin{pmatrix} \text{let } s = x \text{ in } \mathtt{x} := \mathtt{s}; \\ \text{let } r = y \text{ in } \mathtt{y} := \mathtt{r}; \\ z := s + r; () \end{pmatrix}$, we have:



$(\rightharpoonup_{\text{x86}})$

Examples of volatile semantics

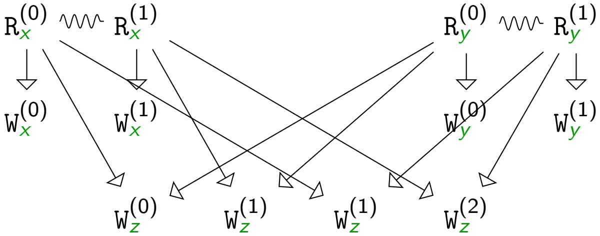
$$\text{For } t = \begin{pmatrix} \text{let } s = x \text{ in } \mathtt{x} := \mathtt{s}; \\ \text{let } r = y \text{ in } \mathtt{y} := \mathtt{r}; \\ z := s + r; () \end{pmatrix}, \text{ we have:}$$



$(\rightarrowtriangle_{x86})$

Examples of volatile semantics

For $t = \begin{pmatrix} \text{let } s = x \text{ in } \mathtt{x} := \mathtt{s}; \\ \text{let } r = y \text{ in } \mathtt{y} := \mathtt{r}; \\ z := s + r; () \end{pmatrix}$, we have:



$(\rightarrowtriangle_{\mathrm{ARM}})$

Example with $\mathscr{E}_{TSO}$

$$
\begin{array}{l|l}
x :=_0 1; & y :=_0 1; \\
\mathrm{let}_0\ s\ = !x\ \mathrm{in} & \mathrm{let}_0\ s'\ = !y\ \mathrm{in} \\
\mathrm{let}_0\ t\ = !y\ \mathrm{in}\ () & \mathrm{let}_0\ t'\ = !x\ \mathrm{in}\ ()
\end{array}
$$

$$
\begin{tikzcd}
 & & \mathtt{W}_x^{(1)} \arrow[dl] \arrow[d] & & & & \mathtt{W}_y^{(1)} \arrow[dl] \arrow[d] & \\
 & \mathtt{R}_x^{(0)} \arrow[dl] \arrow[d] \arrow[r, phantom, "\sim"] & \mathtt{R}_x^{(1)} \arrow[d] \arrow[dr] & & & \mathtt{R}_y^{(0)} \arrow[dl] \arrow[d] \arrow[r, phantom, "\sim"] & \mathtt{R}_y^{(1)} \arrow[d] \arrow[dr] & \\
\mathtt{R}_y^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_y^{(1)} & \mathtt{R}_y^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_y^{(1)} & \mathtt{R}_x^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_x^{(1)} & \mathtt{R}_x^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_x^{(1)}
\end{tikzcd}
$$

(**Volatile semantics** for SC)

Example with $\mathscr{E}_{TSO}$

$$
\begin{array}{l}
x :=_0 1; \\
\mathsf{let}_0\ s\ = !x\ \mathsf{in} \\
\mathsf{let}_0\ t\ = !y\ \mathsf{in}\ ()
\end{array}
\;\Bigg\|\;
\begin{array}{l}
y :=_0 1; \\
\mathsf{let}_0\ s'\ = !y\ \mathsf{in} \\
\mathsf{let}_0\ t'\ = !x\ \mathsf{in}\ ()
\end{array}
$$

$$
\begin{tikzcd}
 & & \mathtt{W}_x^{(1)} \arrow[dl] \arrow[d] & & & & \mathtt{W}_y^{(1)} \arrow[dl] \arrow[d] & \\
 & \mathtt{R}_x^{(0)} \arrow[dl] \arrow[d] \arrow[r, phantom, "\sim"] & \mathtt{R}_x^{(1)} \arrow[d] \arrow[dr] & & & \mathtt{R}_y^{(0)} \arrow[dl] \arrow[d] \arrow[r, phantom, "\sim"] & \mathtt{R}_y^{(1)} \arrow[d] \arrow[dr] & \\
\mathtt{R}_y^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_y^{(1)} & \mathtt{R}_y^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_y^{(1)} & \mathtt{R}_x^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_x^{(1)} & \mathtt{R}_x^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_x^{(1)}
\end{tikzcd}
$$

(Computing $[\![p]\!]_{\mathrm{SC}}^{O} \wedge \mathscr{E}$)

Example with $\mathscr{E}_{TSO}$

$$
\begin{array}{l}
x :=_0 1; \\
\mathsf{let}_0\ s\ = !x\ \mathsf{in} \\
\mathsf{let}_0\ t\ = !y\ \mathsf{in}\ ()
\end{array}
\;\Bigg\|\;
\begin{array}{l}
y :=_0 1; \\
\mathsf{let}_0\ s'\ = !y\ \mathsf{in} \\
\mathsf{let}_0\ t'\ = !x\ \mathsf{in}\ ()
\end{array}
$$

$$
\begin{tikzcd}
 & & \mathtt{W}_x^{(1)} \arrow[dl] \arrow[d] & & & & \mathtt{W}_y^{(1)} \arrow[dl] \arrow[d] & \\
 & \mathtt{R}_x^{(0)} \arrow[dl] \arrow[d] \arrow[r, phantom, "\sim"] & \mathtt{R}_x^{(1)} \arrow[d] \arrow[dr] & & & \mathtt{R}_y^{(0)} \arrow[dl] \arrow[d] \arrow[r, phantom, "\sim"] & \mathtt{R}_y^{(1)} \arrow[d] \arrow[dr] & \\
\mathtt{R}_y^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_y^{(1)} & \mathtt{R}_y^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_y^{(1)} & \mathtt{R}_x^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_x^{(1)} & \mathtt{R}_x^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_x^{(1)}
\end{tikzcd}
$$

(Computing $[\![p]\!]_{\mathrm{SC}}^{O} \wedge \mathscr{E}$)

Example with $\mathscr{E}_{TSO}$

$$
\begin{array}{l|l}
x :=_0 1; & y :=_0 1; \\
\mathsf{let}_0\ s\ = !x\ \mathsf{in} & \mathsf{let}_0\ s'\ = !y\ \mathsf{in} \\
\mathsf{let}_0\ t\ = !y\ \mathsf{in}\ () & \mathsf{let}_0\ t'\ = !x\ \mathsf{in}\ ()
\end{array}
$$

$$
\begin{tikzcd}
\mathtt{W}_x^{(1)} \arrow[d] & & & & \mathtt{W}_y^{(1)} \arrow[dl] \arrow[d] & \\
\mathtt{R}_x^{(1)} \arrow[d] \arrow[dr] & & & \mathtt{R}_y^{(0)} \arrow[dl] \arrow[d] \arrow[r, phantom, "\sim"] & \mathtt{R}_y^{(1)} \arrow[d] \arrow[dr] & \\
\mathtt{R}_y^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_y^{(1)} & \mathtt{R}_x^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_x^{(1)} & \mathtt{R}_x^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_x^{(1)}
\end{tikzcd}
$$

(Computing $[\![p]\!]^{O}_{\mathrm{SC}} \wedge \mathscr{E}$)

Example with $\mathscr{E}_{TSO}$

$$
\begin{array}{l|l}
x :=_0 1; & y :=_0 1; \\
\mathrm{let}_0\ s\ = !x\ \mathrm{in} & \mathrm{let}_0\ s'\ = !y\ \mathrm{in} \\
\mathrm{let}_0\ t\ = !y\ \mathrm{in}\ () & \mathrm{let}_0\ t'\ = !x\ \mathrm{in}\ ()
\end{array}
$$

$$
\begin{tikzcd}
\mathtt{W}_x^{(1)} \arrow[d] & & \mathtt{W}_y^{(1)} \arrow[d] & \\
\mathtt{R}_x^{(1)} \arrow[d] \arrow[dr] & & \mathtt{R}_y^{(1)} \arrow[d] \arrow[dr] & \\
\mathtt{R}_y^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_y^{(1)} & \mathtt{R}_x^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_x^{(1)}
\end{tikzcd}
$$

(Computing $[\![p]\!]_{\mathrm{SC}}^{O} \wedge \mathscr{E}$)

Example with $\mathscr{E}_{TSO}$

$$
\begin{array}{l}
x :=_0 1; \\
\mathrm{let}_0\ s\ = !x \text{ in} \\
\mathrm{let}_0\ t\ = !y \text{ in } ()
\end{array}
\;\Bigg\|\;
\begin{array}{l}
y :=_0 1; \\
\mathrm{let}_0\ s'\ = !y \text{ in} \\
\mathrm{let}_0\ t'\ = !x \text{ in } ()
\end{array}
$$

$$
\begin{tikzcd}
\mathtt{W}_x^{(1)} \arrow[d] & \\
\mathtt{R}_x^{(1)} \arrow[d] \arrow[dr] & \\
\mathtt{R}_y^{(0)} \arrow[r, phantom, "\rightsquigarrow"] & \mathtt{R}_y^{(1)}
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\mathtt{W}_y^{(1)} \arrow[d] & \\
\mathtt{R}_y^{(1)} \arrow[d] \arrow[dr] & \\
\mathtt{R}_x^{(0)} \arrow[r, phantom, "\rightsquigarrow"] & \mathtt{R}_x^{(1)}
\end{tikzcd}
$$

(Computing $[\![p]\!]^{O}_{\mathrm{SC}} \wedge \mathscr{E}$)

Example with $\mathscr{E}_{TSO}$

$$
\begin{array}{l}
x :=_0 1; \\
\mathrm{let}_0\ s = !x \text{ in} \\
\mathrm{let}_0\ t = !y \text{ in } ()
\end{array}
\ \Big\|\
\begin{array}{l}
y :=_0 1; \\
\mathrm{let}_0\ s' = !y \text{ in} \\
\mathrm{let}_0\ t' = !x \text{ in } ()
\end{array}
$$

$$
\begin{tikzcd}
\mathtt{W}_x^{(1)} \arrow[d] & \\
\mathtt{R}_x^{(1)} \arrow[d] \arrow[dr] & \\
\mathtt{R}_y^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_y^{(1)}
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\mathtt{W}_y^{(1)} \arrow[d] & \\
\mathtt{R}_y^{(1)} \arrow[d] \arrow[dr] & \\
\mathtt{R}_x^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_x^{(1)}
\end{tikzcd}
$$

(Computing $[\![p]\!]^{O}_{\mathtt{SC}} \wedge \mathscr{E}$)

Example with $\mathscr{E}_{TSO}$

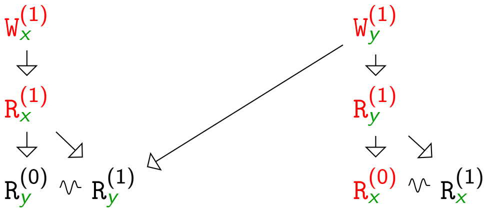
$$
\begin{array}{l|l}
x :=_0 1; & y :=_0 1; \\
\mathrm{let}_0\ s = !x \text{ in} & \mathrm{let}_0\ s' = !y \text{ in} \\
\mathrm{let}_0\ t = !y \text{ in } () & \mathrm{let}_0\ t' = !x \text{ in } ()
\end{array}
$$

$$
\begin{tikzcd}
\mathtt{W}_x^{(1)} \arrow[d] & & \mathtt{W}_y^{(1)} \arrow[d] \arrow[ddl] & \\
\mathtt{R}_x^{(1)} \arrow[d] \arrow[dr] & & \mathtt{R}_y^{(1)} \arrow[d] \arrow[dr] & \\
\mathtt{R}_y^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_y^{(1)} & \mathtt{R}_x^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_x^{(1)}
\end{tikzcd}
$$

(Computing $[\![p]\!]^{O}_{\mathrm{SC}} \wedge \mathscr{E}$)

Example with $\mathscr{E}_{TSO}$

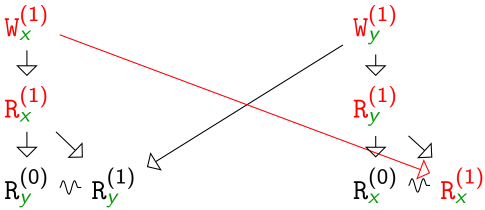
$$
\begin{array}{l}
x :=_0 1; \\
\mathrm{let}_0\ s\ = !x\ \mathrm{in} \\
\mathrm{let}_0\ t\ = !y\ \mathrm{in}\ ()
\end{array}
\ \Bigg\|\
\begin{array}{l}
y :=_0 1; \\
\mathrm{let}_0\ s'\ = !y\ \mathrm{in} \\
\mathrm{let}_0\ t'\ = !x\ \mathrm{in}\ ()
\end{array}
$$



(Computing $[\![p]\!]^{O}_{\mathrm{SC}} \wedge \mathscr{E}$)

Example with $\mathscr{E}_{TSO}$

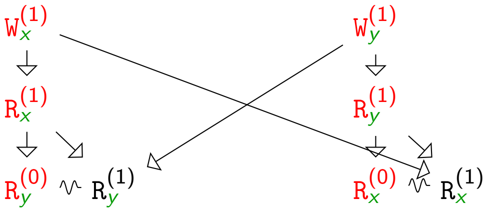
$$\begin{array}{l} x :=_0 1; \\ \mathrm{let}_0\ s\ = !x\ \mathrm{in} \\ \mathrm{let}_0\ t\ = !y\ \mathrm{in}\ () \end{array} \;\Big\Vert\; \begin{array}{l} y :=_0 1; \\ \mathrm{let}_0\ s'\ = !y\ \mathrm{in} \\ \mathrm{let}_0\ t'\ = !x\ \mathrm{in}\ () \end{array}$$



(Computing $[\![p]\!]^{O}_{\mathrm{SC}} \wedge \mathscr{E}$)

Example with $\mathscr{E}_{TSO}$

$$
\begin{array}{l} x :=_0 1; \\ \mathrm{let}_0\ s = !x \ \mathrm{in} \\ \mathrm{let}_0\ t = !y \ \mathrm{in}\ () \end{array}
\;\Big\|\;
\begin{array}{l} y :=_0 1; \\ \mathrm{let}_0\ s' = !y \ \mathrm{in} \\ \mathrm{let}_0\ t' = !x \ \mathrm{in}\ () \end{array}
$$



(Computing $[\![p]\!]^{O}_{\mathtt{SC}} \wedge \mathscr{E}$)

We can observe $s = s' = 1 \wedge t = t' = 0$.

Conclusion

Summary.

- We shown a few ideas how to model weaker memory model using game semantics
- We used those ideas to give a parametric denotational semantics for weak memory models, based on event structures.

Perspectives and future work.

- Adding barriers to the mix
- Link with existing semantics (eg. axiomatic semantics)
- Instruction semantics? (modelling a ASM-like language)