Weak memory models using event structures

Simon Castellan, 17 mars 2016
Journée langages

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:

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

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

Unexpected behaviours

Another simple program:

$$\begin{array}{lcl} x & := & 1 \\ r_1 & \leftarrow & x \\ r_2 & \leftarrow & y \end{array} \;\Bigg\|\; \begin{array}{lcl} y & := & 1 \\ s_1 & \leftarrow & y \\ s_2 & \leftarrow & x \end{array}$$

Expected outcome: $r_1 = s_1 = 1 \Rightarrow r_2 = s_2 = 1$

Unexpected behaviours

Another simple program:

$$\begin{array}{lll|l|lll} x & := & 1 & & y & := & 1 \\ r_1 & \leftarrow & x & & s_1 & \leftarrow & y \\ r_2 & \leftarrow & y & & s_2 & \leftarrow & x \end{array}$$

Expected outcome: $r_1 = s_1 = 1 \Rightarrow r_2 = s_2 = 1$

Wrong even without read exchange (x86: *Read Own Write Early*).

A need to specify the behaviour

What are the expected behaviour of a concurrent programs?
→ It depends on the architectures.

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 programs?
→ It depends on the architectures.

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 the runs of an abstract machines,
- *axiomatic semantics*: the notion of valid execution is axiomatized.

Those models are called *weak memory models*.

Semantics and executions

The semantics generates from a program its possible *executions*:

Program	Some executions
$x := 1 \parallel y := 2$	$\mathtt{W}_x^{(1)} \cdot \mathtt{W}_y^{(2)} \cdot \mathtt{R}_y^{(2)} \cdot \mathtt{R}_x^{(1)}$
$r \leftarrow y \parallel s \leftarrow x$	$\mathtt{W}_y^{(2)} \cdot \mathtt{R}_x^{(0)} \cdot \mathtt{W}_x^{(2)} \cdot \mathtt{R}_y^{(1)}$

Executions can be formalized in different ways: traces, partial-order, ...

This talk

A semantics that is

- **denotational**: executions computed by induction
  - the semantics is thus *compositional*
- **compact**: based on event structures
  - no combinatorial explosion
- **extensible**: inspired from game semantics
  - it is easy to add loops, control operators, higher-order, ...

Outline of the talk:

1. **A semantics warm-up**: compute the SC semantics using *traces*.
2. Getting back the **causality**.
3. Our contribution: A **parametric** semantics using event structures.

I. A DENOTATIONAL SEMANTICS FOR SC

With traces of originality

Syntax precedes semantics

Our very simple programming language:

$$\begin{aligned}
e, e' ::= &\quad \{\ \textit{Expressions}\ \} \\
&\quad k \in \mathbb{N} \mid r \in \mathcal{R} \mid e + e' \\
\iota ::= &\quad \{\ \textit{Instructions}\ \} \\
&\mid a := e & \text{(Write on a variable)} \\
&\mid r \leftarrow a & \text{(Read on a variable)} \\
t ::= &\quad \{\ \textit{Threads}\ \} \\
&\mid \iota; \ldots; \iota \\
p ::= &\quad \{\ \textit{Programs}\ \} \\
&\quad t_1 \parallel \ldots \parallel t_n
\end{aligned}$$

In real life: conditionals and barriers.

Denotational semantics

Goal: compute $[\![t]\!] \in E$ where E is some space of denotations.

Our space here: langages of traces.

$$\begin{aligned} \Sigma_a &= \mathcal{V} \times \{\mathtt{R}, \mathtt{W}\} && \text{(Abstract memory event)} \\ \Sigma_c &= \Sigma_a \times \mathbb{N} && \text{(Concrete memory event)} \\ E &= \mathscr{P}({\Sigma_c}^*) \end{aligned}$$

Notations: $\mathtt{R}_x^{(k)}$, $\mathtt{W}_x^{(k)}$.

Two steps:

1. Volatile semantics $[\![t]\!]^O$: shared variables are considered *volatile*: $[\![x \;\texttt{:=}\; 1; r \leftarrow x]\!]^O$ does not guarantee to read 1 in r.
2. Closed semantics: once $[\![t]\!]^O$ is calculated for the whole program, we restrict the scope of the variable $[\![x \;\texttt{:=}\; 1; r \leftarrow x]\!]$ reads 1 in r.

Volatile semantics

Semantics of threads. Parametrized over $\rho : \mathcal{R} \to \mathbb{N}$.

$$\begin{array}{rl} \text{(Writes)} & [\![x \ \texttt{:=}\ e; t]\!]\rho = \mathtt{W}_x^{(\rho(e))} \cdot [\![t]\!]\rho \\ \text{(Reads)} & [\![r \leftarrow x; t]\!]\rho = \bigcup_{i \in \mathbb{N}} \left(\mathtt{R}_x^{(i)} \cdot [\![t]\!](\rho[r \leftarrow i]) \right) \end{array}$$

Volatile semantics

Semantics of threads. Parametrized over $\rho : \mathcal{R} \to \mathbb{N}$.

$$\begin{array}{rl} \text{(Writes)} & [\![x \;\texttt{:=}\; e; t]\!]\rho = \mathtt{W}_{x}^{(\rho(e))} \cdot [\![t]\!]\rho \\ \text{(Reads)} & [\![r \leftarrow x; t]\!]\rho = \displaystyle\bigcup_{i \in \mathbb{N}} \left(\mathtt{R}_{x}^{(i)} \cdot [\![t]\!](\rho[r \leftarrow i])\right) \end{array}$$

Semantics of programs. Obtained by interleaving ($\circledast$):

$$[\![t_1 \parallel \ldots \parallel t_n]\!] = [\![t_1]\!]\emptyset \circledast \ldots \circledast [\![t_n]\!]\emptyset$$

Volatile semantics

Semantics of threads. Parametrized over $\rho : \mathcal{R} \to \mathbb{N}$.

$$\text{(Writes)} \quad [\![x \;:=\; e; t]\!]\rho = \mathtt{W}_x^{(\rho(e))} \cdot [\![t]\!]\rho$$

$$\text{(Reads)} \quad [\![r \;\leftarrow\; x; t]\!]\rho = \bigcup_{i \in \mathbb{N}} \left(\mathtt{R}_x^{(i)} \cdot [\![t]\!](\rho[r \leftarrow i]) \right)$$

Semantics of programs. Obtained by interleaving ($\circledast$):

$$[\![t_1 \parallel \ldots \parallel t_n]\!] = [\![t_1]\!]\emptyset \circledast \ldots \circledast [\![t_n]\!]\emptyset$$

Example. Define $p = (x \;:=\; 1; y \;\leftarrow\; r \parallel y \;:=\; 1; x \;\leftarrow\; s)$

- $\mathtt{W}_x^{(1)} \cdot \mathtt{W}_y^{(1)} \cdot \mathtt{R}_y^{(3)} \cdot \mathtt{R}_x^{(2)} \in [\![p]\!]$
- but $\mathtt{R}_x^{(0)} \cdot \mathtt{R}_y^{(0)} \cdot \mathtt{W}_x^{(1)} \cdot \mathtt{W}_y^{(1)} \notin [\![p]\!]$.

Closed semantics

Obtained by eliminating "inconsistent" traces (eg. $\mathtt{W}_x^{(2)} \cdot \mathtt{R}_x^{(3)}$)

Linear memory model. A language of "consistent" traces:

$$\begin{aligned} M(\mu : \mathcal{V} \to \mathbb{N}) &::= \epsilon \\ &\quad \mid \mathtt{R}_x^{(\mu(x))} \cdot M(\mu) \\ &\quad \mid \mathtt{W}_x^{(k)} \cdot M(\mu[x \leftarrow k]) \\ M &::= M(x \mapsto 0) \end{aligned}$$

Closed semantics: $[\![p]\!] = [\![p]\!]^O \cap M$.

Example. Write $p = (x \ \mathtt{:=}\ 1; r \leftarrow y) \parallel (y \ \mathtt{:=}\ 2; s \leftarrow x)$

- every trace of $[\![p]\!]$ ends with $\mathtt{R}_x^{(1)}$ or a $\mathtt{R}_y^{(2)}$.

Summary

Advantages.

- Easy to define semantics, by induction on programs.
- By making M more complex, complex cache schemes can be handled

Drawbacks.

- Combinatorial explosion due to interleavings.
- How to model reordering of instructions?

Towards partial-orders.

- Because of reorderings, threads are not totally ordered
- Our goal: compute fine precisely dependencies between the instructions, given an architecture.

II. EVENT STRUCTURES

Raiders of the lost causality

Replacing traces by partial-orders

Idea: volatile semantics should be a set of partial-orders.

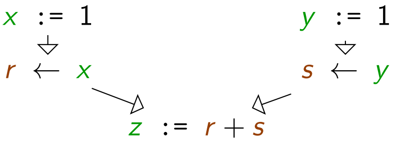
Term:

$$\begin{aligned} &x \;:=\; 1; y \;:=\; 1; \\ &r \;\leftarrow\; x; s \;\leftarrow\; y; \\ &z \;:=\; s + t \end{aligned}$$

Replacing traces by partial-orders

Idea: volatile semantics should be a set of partial-orders.

Dependencies (depends on the architecture):



$x := 1$

$r \leftarrow x$

$y := 1$

$s \leftarrow y$

$z := r + s$

Replacing traces by partial-orders

Idea: volatile semantics should be a set of partial-orders.

Executions (depends on the architecture):

$$\begin{tikzcd}
\mathtt{W}_{x}^{(1)} \arrow[d] & & \mathtt{W}_{y}^{(1)} \arrow[d] \\
\mathtt{R}_{x}^{(i)} \arrow[dr] & & \mathtt{R}_{y}^{(j)} \arrow[dl] \\
& \mathtt{W}_{z}^{(i+j)} &
\end{tikzcd}$$

for $i, j \in \mathbb{N}^2$.

- traces on Σ_c becomes *partially ordered multisets* over Σ_c (pomsets)
- $[\![t]\!]^O$ becomes a set of such *pomsets*.

Replacing traces by partial-orders

Idea: volatile semantics should be a set of partial-orders.

Executions (depends on the architecture):

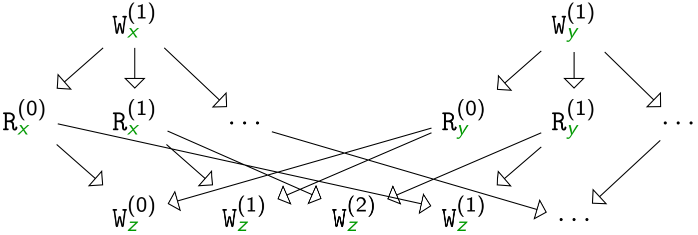
$$\begin{tikzcd}
\mathtt{W}_{x}^{(1)} \arrow[d] & & \mathtt{W}_{y}^{(1)} \arrow[d] \\
\mathtt{R}_{x}^{(i)} \arrow[dr] & & \mathtt{R}_{y}^{(j)} \arrow[dl] \\
& \mathtt{W}_{z}^{(i+j)} &
\end{tikzcd}$$

for $i, j \in \mathbb{N}^2$.

- traces on Σ_c becomes *partially ordered multisets* over Σ_c (pomsets)
- $[\![t]\!]^O$ becomes a set of such *pomsets*.
- **Problem**: lots of redundancies in the pomsets..

Can we sum up *all* executions in a single object?

Can we glue the executions all together in a partial-order? For instance:

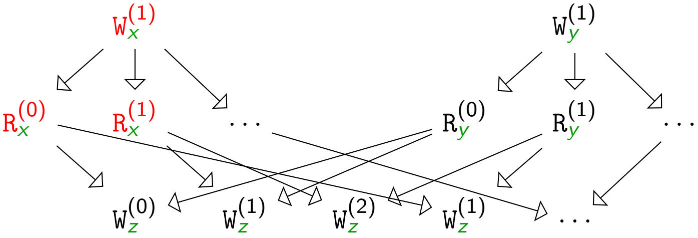


Which sets of events w are (partial) executions?

- w must be downward-closed for $\rightarrowtriangle$

Can we sum up *all* executions in a single object?

Can we glue the executions all together in a partial-order? For instance:

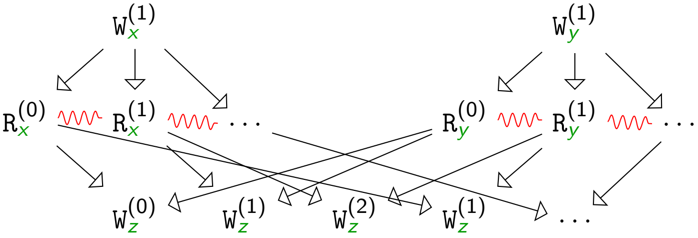


Which sets of events *w* are (partial) executions?

- *w* must be downward-closed for $\rightarrowtriangle$
- and ...? $\{\mathtt{W}_x^{(1)}, \mathtt{R}_x^{(0)}, \mathtt{R}_x^{(1)}\}$ cannot be a valid execution.

Can we sum up *all* executions in a single object?

Can we glue the executions all together in a partial-order? For instance:



Which sets of events *w* are (partial) executions?

- *w* must be downward-closed for $\rightarrowtriangle$
- and ...? $\{\mathtt{W}_x^{(1)}, \mathtt{R}_x^{(0)}, \mathtt{R}_x^{(1)}\}$ cannot be a valid execution.

$\Rightarrow$ Need more structure than a partial-order: **conflicts**.

Event structures save the day

Definition (Event structures)

A set of event E with:

- A notion of **causality** represented by *a partial order* $\leq_E$
- A notion of **conflict** represented by *a relation* $\sim_E$
- A labelling $l : E \to \Sigma$.

(+ axioms)

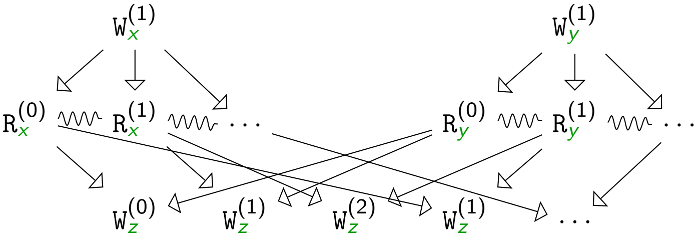
Definition (Configuration or partial execution)

A **configuration** of E is a subset w of E:

- downward-closed: $e \leq e' \in w \Rightarrow e \in w$.
- that does not contain two conflicting events

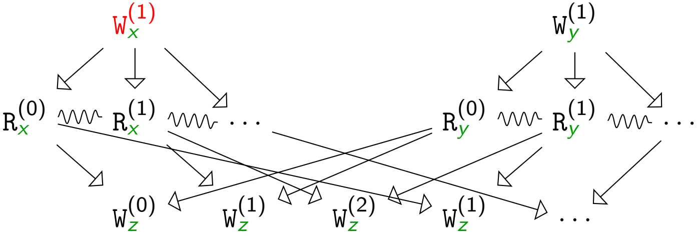
Event structures save the day

On the example:



Event structures save the day

On the example:

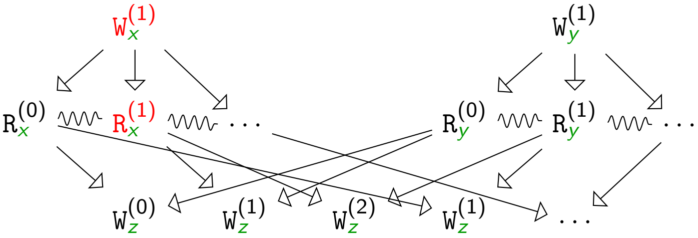


We have the configuration:

$$W_x^{(1)}$$

Event structures save the day

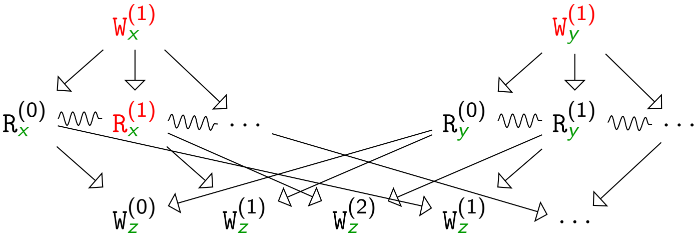
On the example:



We have the configuration:

Event structures save the day

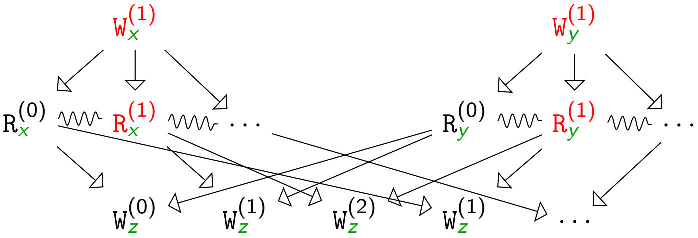
On the example:



We have the configuration:

Event structures save the day

On the example:

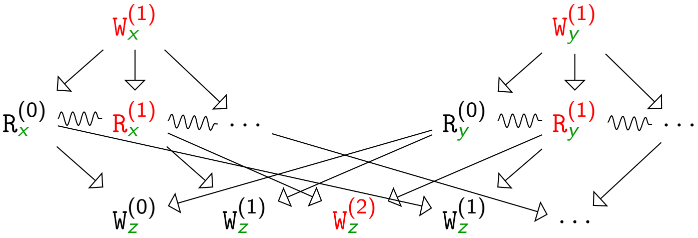


We have the configuration:

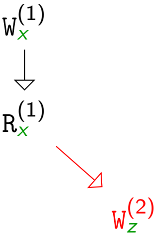
$$\begin{tikzcd}
W_x^{(1)} \arrow[d] & W_y^{(1)} \arrow[d] \\
R_x^{(1)} & R_y^{(1)}
\end{tikzcd}$$

Event structures save the day

On the example:



We have the configuration:



III. DESIGNING A SEMANTICS WITH EVENT STRUCTURES

Dessine-moi une structure d'événements

Defining architectures

Now we define an architecture $\mathscr{A}$ as a pair $(\rightharpoonup_{\mathscr{A}}, E)$:

- $\rightharpoonup_{\mathscr{A}} \subseteq \Sigma_a \times \Sigma_a$ indicates which causality cannot be erased.
- $E_{\mathscr{A}}$ is an event structure representing the memory model.

Examples for $\rightharpoonup_{\mathscr{A}}$:

- $\rightharpoonup_{\mathtt{SC}} = \Sigma_a \times \Sigma_a$
- $\rightharpoonup_{\mathtt{ARM}} = \{(e, e') \mid \mathtt{v}(e) = \mathtt{v}(e')\}$ $(\mathtt{v}(x, \_) = x)$.
- $\rightharpoonup_{\mathtt{x86}} = \dots$

Examples for $E_{\mathscr{A}}$ include all languages $M \subseteq \Sigma_c^*$ (they can be viewed as event structures).

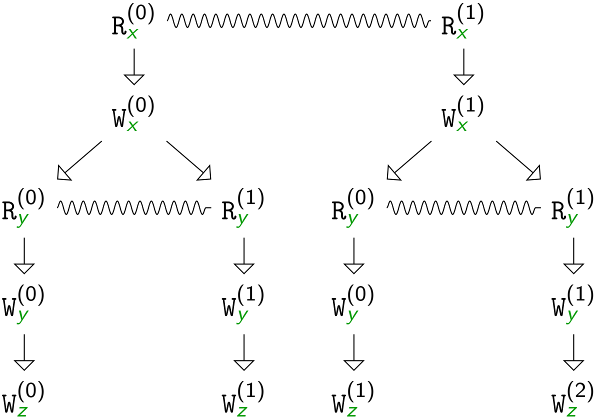
Computing the semantics $[\![p]\!]_{\mathscr{A}}$

As previously, in two steps:

- **Volatile semantics**:
  - *threads*: $[\![t]\!]^{O}_{\mathscr{A}}$ is defined as previously but where the causality outside $\rightarrowtriangle_{\mathscr{A}}$ are relaxed.
  - *programs*: $[\![t_1 \parallel \ldots \parallel t_n]\!]^{O}_{\mathscr{A}} = [\![t_1]\!]^{O}_{\mathscr{A}} \parallel \ldots \parallel [\![t_n]\!]^{O}_{\mathscr{A}}$ where $\parallel$ is *parallel composition*.
- **Closed semantics**: $[\![p]\!]_{\mathscr{A}} = [\![p]\!]^{O}_{\mathscr{A}} \wedge E_{\mathscr{A}}$ where $\wedge$ is the *synchronized product*: a generalization of intersection of languages to event structures.

Volatile semantics

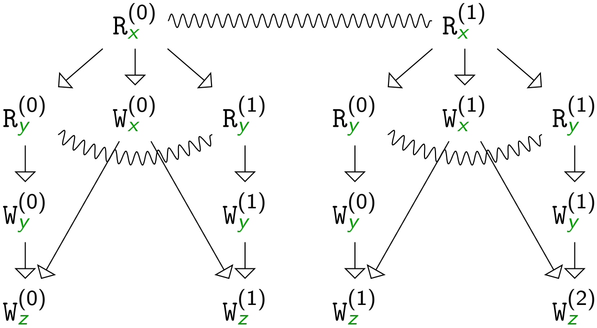
Pour $t = \begin{pmatrix} s & \leftarrow & x; x & := & s; \\ t & \leftarrow & y; y & := & t; \\ z & := & s + t & & \end{pmatrix}$, on a:



(SC)

Volatile semantics

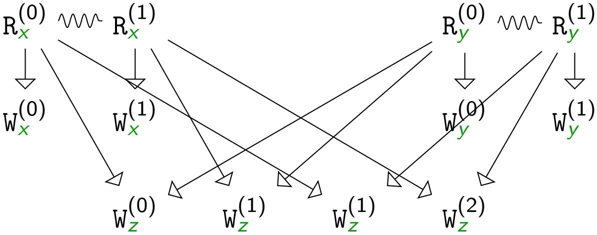
Pour $t = \begin{pmatrix} s \leftarrow x; x := s; \\ t \leftarrow y; y := t; \\ z := s + t \end{pmatrix}$, on a:



(x86)

Volatile semantics

Pour $t = \begin{pmatrix} s & \leftarrow & x; x & := & s; \\ t & \leftarrow & y; y & := & t; \\ z & := & s + t & & \end{pmatrix}$, on a:



(ARM)

Modelling caches

Goal: Model write caches.

Define an event structure $\mathscr{E}$ whose configurations are all partial-orders $(q, \leq_q)$ such that:

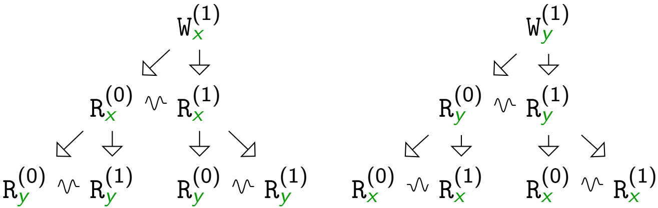
1. For all x, q restricted to writes on x is a total order.
2. For all $(\mathtt{R}_x^{(k)}, \_) \in q$, the set

$$\{\mathtt{W}_x^{(l)} \mid \mathtt{W}_x^{(l)} \in q\}$$

   has a maximum for $\leq_q$ and its value has the shape $\mathtt{W}_x^{(k)}$.

Example

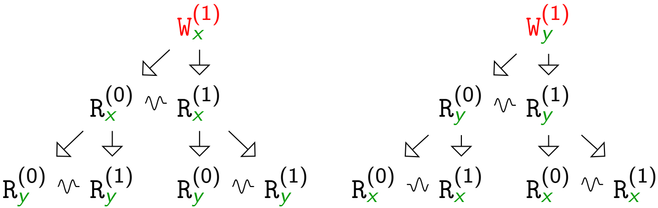
$$p = \begin{array}{ccc} x & := & 1 \\ r_1 & \leftarrow & x \\ r_2 & \leftarrow & y \end{array} \left\| \begin{array}{ccc} y & := & 1 \\ s_1 & \leftarrow & y \\ s_2 & \leftarrow & x \end{array} \right.$$



(**Volatile semantics** for SC)

Example

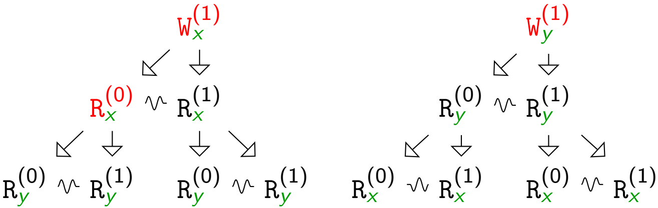
$$p = \begin{array}{ccc||ccc} x & := & 1 & y & := & 1 \\ r_1 & \leftarrow & x & s_1 & \leftarrow & y \\ r_2 & \leftarrow & y & s_2 & \leftarrow & x \end{array}$$



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

Example

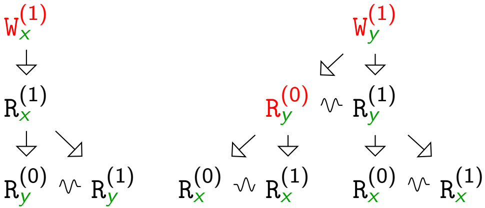
$$p = \begin{array}{rcl} x & := & 1 \\ r_1 & \leftarrow & x \\ r_2 & \leftarrow & y \end{array} \;\Bigg\|\; \begin{array}{rcl} y & := & 1 \\ s_1 & \leftarrow & y \\ s_2 & \leftarrow & x \end{array}$$



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

Example

$$p = \begin{array}{rcl} x & := & 1 \\ r_1 & \leftarrow & x \\ r_2 & \leftarrow & y \end{array} \Bigg\| \begin{array}{rcl} y & := & 1 \\ s_1 & \leftarrow & y \\ s_2 & \leftarrow & x \end{array}$$



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

Example

$$p = \begin{array}{rcl|rcl} x & := & 1 & y & := & 1 \\ r_1 & \leftarrow & x & s_1 & \leftarrow & y \\ r_2 & \leftarrow & y & s_2 & \leftarrow & x \end{array}$$

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

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

Example

$$p = \begin{array}{ccc|ccc} x & := & 1 & y & := & 1 \\ r_1 & \leftarrow & x & s_1 & \leftarrow & y \\ r_2 & \leftarrow & y & s_2 & \leftarrow & x \end{array}$$

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

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

Example

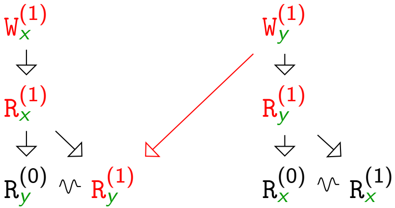
$$p = \begin{array}{rcl|rcl} x & := & 1 & y & := & 1 \\ r_1 & \leftarrow & x & s_1 & \leftarrow & y \\ r_2 & \leftarrow & y & s_2 & \leftarrow & x \end{array}$$

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

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

Example

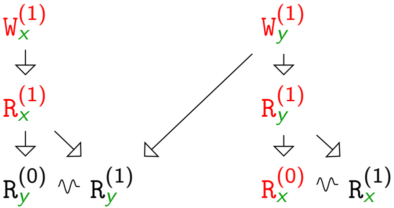
$$p = \begin{array}{rcl} x & := & 1 \\ r_1 & \leftarrow & x \\ r_2 & \leftarrow & y \end{array} \;\Bigg\|\; \begin{array}{rcl} y & := & 1 \\ s_1 & \leftarrow & y \\ s_2 & \leftarrow & x \end{array}$$



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

Example

$$p = \begin{array}{ccc} x & := & 1 \\ r_1 & \leftarrow & x \\ r_2 & \leftarrow & y \end{array} \Bigg\| \begin{array}{ccc} y & := & 1 \\ s_1 & \leftarrow & y \\ s_2 & \leftarrow & x \end{array}$$

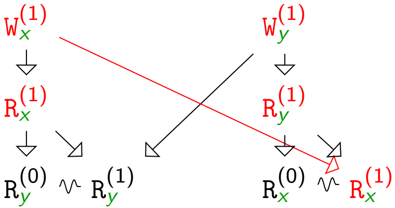


$$\begin{tikzcd}
\mathtt{W}_x^{(1)} \arrow[d] & & & \mathtt{W}_y^{(1)} \arrow[d] \arrow[ddll] & \\
\mathtt{R}_x^{(1)} \arrow[d] \arrow[dr] & & & \mathtt{R}_y^{(1)} \arrow[d] \arrow[dr] & \\
\mathtt{R}_y^{(0)} \arrow[r, phantom, "\sim"] & \mathtt{R}_y^{(1)} & & \mathtt{R}_x^{(0)} \arrow[r, phantom, "\sim"]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

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

Example

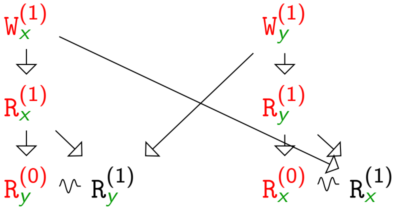
$$p = \begin{array}{rcl} x & := & 1 \\ r_1 & \leftarrow & x \\ r_2 & \leftarrow & y \end{array} \Bigg\| \begin{array}{rcl} y & := & 1 \\ s_1 & \leftarrow & y \\ s_2 & \leftarrow & x \end{array}$$



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

Example

$$p = \begin{array}{lll||lll} x & := & 1 & y & := & 1 \\ r_1 & \leftarrow & x & s_1 & \leftarrow & y \\ r_2 & \leftarrow & y & s_2 & \leftarrow & x \end{array}$$



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

We can see that we can observe $r_1 = s_1 = 1 \wedge r_2 = s_2 = 0$.

Conclusion

Extensions.

- We can define sub-models of $\mathscr{E}$ corresponding to actual architectures.
- The model is inspired from a game semantics model and simplified in this first-order setting.

To go further.

- Look at barriers
- Compare that with axiomatic semantics (compare the executions)

Enrichir les labels

Considérons: $t = \begin{pmatrix} r & \leftarrow & x; \\ x & := & 1; \\ x & := & r. \end{pmatrix}.$

Sans réordonancement:

$$\begin{tikzcd}
\mathtt{R}_x^{(0)} \arrow[r, squiggly, no head] \arrow[d] & \mathtt{R}_x^{(1)} \arrow[d] \\
\mathtt{W}_x^{(1)} \arrow[d] & \mathtt{W}_x^{(1)} \arrow[d] \\
\mathtt{W}_x^{(0)} & \mathtt{W}_x^{(1)}
\end{tikzcd}$$

On a $\mathtt{W}_x^{(1)} \rightharpoonup \mathtt{W}_x^{(0)}$ et $\mathtt{W}_x^{(1)} \rightharpoonup \mathtt{W}_x^{(1)}$ selon le passé.

Enrichir les labels

Considérons: $t = \begin{pmatrix} r & \leftarrow & x; \\ x & := & 1; \\ x & := & r. \end{pmatrix}$.

Sans réordonancement:

$$\begin{tikzcd}
\mathtt{R}_x^{(0),r:0} \arrow[r, no head, squiggly] \arrow[d, -{Triangle[open]}] & \mathtt{R}_x^{(1),r:1} \arrow[d, -{Triangle[open]}] \\
\mathtt{W}_x^{(1),r:0} \arrow[d, -{Triangle[open]}] & \mathtt{W}_x^{(1),r:1} \arrow[d, -{Triangle[open]}] \\
\mathtt{W}_x^{(0),r:0} & \mathtt{W}_x^{(1),r:1}
\end{tikzcd}$$

On a $\mathtt{W}_x^{(1)} \rightarrowtriangle \mathtt{W}_x^{(0)}$ et $\mathtt{W}_x^{(1)} \rightarrowtriangle \mathtt{W}_x^{(1)}$ selon le passé.
Pour lever l'ambiguïté, on définit

$$\bar{\Sigma} = \Sigma \times (\mathcal{R} \rightharpoonup \mathbb{N}).$$

Dépendances entre labels

On peut définir maintenant $\xrightarrow{\text{dep}} \subseteq \bar{\Sigma} \times \bar{\Sigma}$ selon l'architecture:

- x86 (permute écriture/lecture sur des zones différentes):

$$\begin{aligned}\xrightarrow{\text{dep}}_{\texttt{x86}} = \{(e, \rho), (e', \rho') \mid \\ &\rho \subseteq \rho' \\ &\wedge (\texttt{var}(e) = \texttt{var}(e') \vee \texttt{type}(\{e, e'\}) \neq \{\texttt{R}, \texttt{W}\})\}\end{aligned}$$

- ARM (permute les instructions sur des zones différentes):

$$\begin{aligned}\xrightarrow{\text{dep}}_{\texttt{ARM}} = \{(e, \rho), (e', \rho') \mid \\ &\rho \subseteq \rho' \\ &\wedge (\texttt{var}(e) = \texttt{var}(e'))\}\end{aligned}$$

avec $\texttt{type} : \Sigma \to \{\texttt{R}, \texttt{W}\}, \texttt{var} : \Sigma \to \mathcal{V}$.

Opérations sur les structures d'évènements

[noframenumbering]

- **Somme**. Pour $(l_x \in \bar{\Sigma})_{x \in X}$, on forme $\sum_{x \in X} u_x$:
  - évènements: X
  - causalité: $\leq$ est l'égalité
  - conflict: deux évènements distincts sont en conflits:

$$\sum_{k \in \mathbb{N}} \mathtt{R}_x^{(k)} = \mathtt{R}_x^{(0)} \sim \mathtt{R}_x^{(1)} \sim \dots .$$

- **Mise en parallèle**. Soit E, F des structures d'évènements et $\rightarrowtriangle_d \subseteq E \times F$. On forme $E \parallel_{\rightarrowtriangle_d} F$:
  - évènements: union disjointe de E et F
  - causalité: plus petit ordre qui contient $\leq_E, \leq_F$ et $\rightarrowtriangle_d$
  - conflit: union de $\sim_E$ et $\sim_F$.

On note $E \parallel F$ pour $E \parallel_\emptyset F$.

Définition de la sémantique

Note: x := 1 dépend des lectures sur x précédentes.

Définition de la sémantique

Note: x `:=` 1 dépend des lectures sur x précédentes.
Notre sémantique est donc paramétrée par $\sigma : \mathcal{R} \rightharpoonup \mathcal{V}$.

1. Pour une instruction ι, D_ι: les registres dont ι dépend:

	Écriture	Lecture
x86 **ARM**	$D^\sigma_{x \texttt{ := } e} = \sigma^{-1}(x) \cup \mathrm{fv}(e)$	$D_{r \leftarrow x} = \{r\} \cup \mathrm{dom}(\sigma)$ $D_{r \leftarrow x} = \{r\} \cup \sigma^{-1}(x)$

Définition de la sémantique

Note: $x \ := \ 1$ dépend des lectures sur x précédentes.
Notre sémantique est donc paramétrée par $\sigma : \mathcal{R} \rightharpoonup \mathcal{V}$.

1. Pour une instruction ι, D_ι: les registres dont ι dépend:

	Écriture	Lecture
x86	$D^\sigma_{x \ := \ e} = \sigma^{-1}(x) \cup \mathsf{fv}(e)$	$D_{r \ \leftarrow \ x} = \{r\} \cup \mathsf{dom}(\sigma)$
ARM		$D_{r \ \leftarrow \ x} = \{r\} \cup \sigma^{-1}(x)$

2. **Sémantique des threads**:

$$[\![x \ := \ e; t]\!]\sigma = \left(\sum_{\rho : (D^\sigma_{x \ := \ e}) \to \mathbb{N}} \mathtt{W}_x^{(\rho(e)),\rho} \right) \parallel \xrightarrow{\textsf{dep}} [\![t]\!]\sigma$$

$$[\![r \ \leftarrow \ x; t]\!]\sigma = \left(\sum_{\rho : (D^\sigma_{r \ \leftarrow \ x}) \to \mathbb{N}} \mathtt{R}_x^{(\rho(r)),\rho} \right) \parallel \xrightarrow{\textsf{dep}} [\![t]\!](\sigma[r \leftarrow x])$$

Définition de la sémantique

Note: $x := 1$ dépend des lectures sur x précédentes.
Notre sémantique est donc paramétrée par $\sigma : \mathcal{R} \rightharpoonup \mathcal{V}$.

1. Pour une instruction ι, D_ι: les registres dont ι dépend:

	Écriture	Lecture
x86	$D^\sigma_{x := e} = \sigma^{-1}(x) \cup \mathrm{fv}(e)$	$D_{r \leftarrow x} = \{r\} \cup \mathrm{dom}(\sigma)$
ARM		$D_{r \leftarrow x} = \{r\} \cup \sigma^{-1}(x)$

2. **Sémantique des threads**:

$$[\![x := e; t]\!]\sigma = \left(\sum_{\rho : (D^\sigma_{x := e}) \to \mathbb{N}} \mathtt{W}_x^{(\rho(e)),\rho} \right) \parallel \xrightarrow{\mathbf{dep}} [\![t]\!]\sigma$$

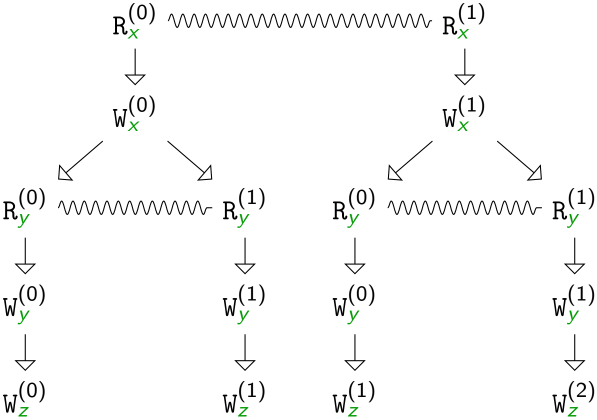
$$[\![r \leftarrow x; t]\!]\sigma = \left(\sum_{\rho : (D^\sigma_{r \leftarrow x}) \to \mathbb{N}} \mathtt{R}_x^{(\rho(r)),\rho} \right) \parallel \xrightarrow{\mathbf{dep}} [\![t]\!](\sigma[r \leftarrow x])$$

3. **Sémantique des programmes**:

$$[\![t_1 \parallel \ldots \parallel t_n]\!] = [\![t_1]\!]\emptyset \parallel \ldots \parallel [\![t_n]\!]\emptyset$$

Exemples

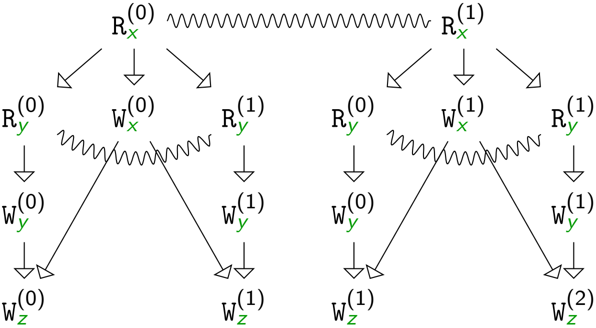
$$\text{Pour } t = \begin{pmatrix} s \leftarrow x; x := s; \\ t \leftarrow y; y := t; \\ z := s + t \end{pmatrix}, \text{ on a:}$$



(SC)

Exemples

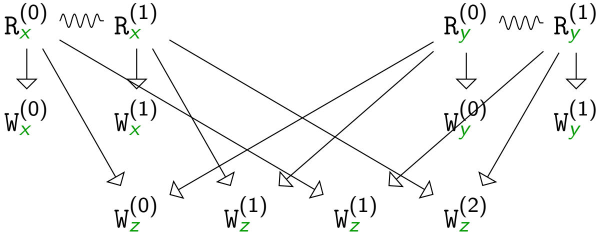
$$\text{Pour } t = \begin{pmatrix} s & \leftarrow & x; x & := & s; \\ t & \leftarrow & y; y & := & t; \\ z & := & s + t & & \end{pmatrix}, \text{ on a:}$$



(x86)

Exemples

$$\text{Pour } t = \begin{pmatrix} s \leftarrow x; x := s; \\ t \leftarrow y; y := t; \\ z := s + t \end{pmatrix}, \text{ on a:}$$



(ARM)

Et maintenant, la sémantique close

Pour calculer la sémantique close, on peut

1. Voir M comme une structure d'évènements E_M.
   (Ses configurations sont les traces de M)
2. Calculer le produit synchronisé: $[\![p]\!] = [\![p]\!]^O \wedge E_M$.
   (Opération combinatoire très compliquée)

Problème: l'ordre sur E_M est total donc $[\![p]\!]$ n'a pas de concurrence.

Solutions possibles:

1. Construire E_M comme $E_{M_x} \parallel E_{M_y} \parallel \ldots$: concurrence inter-variable.
2. Construire E_M comme une collection d'ordre partiels: concurrence intra-variable.

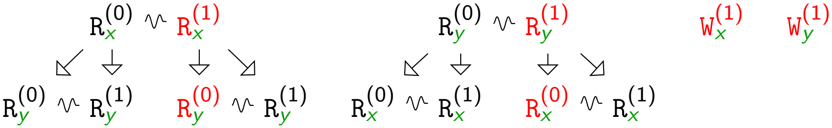
Un exemple

$$p = \begin{array}{l} r \leftarrow x \\ u \leftarrow y \end{array} \Bigg\| \begin{array}{l} s \leftarrow y \\ v \leftarrow x \end{array} \Bigg\| \; x := 1 \;\|\; y := 1$$

Un exemple

$$p = \begin{array}{l} r \leftarrow x \\ u \leftarrow y \end{array} \;\Bigg\|\; \begin{array}{l} s \leftarrow y \\ v \leftarrow x \end{array} \;\Bigg\|\; x := 1 \;\Big\|\; y := 1$$

Sémantique ouverte (sur x86):



Un exemple

$$p = \begin{array}{ccc} r & \leftarrow & x \\ u & \leftarrow & y \end{array} \Bigg\| \begin{array}{ccc} s & \leftarrow & y \\ v & \leftarrow & x \end{array} \Bigg\| \; x := 1 \; \| \; y := 1$$

Sémantique ouverte (sur x86):

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

Sémantique close. (Mémoire par traces)

$$\left.\begin{array}{l}
\mathtt{W}_x^{(1)} \to \mathtt{W}_y^{(1)} \to \mathtt{R}_x^{(1)} \to \mathtt{R}_y^{(1)} \to \mathtt{R}_y^{(0)} \to \mathtt{R}_x^{(0)} \\
\mathtt{W}_y^{(1)} \to \mathtt{W}_x^{(1)} \to \mathtt{R}_x^{(1)} \to \mathtt{R}_y^{(1)} \to \mathtt{R}_y^{(0)} \to \mathtt{R}_x^{(0)} \\
\mathtt{W}_x^{(1)} \to \mathtt{R}_x^{(1)} \to \mathtt{W}_y^{(1)} \to \mathtt{R}_y^{(1)} \to \mathtt{R}_y^{(0)} \to \mathtt{R}_x^{(0)} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \ldots
\end{array}\right. \quad \in \mathscr{C}(\llbracket p \rrbracket)$$

Un exemple

$$p = \begin{array}{ccc} r & \leftarrow & x \\ u & \leftarrow & y \end{array} \Bigg\| \begin{array}{ccc} s & \leftarrow & y \\ v & \leftarrow & x \end{array} \Bigg\| \; x \; := \; 1 \; \| \; y \; := \; 1$$

Sémantique ouverte (sur x86):

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

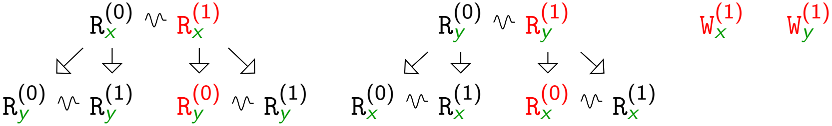
Sémantique close. (Mémoire avec concurrence inter-variable)

$$\begin{tikzcd} \mathtt{W}_x^{(1)} \arrow[r] & \mathtt{R}_x^{(1)} \arrow[r] \arrow[dr] & \mathtt{R}_y^{(0)} \\ \mathtt{W}_y^{(1)} \arrow[r] & \mathtt{R}_y^{(1)} \arrow[r] \arrow[ur] & \mathtt{R}_x^{(0)} \end{tikzcd} \qquad \in \mathscr{C}([\![p]\!])$$

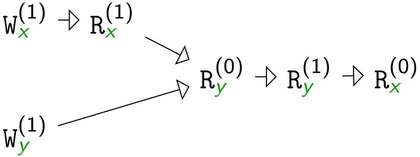
Un exemple

$$p = \begin{array}{l} r \leftarrow x \\ u \leftarrow y \end{array} \;\Bigg\|\; \begin{array}{l} s \leftarrow y \\ v \leftarrow x \end{array} \;\Bigg\|\; x := 1 \;\Big\|\; y := 1$$

Sémantique ouverte (sur x86):



Sémantique close. (Mémoire avec concurrence inter-variable)



$\in \mathscr{C}(\llbracket p \rrbracket)$

Un exemple

$$p = \begin{array}{l} r \leftarrow x \\ u \leftarrow y \end{array} \;\Bigg\|\; \begin{array}{l} s \leftarrow y \\ v \leftarrow x \end{array} \;\Bigg\|\; x := 1 \;\Big\|\; y := 1$$

Sémantique ouverte (sur x86):

$$\begin{tikzcd}
 & \mathtt{R}_{x}^{(0)} \arrow[dl] \arrow[d] \arrow[r, no head, squiggly] & {\color{red}\mathtt{R}_{x}^{(1)}} \arrow[d] \arrow[dr] & & & \mathtt{R}_{y}^{(0)} \arrow[dl] \arrow[d] \arrow[r, no head, squiggly] & {\color{red}\mathtt{R}_{y}^{(1)}} \arrow[d] \arrow[dr] & & {\color{red}\mathtt{W}_{x}^{(1)}} & {\color{red}\mathtt{W}_{y}^{(1)}} \\
\mathtt{R}_{y}^{(0)} \arrow[r, no head, squiggly] & \mathtt{R}_{y}^{(1)} & {\color{red}\mathtt{R}_{y}^{(0)}} \arrow[r, no head, squiggly] & \mathtt{R}_{y}^{(1)} & \mathtt{R}_{x}^{(0)} \arrow[r, no head, squiggly] & \mathtt{R}_{x}^{(1)} & {\color{red}\mathtt{R}_{x}^{(0)}} \arrow[r, no head, squiggly] & \mathtt{R}_{x}^{(1)} & &
\end{tikzcd}$$

Sémantique close. (Mémoire avec concurrence intra-variable)

$$\begin{array}{l} \mathtt{W}_{x}^{(1)} \rightarrow \mathtt{R}_{x}^{(1)} \rightarrow \mathtt{R}_{y}^{(0)} \\ \\ \mathtt{W}_{y}^{(1)} \rightarrow \mathtt{R}_{y}^{(1)} \rightarrow \mathtt{R}_{x}^{(0)} \end{array} \in \mathscr{C}(\llbracket p \rrbracket)$$

Un exemple de M non non séquentiel

On a la sémantique *volatile*, comment calculer la sémantique *close* ?
But: modéliser des écritures différées.

- Les labels indiquent le *thread* dont ils viennent: $\Sigma_{id} = \Sigma \times \mathbb{N}$
- À la mémoire globale $\mu : \mathcal{V} \rightharpoonup \mathbb{N}$, s'ajoute une mémoire locale $\lambda : \mathbb{N} \to (\mathcal{V} \rightharpoonup \mathbb{N})$.

Ensuite, on peut définir $M : (\mathbb{N} \to (\mathcal{V} \rightharpoonup \mathbb{N})) \to (\mathcal{V} \rightharpoonup \mathbb{N}) \to \Sigma_{id}^*$:

$$M_{\lambda,\rho} ::= \epsilon$$

Un exemple de M non non séquentiel

On a la sémantique *volatile*, comment calculer la sémantique *close* ?
But: modéliser des écritures différées.

- Les labels indiquent le *thread* dont ils viennent: $\Sigma_{id} = \Sigma \times \mathbb{N}$
- À la mémoire globale $\mu : \mathcal{V} \rightharpoonup \mathbb{N}$, s'ajoute une mémoire locale $\lambda : \mathbb{N} \to (\mathcal{V} \rightharpoonup \mathbb{N})$.

Ensuite, on peut définir $M : (\mathbb{N} \to (\mathcal{V} \rightharpoonup \mathbb{N})) \to (\mathcal{V} \rightharpoonup \mathbb{N}) \to \Sigma_{id}^*$:

$$
\begin{aligned}
M_{\lambda,\rho} ::= {} & \epsilon \\
& \mid (\mathtt{R}_x^{(\lambda(\iota)(x))}, \iota) \cdot M_{\lambda,\rho} \qquad \text{(Lecture cache)}
\end{aligned}
$$

Un exemple de M non non séquentiel

On a la sémantique *volatile*, comment calculer la sémantique *close* ?
But: modéliser des écritures différées.

- Les labels indiquent le *thread* dont ils viennent: $\Sigma_{id} = \Sigma \times \mathbb{N}$
- À la mémoire globale $\mu : \mathcal{V} \rightharpoonup \mathbb{N}$, s'ajoute une mémoire locale $\lambda : \mathbb{N} \to (\mathcal{V} \rightharpoonup \mathbb{N})$.

Ensuite, on peut définir $M : (\mathbb{N} \to (\mathcal{V} \rightharpoonup \mathbb{N})) \to (\mathcal{V} \rightharpoonup \mathbb{N}) \to \Sigma_{id}^*$:

$$\begin{aligned} M_{\lambda,\rho} ::= {} & \epsilon \\ & \mid (\mathtt{R}_x^{(\lambda(\iota)(x))}, \iota) \cdot M_{\lambda,\rho} && \text{(Lecture cache)} \\ & \mid (\mathtt{R}_x^{(\mu(x))}, \iota) \cdot M_{\lambda[(\iota,x)\leftarrow n],\mu} && \text{(Lecture globale)} \end{aligned}$$

Un exemple de M non non séquentiel

On a la sémantique *volatile*, comment calculer la sémantique *close* ?
But: modéliser des écritures différées.

- Les labels indiquent le *thread* dont ils viennent: $\Sigma_{id} = \Sigma \times \mathbb{N}$
- À la mémoire globale $\mu : \mathcal{V} \rightharpoonup \mathbb{N}$, s'ajoute une mémoire locale $\lambda : \mathbb{N} \to (\mathcal{V} \rightharpoonup \mathbb{N})$.

Ensuite, on peut définir $M : (\mathbb{N} \to (\mathcal{V} \rightharpoonup \mathbb{N})) \to (\mathcal{V} \rightharpoonup \mathbb{N}) \to \Sigma_{id}^{*}$:

$$\begin{aligned}
M_{\lambda,\rho} ::= {} & \epsilon \\
& \mid (\mathtt{R}_{x}^{(\lambda(\iota)(x))}, \iota) \cdot M_{\lambda,\rho} && \text{(Lecture cache)} \\
& \mid (\mathtt{R}_{x}^{(\mu(x))}, \iota) \cdot M_{\lambda[(\iota,x)\leftarrow n],\mu} && \text{(Lecture globale)} \\
& \mid (\mathtt{W}_{\iota}^{(n)}, \iota) \cdot M_{\lambda[(\iota,x)\leftarrow n],\mu[x\leftarrow n]} && \text{(Écriture)}
\end{aligned}$$

Un exemple de M non non séquentiel

On a la sémantique *volatile*, comment calculer la sémantique *close* ?
But: modéliser des écritures différées.

- Les labels indiquent le *thread* dont ils viennent: $\Sigma_{id} = \Sigma \times \mathbb{N}$
- À la mémoire globale $\mu : \mathcal{V} \rightharpoonup \mathbb{N}$, s'ajoute une mémoire locale $\lambda : \mathbb{N} \to (\mathcal{V} \rightharpoonup \mathbb{N})$.

Ensuite, on peut définir $M : (\mathbb{N} \to (\mathcal{V} \rightharpoonup \mathbb{N})) \to (\mathcal{V} \rightharpoonup \mathbb{N}) \to \Sigma_{id}^*$:

$$\begin{aligned}
M_{\lambda,\rho} ::= {} & \epsilon \\
& \mid (\mathtt{R}_x^{(\lambda(\iota)(x))}, \iota) \cdot M_{\lambda,\rho} && \text{(Lecture cache)} \\
& \mid (\mathtt{R}_x^{(\mu(x))}, \iota) \cdot M_{\lambda[(\iota,x)\leftarrow n],\mu} && \text{(Lecture globale)} \\
& \mid (\mathtt{W}_\iota^{(n)}, \iota) \cdot M_{\lambda[(\iota,x)\leftarrow n],\mu[x\leftarrow n]} && \text{(Écriture)} \\
M ::= {} & M(\iota \mapsto (x \mapsto 0))(x \mapsto 0)
\end{aligned}$$

Traces d'une structure d'évènements

Definition (Trace d'une configuration)

Une trace de $w \in \mathscr{C}(E)$ est une linéarisation de l'ordre partiel w.
L'ensemble des traces de w est dénoté $\mathrm{tr}(w)$.

Exemple: $\mathrm{tr}(\mathtt{R}_{x}^{(1)} \quad \mathtt{W}_{y}^{(2)}) = \{\mathtt{R}_{x}^{(1)} \cdot \mathtt{W}_{y}^{(2)}, \mathtt{W}_{y}^{(2)} \cdot \mathtt{R}_{x}^{(1)}\}$.

Traces d'une structure d'évènements

Definition (Trace d'une configuration)

Une trace de $w \in \mathscr{C}(E)$ est une linéarisation de l'ordre partiel w.
L'ensemble des traces de w est dénoté $\mathrm{tr}(w)$.
Exemple: $\mathrm{tr}(\mathtt{R}_x^{(1)} \quad \mathtt{W}_y^{(2)}) = \{\mathtt{R}_x^{(1)} \cdot \mathtt{W}_y^{(2)}, \mathtt{W}_y^{(2)} \cdot \mathtt{R}_x^{(1)}\}$.

On en déduit une nouvelle sémantique close par traces:

$$[\![p]\!] = \bigcup \{\mathrm{tr}(w) \cap M \mid w \in \mathscr{C}([\![p]\!]^O)\}$$

Programme	$(x \;:=\; 1; r \;\leftarrow\; y) \parallel (y \;:=\; 1; s \;\leftarrow\; x)$
Sémantique volatile	$\mathtt{W}_x^{(1)} \quad \mathtt{R}_y^{(0)} \sim \mathtt{R}_y^{(1)} \quad \mathtt{W}_y^{(1)} \quad \mathtt{R}_x^{(0)} \sim \mathtt{R}_y^{(2)}$
Éléments dans $[\![p]\!]$	$\mathtt{R}_x^{(0)} \cdot \mathtt{R}_y^{(0)} \cdot \mathtt{W}_x^{(1)} \cdot \mathtt{W}_y^{(1)}$ $\mathtt{R}_x^{(0)} \cdot \mathtt{W}_x^{(1)} \cdot \mathtt{R}_y^{(0)} \cdot \mathtt{W}_y^{(1)}$

Et avec une structure d'évènements

- On doit séquentialiser *toute l'exécution* → explosion
- Or, seulement l'histoire d'**une** variable doit l'être.

Et avec une structure d'évènements

- On doit séquentialiser *toute l'exécution* → explosion
- Or, seulement l'histoire d'**une** variable doit l'être.

- Tous nos modèles mémoires ont une forme particulière:

$$M = M_x \circledast M_y \circledast \dots \qquad (\text{avec } \mathtt{var}(M_x) = \{x\} \text{ pour } x \in \mathcal{V})$$

- Cela mène à une notion d'exécutions partiellement ordonnées: paires $(w, (t_x))$
  - $w \in \mathscr{C}(\llbracket p \rrbracket^O)$
  - pour x, $t_x \in M_x$ avec $\mathrm{tr}(w) \cap (t_x \circledast t_y \circledast \dots) \neq \emptyset$

- **Théorème**. Il existe une structure d'évènements dont les configurations sont les exécutions partiellement ordonnées.