Mémoires modèles faibles avec des structures d'évènements

Simon Castellan, 27 janvier 2016

JFLA 2016

Des comportements inattendus

Un simple programme concurrent et impératif :

$$
\begin{array}{c}
x, y \text{ initialisées à } 0 \\
x := 1 \;\|\; y := 2 \\
r \leftarrow y \;\|\; s \leftarrow x
\end{array}
$$

variable globale · registre local

Résultat attendu : $r \neq 0 \vee s \neq 0$.

Des comportements inattendus

Un simple programme concurrent et impératif :

x, y initialisées à 0

$$\begin{array}{l|l} r \leftarrow y & s \leftarrow x \\ x := 1 & y := 2 \end{array}$$

variable globale · registre local

Résultat attendu : $r \neq 0 \vee s \neq 0$.
Faux sur les architectures modernes (x86, ARM, . . .).

Des comportements inattendus

Un autre simple programme concurrent impératif :

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

Résultat attendu : $r_1 = s_1 = 1 \Rightarrow r_2 = s_2 = 1$

Des comportements inattendus

Un autre simple programme concurrent impératif :

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

Résultat attendu : $r_1 = s_1 = 1 \Rightarrow r_2 = s_2 = 1$

Faux même sur les architectures n'échangeant pas les lectures (x86).

Spécifier les architectures

Problème. Des exécutions qui ne sont plus des entrelacements.
$\rightarrow$ On parle de *modèles mémoires faibles*.

Il faut spécifier les exécutions sur ces architectures :

- quelles instructions peuvent être réordonnées ?
- comment se propagent les écritures d'un *thread* aux autres ?

Spécifier les architectures

Problème. Des exécutions qui ne sont plus des entrelacements.
$\rightarrow$ On parle de *modèles mémoires faibles*.

Il faut spécifier les exécutions sur ces architectures :

- quelles instructions peuvent être réordonnées ?
- comment se propagent les écritures d'un *thread* aux autres ?

Exemples d'architectures :

- `SC` (*Sequential consistency*) : pas de réordonancement, mémoire séquentielle
- `x86` : commutations lecture/écriture sur des variables différents, caches
- `ARM` : commutations de toutes instructions sur des variables différents, caches

Les constructeurs produisent des documents **en prose** :

- *ambiguïté* : des comportements ne sont pas spécifiés.
- *incohérence* : ne coïncide pas avec l'observation.

Sémantique à la rescousse

Formaliser les exécutions : comme des collections d'évènements mémoires *ordonnés*.

Programme	Exemple d'exécution
$x := 1 \;\Vert\; y := 2$ $r \leftarrow y \;\Vert\; s \leftarrow x$	$\mathtt{W}_x^{(1)} \cdot \mathtt{W}_y^{(2)} \cdot \mathtt{R}_y^{(2)} \cdot \mathtt{R}_x^{(1)}$

Deux principales écoles parmi les modèles existants :

- *sémantique opérationnelle* : les exécutions sont obtenues par une machine abstraite.
- *sémantique axiomatique* : les exécutions valides d'un programme sur une architecture est axiomatisée.

Cet exposé

Une sémantique

- **dénotationnelle** : calcul *par induction* des exécutions possibles
  - la sémantique est donc *compositionnelle*
- **compacte** : basée sur des structures d'évènements
  - pas d'explosion combinatoire
- **extensible** : inspirée de la sémantique des jeux
  - rajout facile de l'ordre supérieur, boucles, contrôle, ...

Plan de l'exposé :

1. Calculer les entrelacements
2. Retrouver la causalité
3. Une sémantique avec des structures d'évènements

I. Une sémantique dénotationnelle pour SC

Avec une trace d'originalité

Nos programmes

Structure très simple des programmes :

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

Dans la vraie vie : au moins `if`, et barrières.
Restriction. Une écriture au plus sur chaque registre ! ($\sim$SSA)

Sémantique dénotationnelle

Objectif : calculer $[\![t]\!] \in E$ où E est l'espace des dénotations.

Notre espace ici : langages de traces.

$$\Sigma = \mathtt{R}_x^{(k)} \mid \mathtt{W}_x^{(k)} \qquad (k \in \mathbb{N})$$
$$E = \mathscr{P}(\Sigma^*)$$

Deux étapes :

1. Sémantique volatile $[\![t]\!]^O$: les variables globales sont *volatiles* : $[\![x := 1; r \leftarrow x]\!]^O$ ne garantit pas lire 1 dans r.
2. Sémantique close : une fois qu'on a calculé $[\![t]\!]^O$, on restreint la portée des variables : $[\![x := 1; r \leftarrow x]\!]$ lit 1 dans r.

Sémantique volatile

Sémantique des *threads*. Paramétrée par $\rho : \mathcal{R} \to \mathbb{N}$.

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

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

Sémantique volatile

Sémantique des *threads*. Paramétrée par $\rho : \mathcal{R} \to \mathbb{N}$.

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

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

Sémantique des programmes. Obtenue par entrelacement ($\circledast$) :

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

Sémantique volatile

Sémantique des *threads*. Paramétrée par $\rho : \mathcal{R} \to \mathbb{N}$.

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

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

Sémantique des programmes. Obtenue par entrelacement ($\circledast$) :

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

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

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

Sémantique close

Définie en éliminant les traces «incohérentes ».
(Par ex. $\mathtt{W}_x^{(2)} \cdot \mathtt{R}_x^{(3)}$)

Modèle mémoire linéaire. Un langage de traces «cohérentes » :

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

Sémantique close : $[\![p]\!] = [\![p]\!]^O \cap M$.

Exemple. Notons $p = (x := 1; r \leftarrow y) \parallel (y := 2; s \leftarrow x)$

- toute trace de $[\![p]\!]$ contient en dernier un $\mathtt{R}_x^{(1)}$ ou un $\mathtt{R}_y^{(2)}$.

Bilan

Les plus.

- Sémantique facile à définir, par induction sur les programmes
- Peut gérer des mémoires avec cache (en complexifiant M)

Les moins.

- Explosion combinatoire à cause des entrelacements
- Comment autoriser le réordonancements d'instructions ?

Vers des ordres partiels.

- À cause des réordonancements, les *threads* ne sont plus totalement ordonnés.
- Notre objectif : calculer finement la dépendance entre instruction, étant donné une architecture.

II. Structures d'évènements

À la recherche de la causalité perdue

Remplacer les traces par des ordres partiels

Idée : la sémantique volatile devient un ensemble d'ordre partiels :

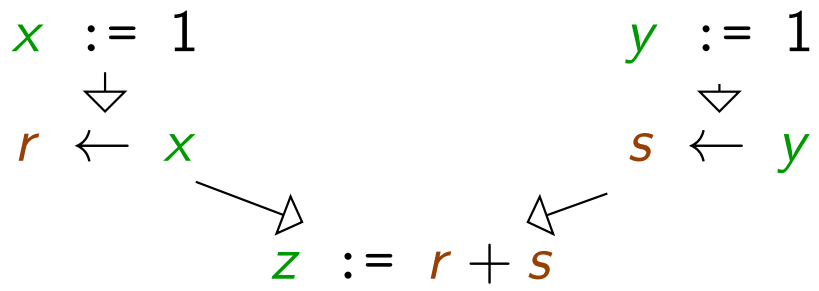
Terme :

$$x := 1; y := 1;$$
$$r \leftarrow x; s \leftarrow y;$$
$$z := s + t$$

Remplacer les traces par des ordres partiels

Idée : la sémantique volatile devient un ensemble d'ordre partiels :

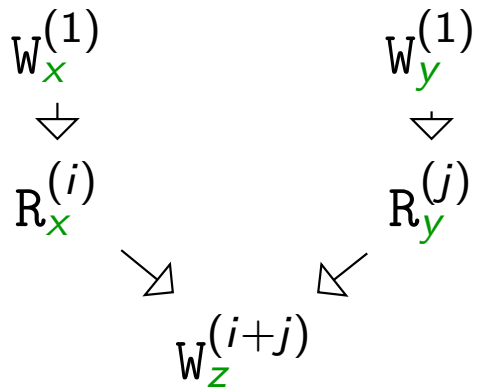
Dépendances (dépend de l'architecture visée) :



Remplacer les traces par des ordres partiels

Idée : la sémantique volatile devient un ensemble d'ordre partiels :

Exécutions (dépend de l'architecture visée) :



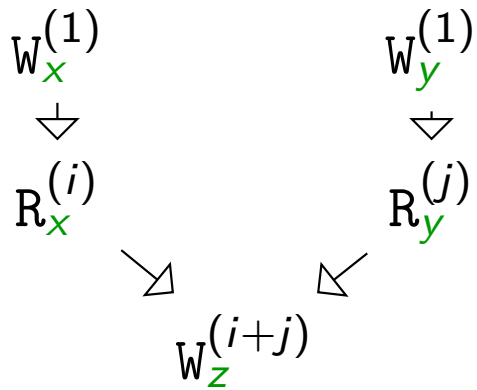
pour $i, j \in \mathbb{N}^2$.

- les traces sur Σ deviennent des *multi-ensembles partiellement ordonnés* sur Σ (*pomset*)
- $[\![t]\!]^O$ devient un ensemble de tels *pomsets*.

Remplacer les traces par des ordres partiels

Idée : la sémantique volatile devient un ensemble d'ordre partiels :

Exécutions (dépend de l'architecture visée) :

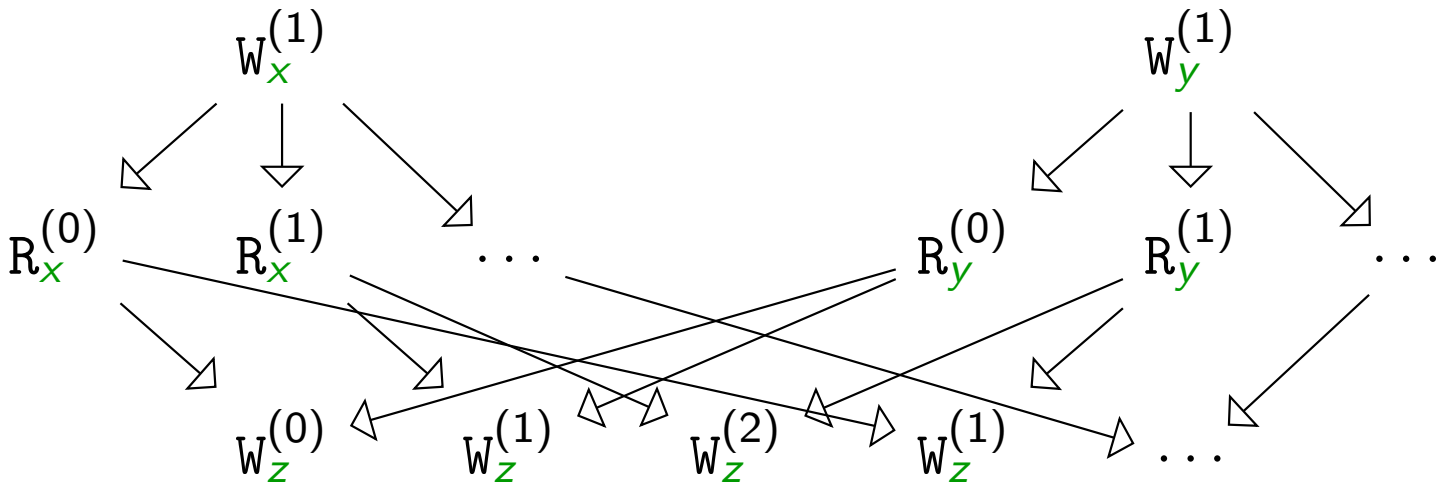


pour $i, j \in \mathbb{N}^2$.

- les traces sur Σ deviennent des *multi-ensembles partiellement ordonnés* sur Σ (*pomset*)
- $[\![t]\!]^O$ devient un ensemble de tels *pomsets*.
- **Problème** : beaucoup de redondance dans multi-ensembles ...

Peut-on résumer *toutes* les exécutions en un objet ?

Peut-on tout résumer dans un seul ordre partiel ? Par exemple :

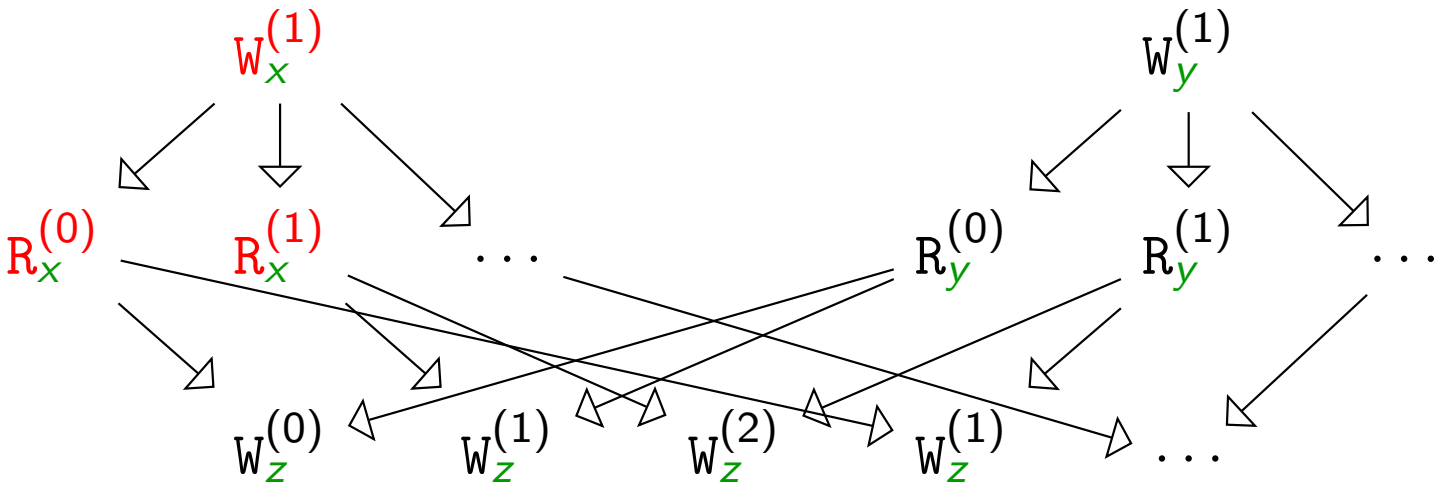


Quels ensembles d'évènements w sont des exécutions (partielles) ?

- w doit être clos vers le bas

Peut-on résumer *toutes* les exécutions en un objet ?

Peut-on tout résumer dans un seul ordre partiel ? Par exemple :

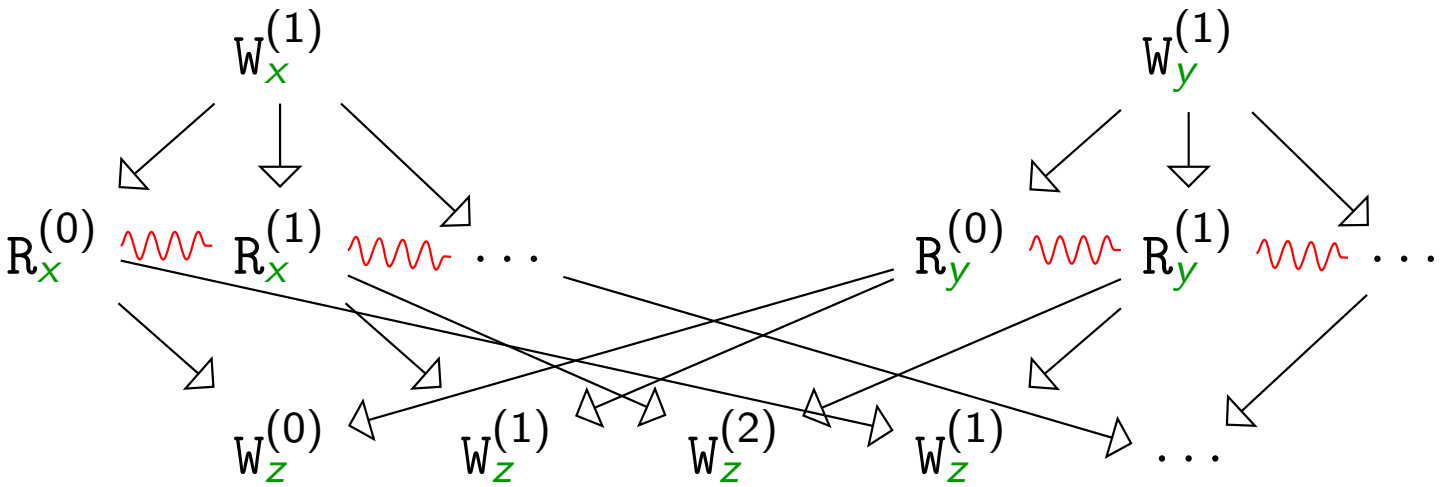


Quels ensembles d'évènements w sont des exécutions (partielles) ?

- w doit être clos vers le bas
- et ... ? $\{W_x^{(1)}, R_x^{(0)}, R_x^{(1)}\}$ ne peut être une exécution valide.

Peut-on résumer *toutes* les exécutions en un objet ?

Peut-on tout résumer dans un seul ordre partiel ? Par exemple :



Quels ensembles d'évènements w sont des exécutions (partielles) ?

- w doit être clos vers le bas
- et ... ? $\{W_x^{(1)}, R_x^{(0)}, R_x^{(1)}\}$ ne peut être une exécution valide.

$\Rightarrow$ Besoin de plus de structure qu'un ordre partiel : conflits

Event structures save the day

Definition (Structure d'évènements.)

Un ensemble d'évènement E avec :

- Une notion de **causalité** représentée par *un ordre partiel* $\leq_E$
- Une notion de **conflit** représentée par *une relation* $\sim_E$
- Un étiquetage $l : E \to \Sigma$.

(+ axiomes)

Definition (Configuration ou exécution partielle)

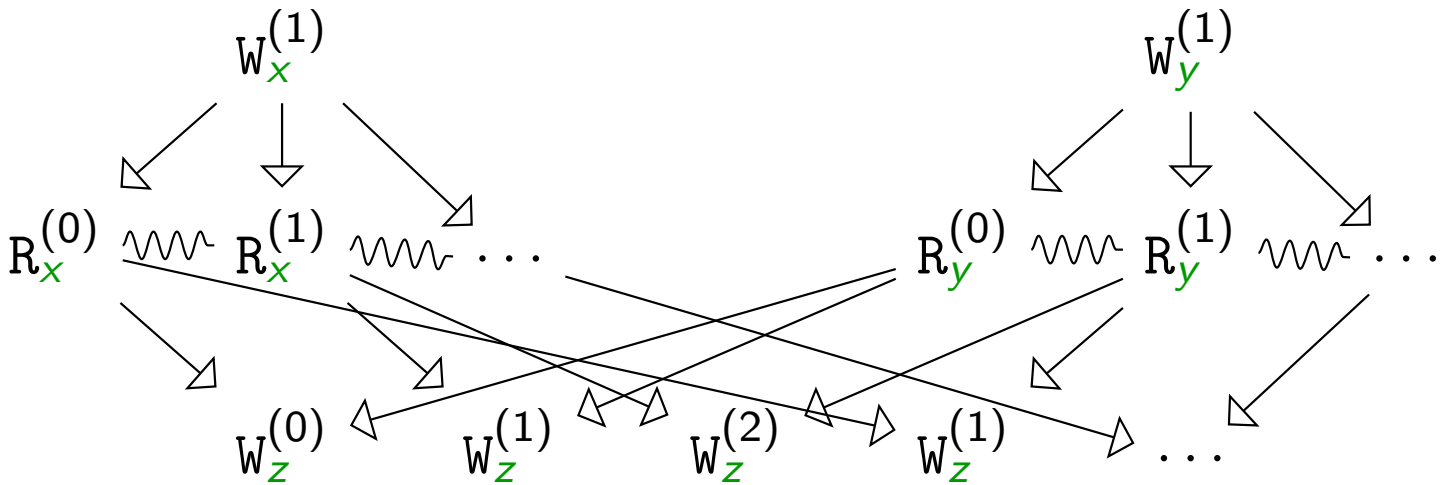
Une **configuration** de E est un sous-ensemble w de E :

- clos vers le bas : $e \leq e' \in w \Rightarrow e \in w$.
- qui ne contient pas deux évènements en conflit.

$\mathscr{C}(E)$ désigne l'ensemble des configurations de E.

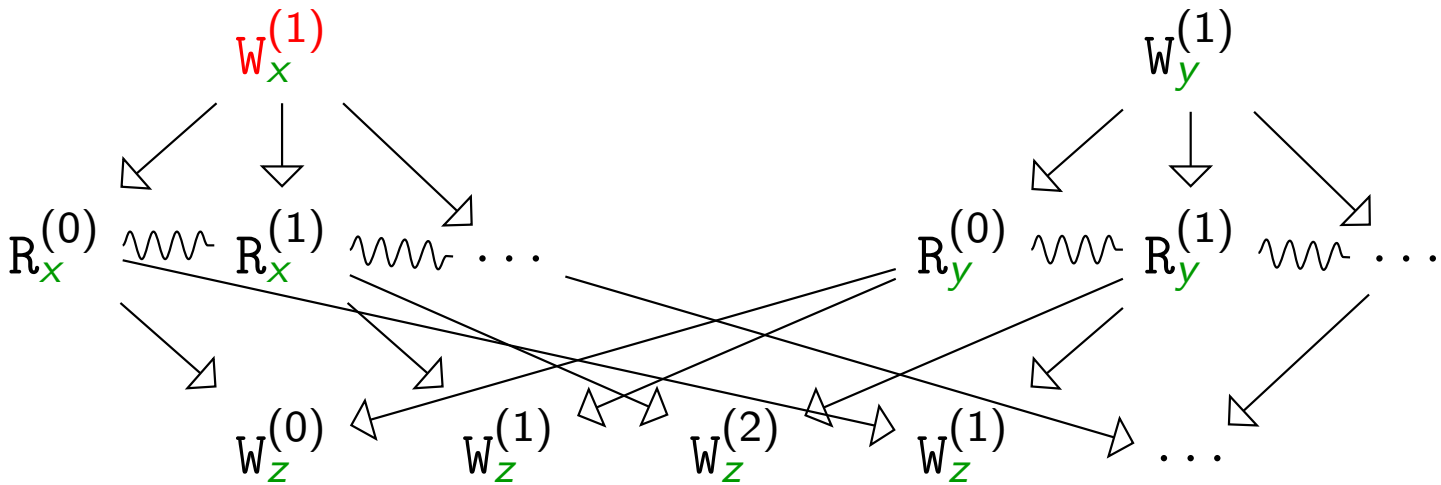
Event structures save the day

Sur l'exemple :



Event structures save the day

Sur l'exemple :

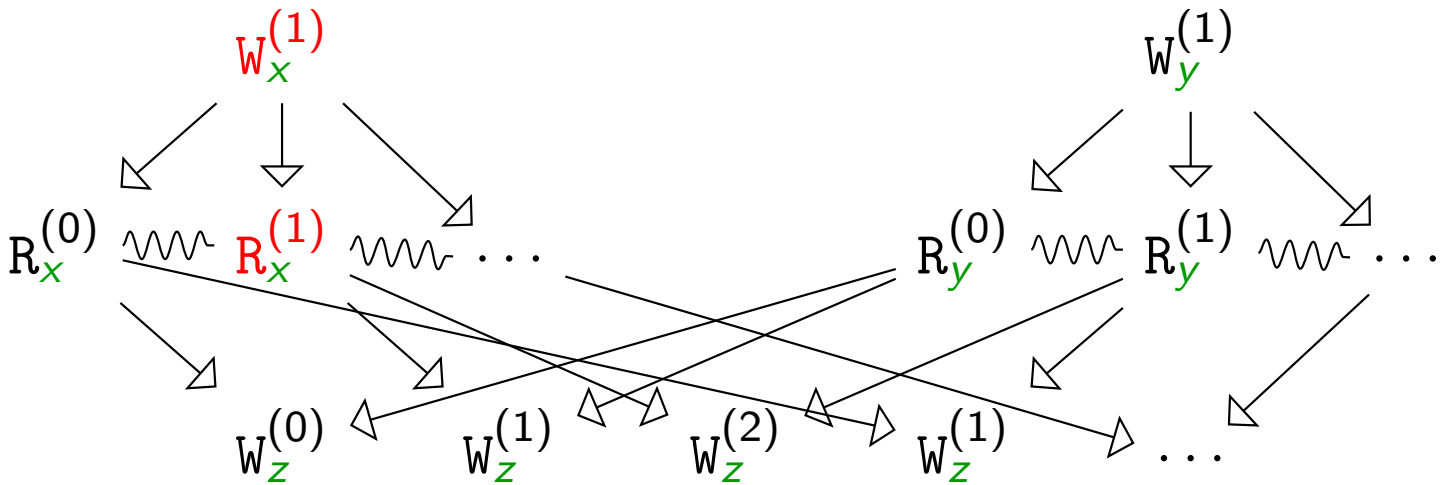


On a la configuration :

$W_x^{(1)}$

Event structures save the day

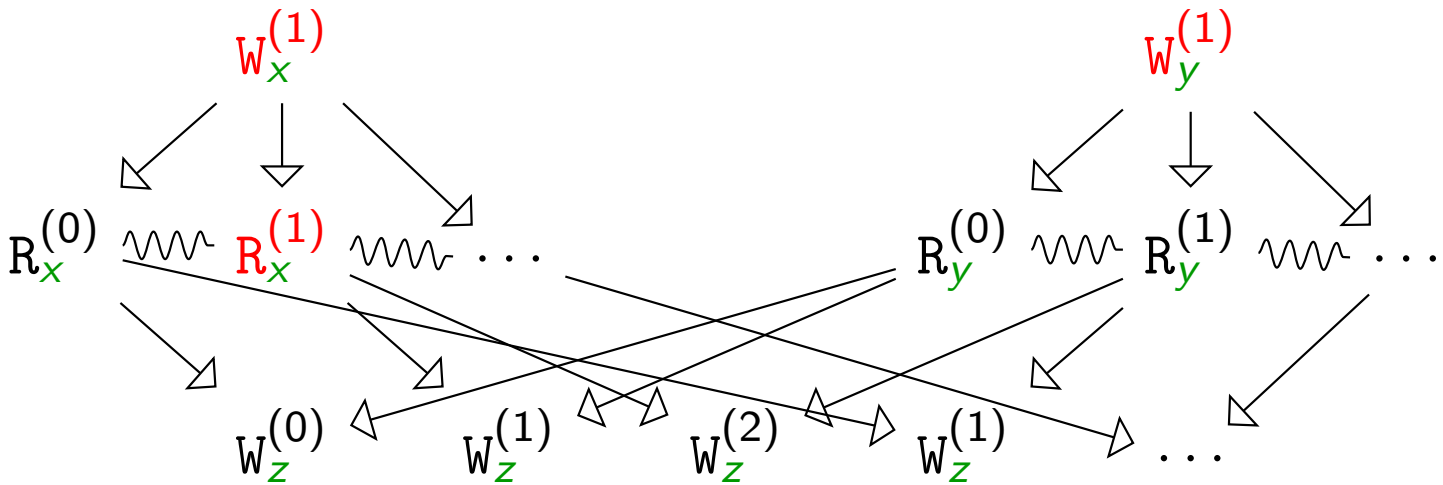
Sur l'exemple :



On a la configuration :

Event structures save the day

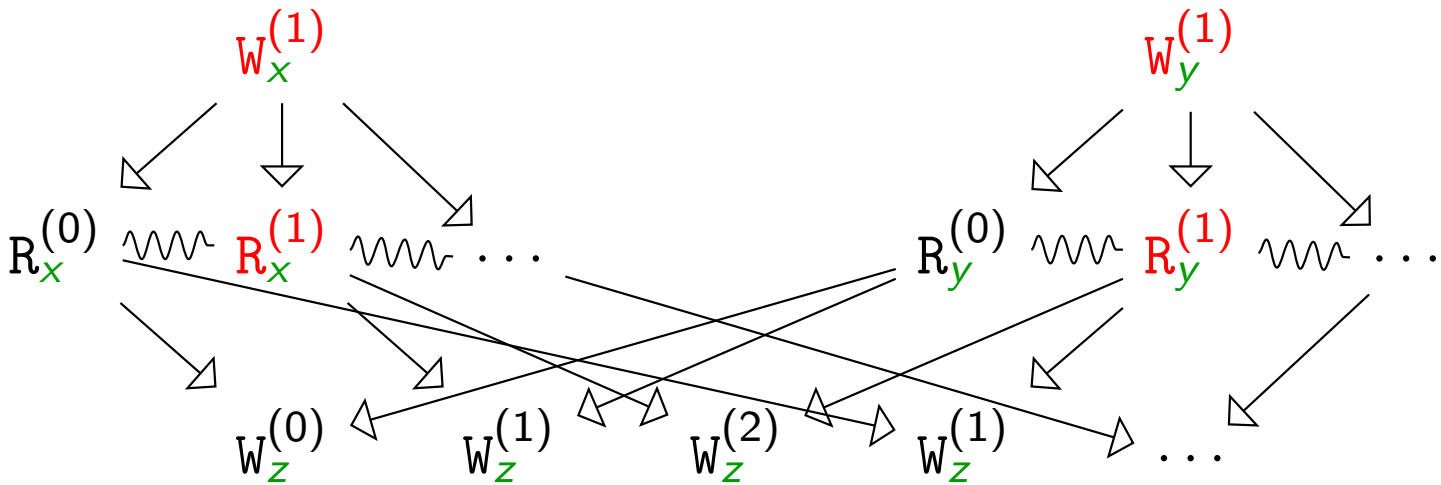
Sur l'exemple :



On a la configuration :

Event structures save the day

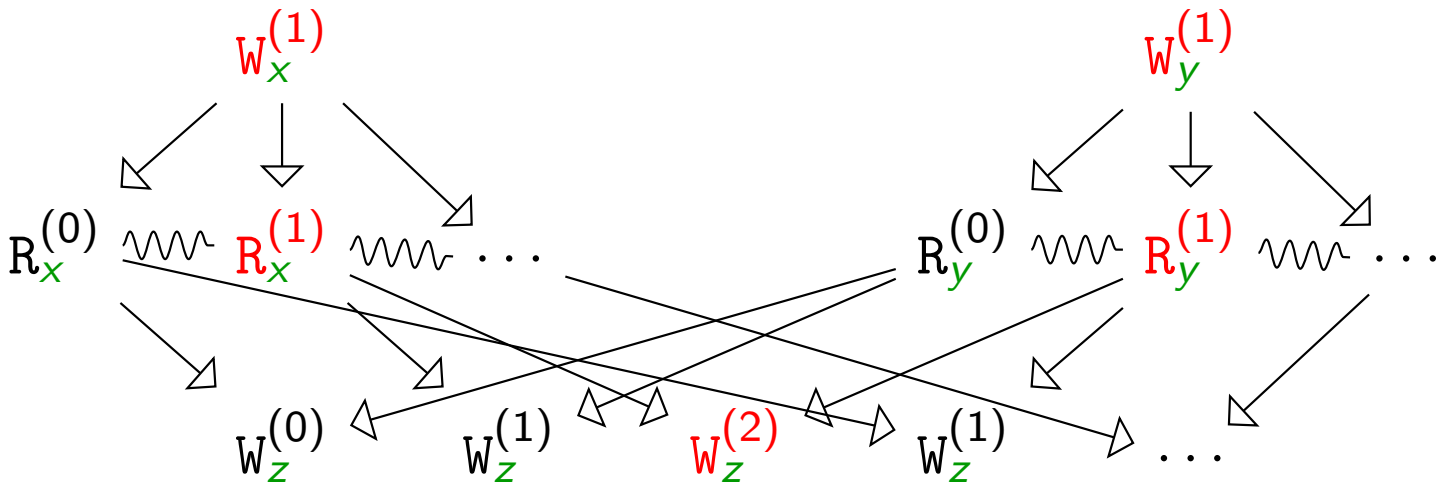
Sur l'exemple :



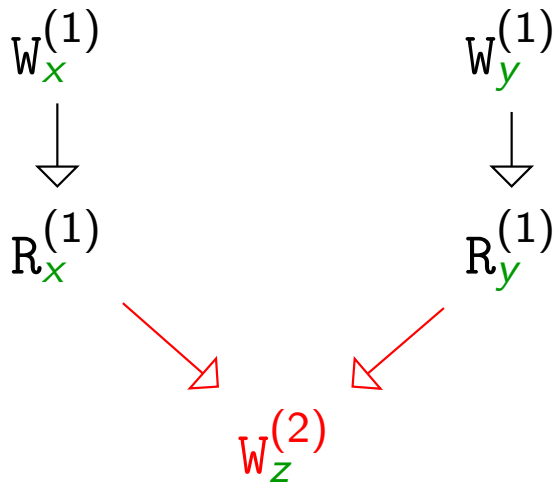
On a la configuration :

Event structures save the day

Sur l'exemple :



On a la configuration :



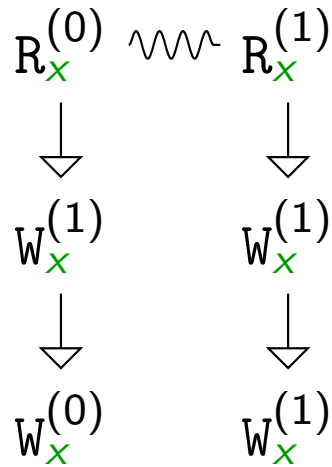
III. Une sémantique avec des des structures d'événements

Un programme déguisé pour calculer ce modèle

Enrichir les labels

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

Sans réordonancement :

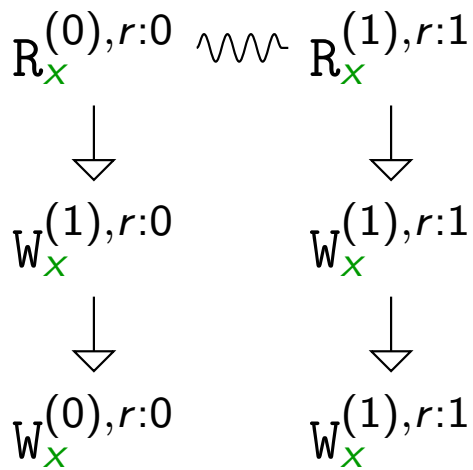


On a $\mathtt{W}_x^{(1)} \rightarrowtriangle \mathtt{W}_x^{(0)}$ et $\mathtt{W}_x^{(1)} \rightarrowtriangle \mathtt{W}_x^{(1)}$ selon le passé.

Enrichir les labels

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

Sans réordonancement :



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) :

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

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

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

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

Opérations sur les structures d'évènements

- **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 :

$$\sum_{k \in \mathbb{N}} \mathrm{R}_x^{(k)} = \mathrm{R}_x^{(0)} \quad \sim \mathrm{R}_x^{(1)} \quad \sim \ldots.$$

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

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** :

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

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

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** :

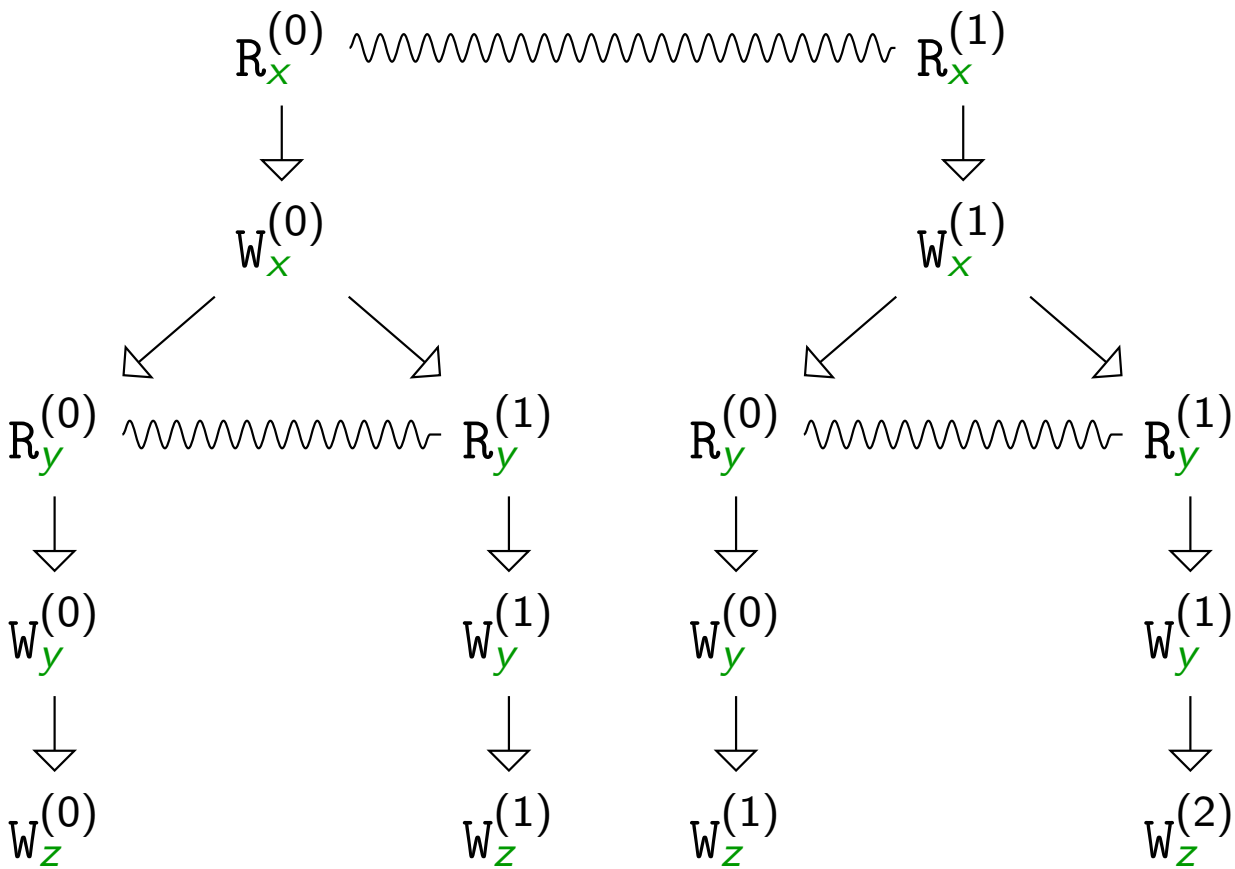
$$[\![x := e; t]\!]\sigma = \left(\sum_{\rho:(D^\sigma_{x := e}) \to \mathbb{N}} \mathtt{W}_x^{(\rho(e)),\rho} \right) \parallel_{\xrightarrow{\mathbf{dep}}} [\![t]\!]\sigma$$

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

3. **Sémantique des programmes** :
$[\![t_1 \parallel \ldots \parallel t_n]\!] = [\![t_1]\!]\emptyset \parallel \ldots \parallel [\![t_n]\!]\emptyset$

Exemples

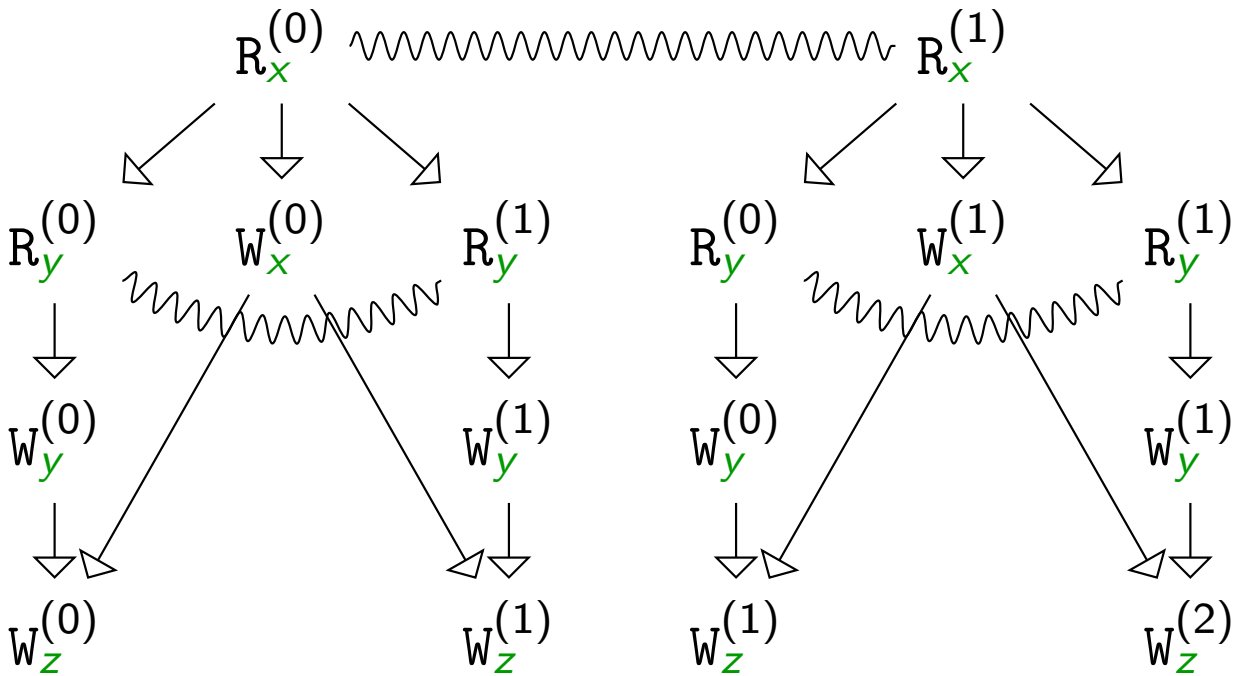
Pour $t = \begin{pmatrix} s \leftarrow x; x := s; \\ t \leftarrow y; y := t; \\ z := s + t \end{pmatrix}$, on a :



(SC)

Exemples

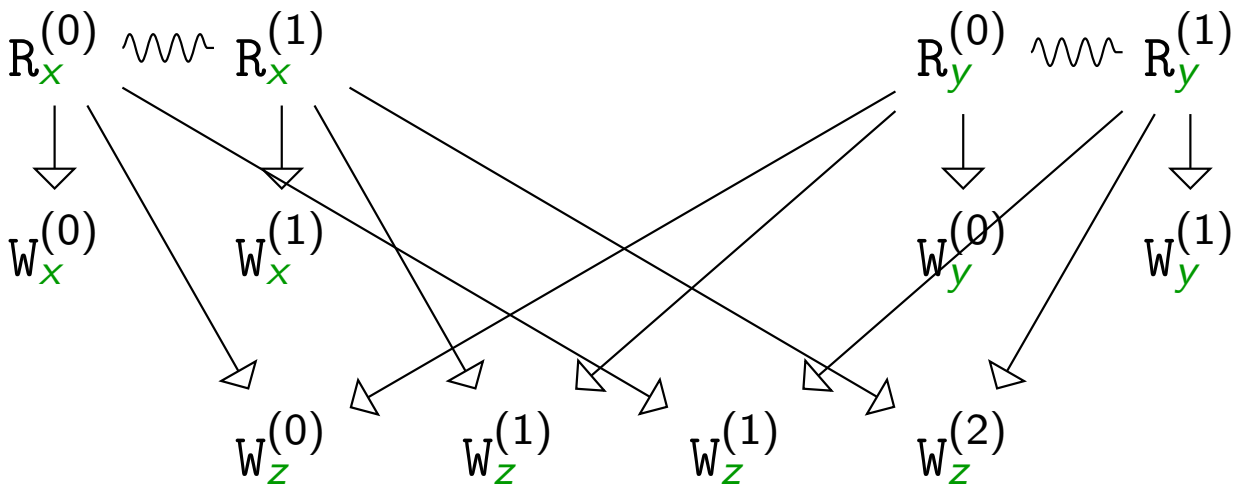
Pour $t = \begin{pmatrix} s \leftarrow x; x := s; \\ t \leftarrow y; y := t; \\ z := s + t \end{pmatrix}$, on a :



(x86)

Exemples

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

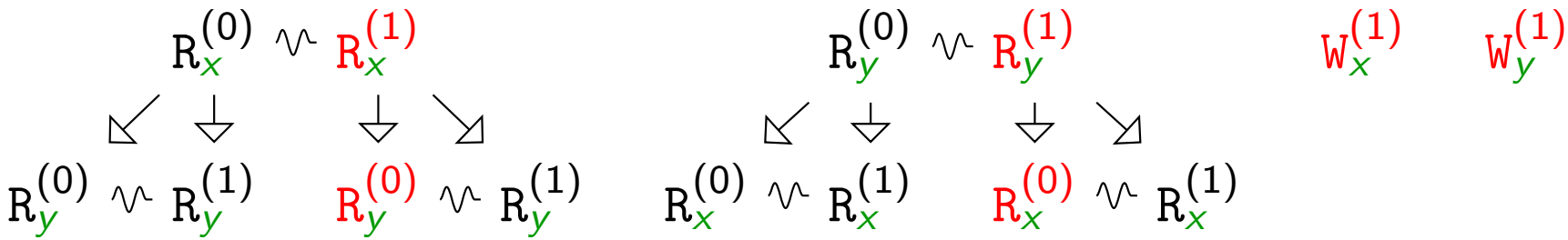
Un exemple

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

Un exemple

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

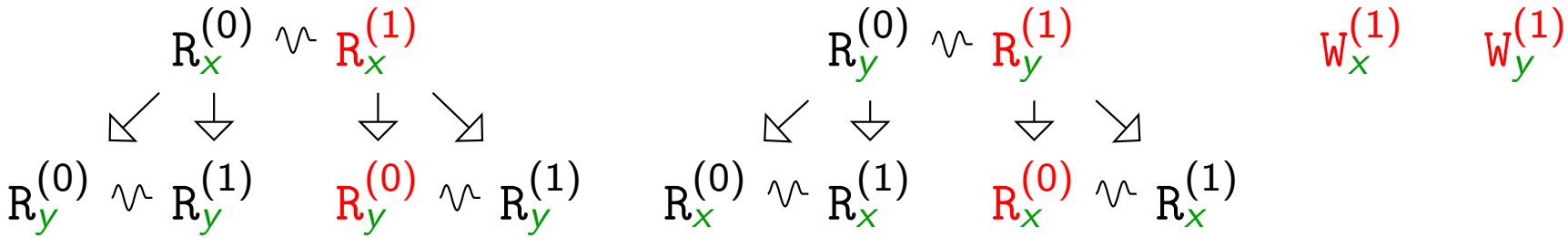
Sémantique ouverte (sur x86) :



Un exemple

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

Sémantique ouverte (sur x86) :



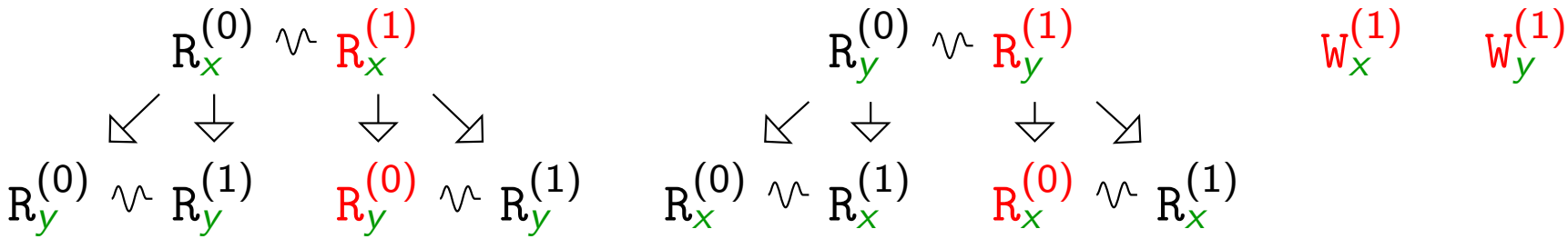
Sémantique close. (Mémoire par traces)

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

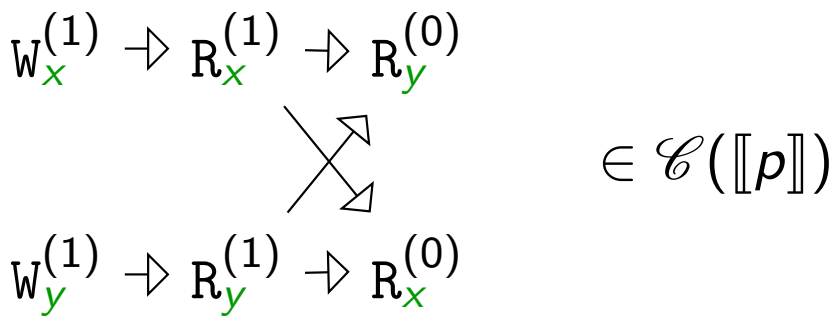
Un exemple

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

Sémantique ouverte (sur x86) :



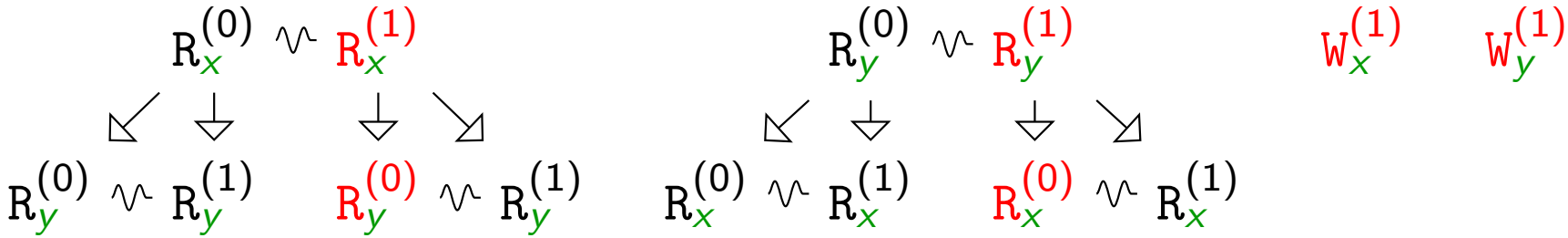
Sémantique close. (Mémoire avec concurrence inter-variable)



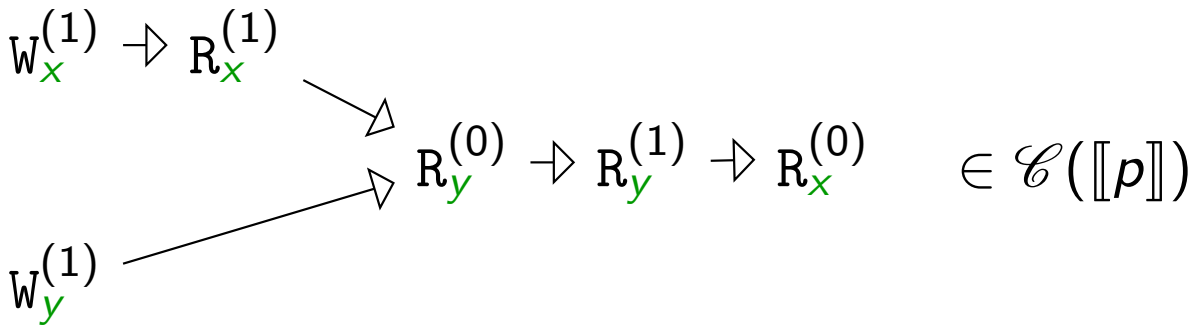
Un exemple

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

Sémantique ouverte (sur x86) :



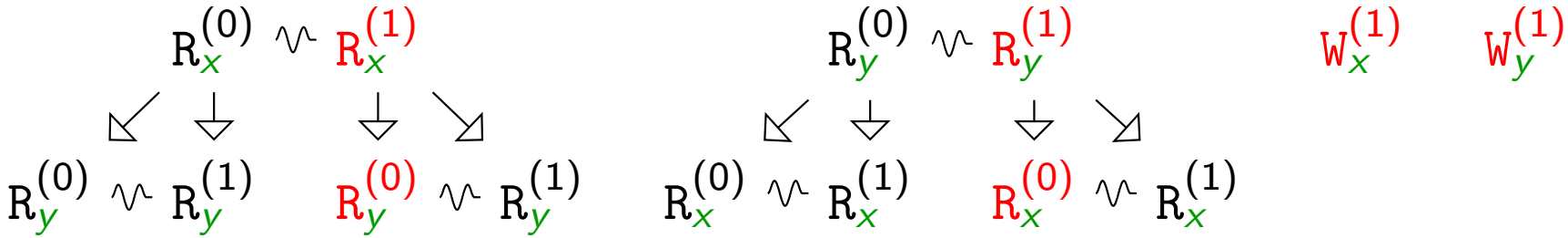
Sémantique close. (Mémoire avec concurrence inter-variable)



Un exemple

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

Sémantique ouverte (sur x86) :



Sémantique close. (Mémoire avec concurrence intra-variable)

$$\mathtt{W}_x^{(1)} \to \mathtt{R}_x^{(1)} \to \mathtt{R}_y^{(0)}$$

$$\in \mathscr{C}([\![p]\!])$$

$$\mathtt{W}_y^{(1)} \to \mathtt{R}_y^{(1)} \to \mathtt{R}_x^{(0)}$$

Conclusion

Pour aller plus loin.

- On peut jouer le même jeu avec des modèles mémoires non-linéaires (traces $\rightarrow$ ordre partiel)
- Modèle inspiré de la sémantique des jeux et simplifié au cadre premier ordre.

Pour plus tard.

- Regarder le comportement des barrières
- Faire le lien avec les sémantiques axiomatiques. (Comparer nos exécutions et les leurs)

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 $M : (\mathbb{N} \to (\mathcal{V} \rightharpoonup \mathbb{N})) \to (\mathcal{V} \rightharpoonup \mathbb{N}) \to \Sigma_{id}^*$:

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

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	$\mathtt{W}_x^{(1)} \quad \mathtt{R}_y^{(0)} \rightsquigarrow \mathtt{R}_y^{(1)} \quad \mathtt{W}_y^{(1)} \quad \mathtt{R}_x^{(0)} \rightsquigarrow \mathtt{R}_y^{(2)}$
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* $\rightarrow$ explosion
- Or, seulement l'histoire d'**une** variable doit l'être.

Et avec une structure d'évènements

- On doit séquentialiser *toute l'exécution* $\rightarrow$ 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 $(w, (t_x))$
  - $w \in \mathscr{C}([\![p]\!]^O)$
  - 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.