Formaliser l'exécution de *LeanMachines* à l'aide du π-calcul

Journées APR

Danaël Carbonneau, Romain Demangeon, Frédéric Peschanski

LIP6, Sorbonne Université, danael.carbonneau@lip6.fr

19 juin 2026

Introduction

- *Framework* pour spécifier et développer des systèmes réactifs

- *Framework* pour spécifier et développer des systèmes réactifs
- Inspiré par la méthode formelle *Event-B*

LeanMachines [CP26]

- *Framework* pour spécifier et développer des systèmes réactifs
- Inspiré par la méthode formelle *Event-B*
- Principe de correction par construction : on prouve la correction du programme durant son développement

Formalisation des événements en *LeanMachines*

```
class Machine (CTX : outParam (Type u)) (M) where
  context : CTX
  invariant : M → Prop
structure Event (M) [Machine CTX M] (α : Type) (β  : Type)
  guard (m : M) (x : α ) : Prop
  action (m : M) (x : α ) (grd : guard m x): (β ×  M)
```

Encodage des machines et des événements dans la bibliothèque *LeanMachines*

Prouver des propriétés sur les événements en *LeanMachines*

```
structure OrdinaryEvent (M) [Machine CTX M] (α : Type) (β  : Type)
    extends Event M α β
    safety (m : M) (x : α) : Machine.invariant m → guard m x → Machine.invariant
    (action m x).2
```

Propriété de *safety* d'un événement dans notre *framework*

Composition d'événements d'une même machine

- Extension du *framework* présentée dans [CP26]

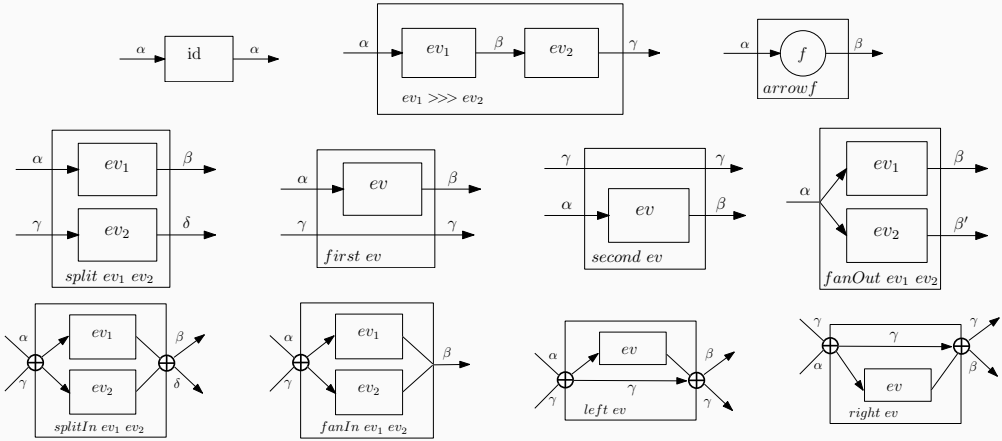
Composition d'événements d'une même machine

- Extension du *framework* présentée dans [CP26]
- Permet de réutiliser le code et les preuves pour définir des événements complexes

Composition d'événements d'une même machine

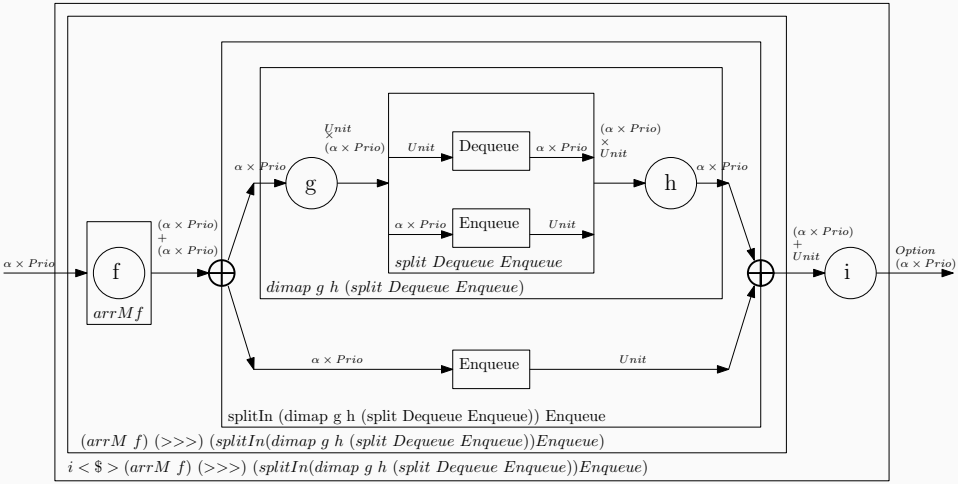
- Extension du *framework* présentée dans [CP26]
- Permet de réutiliser le code et les preuves pour définir des événements complexes
- Abstraction du flot de **contrôle** des événements d'une machine

Composition d'événements d'une même machine



Combinateurs d'événements [CP26]

Exemple de composition d'événements



Événement *PushForce* d'une file de messages obtenue en composant des événements élémentaires (*dequeue* et *enqueue*)

Notre problématique

- En *LeanMachines*, nous avons formalisé les événements (transitions des machines)

Notre problématique

- En *LeanMachines*, nous avons formalisé les événements (transitions des machines)
- Nous pouvons exprimer des propriétés sur les événements (*safety*, raffinement, *etc.*)

Notre problématique

- En *LeanMachines*, nous avons formalisé les événements (transitions des machines)
- Nous pouvons exprimer des propriétés sur les événements (*safety*, raffinement, *etc.*)
- Nous pouvons composer les événements avec des combinateurs algébriques

Notre problématique

- En *LeanMachines*, nous avons formalisé les événements (transitions des machines)
- Nous pouvons exprimer des propriétés sur les événements (*safety*, raffinement, *etc.*)
- Nous pouvons composer les événements avec des combinateurs algébriques
- Afin de pouvoir faire de même pour les machines, il nous faut donner une sémantique à leur exécution.

Étendre le π-calcul avec les *LeanMachines*

Le π-calcul

Le π-calcul

- Algèbre de processus pour modéliser des systèmes concurrents / réactifs

Le π-calcul

- Algèbre de processus pour modéliser des systèmes concurrents / réactifs
- Se concentre sur les **communications** entre les processus (envoi de noms sur des canaux, qui sont aussi des noms)

Le π-calcul

- Algèbre de processus pour modéliser des systèmes concurrents / réactifs
- Se concentre sur les **communications** entre les processus (envoi de noms sur des canaux, qui sont aussi des noms)
- Permet de raisonner sur les propriétés concurrentes de ces processus (fuites d'informations, blocage, divergence, etc.)

Le π-calcul

- Algèbre de processus pour modéliser des systèmes concurrents / réactifs
- Se concentre sur les **communications** entre les processus (envoi de noms sur des canaux, qui sont aussi des noms)
- Permet de raisonner sur les propriétés concurrentes de ces processus (fuites d'informations, blocage, divergence, etc.)

$$P ::= 0 \mid (\nu z)P \mid P \| P \mid\ !x(\tilde{y}).P \mid\ \mid P + P \mid \bar{x}\langle\tilde{y}\rangle.P \mid x(\tilde{y}).P \mid [x = y].P$$

Syntaxe (simplifiée) du π-calcul[SW01]

Sémantique du π-calcul

$$\frac{}{!a(X).P \xrightarrow{a\ V} !a(X).P \| P\{V/X\}} \quad (\textsc{Repl}) \qquad \frac{P \xrightarrow{\bar{a}\ X} P' \qquad Q \xrightarrow{a\ X} Q'}{P \| Q \xrightarrow{\tau} P' \| Q'} \quad (\textsc{Comm-L})$$

$$\frac{}{a(x_1, \ldots, x_n).P \xrightarrow{a\ v_1, \ldots, v_n} P\{{}^{v_1}/_{x_1}, \ldots, {}^{v_n}/_{x_n}\}} \quad (\textsc{In}) \qquad \frac{}{\bar{a}\langle X \rangle.P \xrightarrow{\bar{a}\ X} P} \quad (\textsc{Out})$$

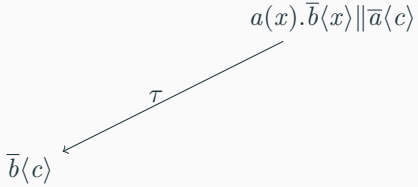
$$\frac{P \xrightarrow{\bar{a}\ n} P'}{\nu(n)P \xrightarrow{\bar{a}(n)} P'} \quad (\textsc{Open})$$

Extrait des règles de transition du π-calcul [SW01]

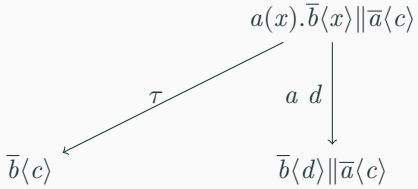
Exemples de processus

$$a(x).\overline{b}\langle x\rangle \| \overline{a}\langle c\rangle$$

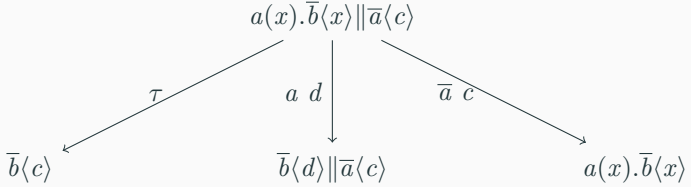
Exemples de processus



Exemples de processus



Exemples de processus



Exemples de processus

$$(\nu\, a)(a(x).\overline{b}\langle x\rangle \| \overline{a}\langle c\rangle)$$

Exemples de processus

$$(\nu a)(a(x).\overline{b}\langle x\rangle \| \overline{a}\langle c\rangle)$$

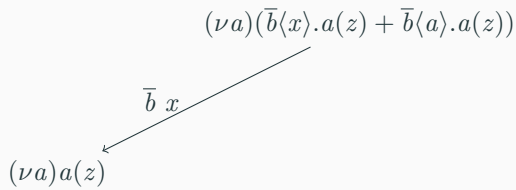
$$\downarrow \tau$$

$$\overline{b}\langle c\rangle$$

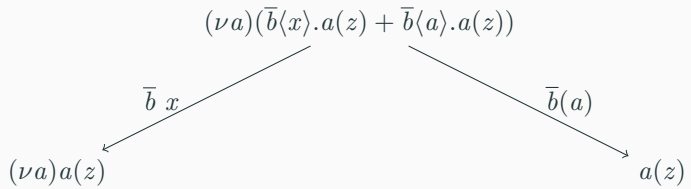
Exemples de processus

$$(\nu a)(\overline{b}\langle x\rangle.a(z) + \overline{b}\langle a\rangle.a(z))$$

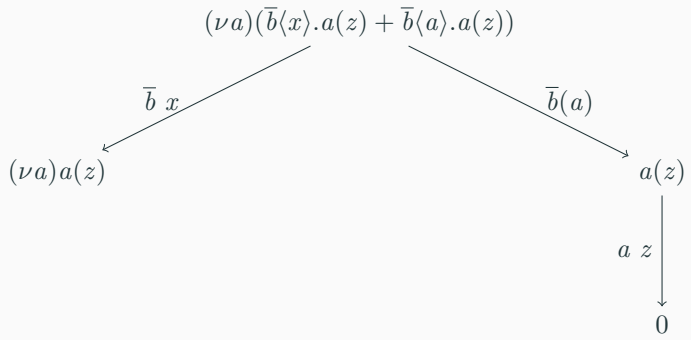
Exemples de processus



Exemples de processus



Exemples de processus



Notre extension du π-calcul

$$P ::= 0 \mid (\nu z)P \mid P \| P \mid !x(\tilde{y}).P \mid \ \mid P + P \mid \bar{x}\langle\tilde{y}\rangle.P \mid x(\tilde{y}).P \mid [x = y].P$$

Notre extension du π-calcul

état interne de type M

$$P ::= 0 \mid (\nu z)P \mid P \| P \mid !x(\tilde{y}).P \mid \ \mid P + P \mid \bar{x}\langle\tilde{y}\rangle.P \mid x(\tilde{y}).P \mid [x = y].P \mid \lfloor m \rfloor_{ctx}, E$$

Notre extension du π-calcul

contexte de type CTX

état interne de type M

$$P ::= 0 \mid (\nu z)P \mid P \| P \mid !x(\tilde{y}).P \mid \ \mid P + P \mid \bar{x}\langle\tilde{y}\rangle.P \mid x(\tilde{y}).P \mid [x = y].P \mid \lfloor m \rfloor_{ctx}^{E}$$

Notre extension du π-calcul

collection d'événements

contexte de type CTX

état interne de type M

$$P ::= 0 \mid (\nu z)P \mid P \| P \mid !x(\tilde{y}).P \mid \ \mid P + P \mid \bar{x}\langle\tilde{y}\rangle.P \mid x(\tilde{y}).P \mid [x = y].P \mid \lfloor m \rfloor_{ctx}^{E}$$

Notre extension du π-calcul

collection d'événements

contexte de type CTX

état interne de type M

$$P ::= 0 \mid (\nu z)P \mid P \| P \mid !x(\tilde{y}).P \mid \ \mid P + P \mid \bar{x}\langle\tilde{y}\rangle.P \mid x(\tilde{y}).P \mid [x = y].P \mid \lfloor m \rfloor_{ctx, E}$$

$$\frac{ev \in E \qquad m, x \xrightarrow{ev} y, m'}{\lfloor m \rfloor_{ctx,E} \xrightarrow{ev\ x,s} \lfloor m' \rfloor_{ctx,E} \mid \bar{s}\langle y \rangle} \quad (\textsc{Event})$$

Sémantique de transition des machines

Exemples de processus

Communications entre deux machines

$$P_{sync} \stackrel{def}{=} !sync(x, s, ev_1, ev_2).(\nu\, c)\overline{ev_1}\langle x, c\rangle.c(y).\overline{ev_2}\langle y, s\rangle$$

Communications entre deux machines

$$!sync(x, s, ev_1, ev_2).(\nu c)(\overline{ev_1}\langle x, c\rangle.c(y).\overline{ev_2}\langle y, s\rangle) \parallel \lfloor m_1 \rfloor_{c_1, ev_1} \parallel \lfloor m_2 \rfloor_{c_2, ev_2}$$

Communications entre deux machines

$$!sync(x, s, ev_1, ev_2).(\nu c)(\overline{ev_1}\langle x, c\rangle.c(y).\overline{ev_2}\langle y, s\rangle) \| \lfloor m_1 \rfloor_{c_1, ev_1} \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

$$sync\ x, s, ev_1, ev_2 \downarrow$$

$$P_{sync} \| (\nu c)(\overline{ev_1}\langle x, c\rangle.c(y).\overline{ev_2}\langle y, s\rangle) \| \lfloor m_1 \rfloor_{c_1, ev_1} \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

Communications entre deux machines

$$!sync(x, s, ev_1, ev_2).(\nu c)(\overline{ev_1}\langle x, c\rangle.c(y).\overline{ev_2}\langle y, s\rangle) \| \lfloor m_1 \rfloor_{c_1, ev_1} \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

$$\downarrow \; sync\ x, s, ev_1, ev_2$$

$$P_{sync} \| (\nu c)(\overline{ev_1}\langle x, c\rangle.c(y).\overline{ev_2}\langle y, s\rangle) \| \lfloor m_1 \rfloor_{c_1, ev_1} \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

$$\downarrow \; \tau$$

$$P_{sync} \| (\nu c)(c(y).\overline{ev_2}\langle y, s\rangle \| \lfloor m_1' \rfloor_{c_1, ev_1} \| \overline{c}\langle y\rangle) \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

Communications entre deux machines

$$!sync(x, s, ev_1, ev_2).(\nu c)(\overline{ev_1}\langle x, c\rangle.c(y).\overline{ev_2}\langle y, s\rangle) \| \lfloor m_1 \rfloor_{c_1, ev_1} \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

$$\downarrow \; sync\ x, s, ev_1, ev_2$$

$$P_{sync} \| (\nu c)(\overline{ev_1}\langle x, c\rangle.c(y).\overline{ev_2}\langle y, s\rangle) \| \lfloor m_1 \rfloor_{c_1, ev_1} \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

$$\downarrow \; \tau$$

$$P_{sync} \| (\nu c)(c(y).\overline{ev_2}\langle y, s\rangle \| \lfloor m_1' \rfloor_{c_1, ev_1} \| \overline{c}\langle y\rangle) \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

$$\downarrow \; \tau$$

$$P_{sync} \| (\nu c)(\overline{ev_2}\langle y, s\rangle \| \lfloor m_1' \rfloor_{c_1, ev_1}) \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

Communications entre deux machines

$$!sync(x, s, ev_1, ev_2).(\nu c)(\overline{ev_1}\langle x, c\rangle.c(y).\overline{ev_2}\langle y, s\rangle) \| \lfloor m_1 \rfloor_{c_1, ev_1} \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

$$\downarrow \; sync\ x, s, ev_1, ev_2$$

$$P_{sync} \| (\nu c)(\overline{ev_1}\langle x, c\rangle.c(y).\overline{ev_2}\langle y, s\rangle) \| \lfloor m_1 \rfloor_{c_1, ev_1} \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

$$\downarrow \; \tau$$

$$P_{sync} \| (\nu c)(c(y).\overline{ev_2}\langle y, s\rangle \| \lfloor m_1' \rfloor_{c_1, ev_1} \| \overline{c}\langle y\rangle) \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

$$\downarrow \; \tau$$

$$P_{sync} \| (\nu c)(\overline{ev_2}\langle y, s\rangle \| \lfloor m_1' \rfloor_{c_1, ev_1}) \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

$$\downarrow \; \tau$$

$$P_{sync} \| (\nu c)(\lfloor m_1' \rfloor_{c_1, ev_1}) \| \lfloor m_2' \rfloor_{c_2, ev_2} \| \overline{s}\langle y_2\rangle$$

Communications entre deux machines

$$!sync(x, s, ev_1, ev_2).(\nu c)(\overline{ev_1}\langle x, c\rangle.c(y).\overline{ev_2}\langle y, s\rangle) \| \lfloor m_1 \rfloor_{c_1, ev_1} \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

$$\downarrow \; sync\ x, s, ev_1, ev_2$$

$$P_{sync} \| (\nu c)(\overline{ev_1}\langle x, c\rangle.c(y).\overline{ev_2}\langle y, s\rangle) \| \lfloor m_1 \rfloor_{c_1, ev_1} \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

$$\downarrow \; \tau$$

$$P_{sync} \| (\nu c)(c(y).\overline{ev_2}\langle y, s\rangle \| \lfloor m_1' \rfloor_{c_1, ev_1} \| \overline{c}\langle y\rangle) \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

$$\downarrow \; \tau$$

$$P_{sync} \| (\nu c)(\overline{ev_2}\langle y, s\rangle \| \lfloor m_1' \rfloor_{c_1, ev_1}) \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

$$\downarrow \; \tau$$

$$P_{sync} \| \lfloor m_1' \rfloor_{c_1, ev_1} \| \lfloor m_2' \rfloor_{c_2, ev_2} \| \overline{s}\langle y_2 \rangle$$

$$c \notin variables(P) \implies (\nu c)P \equiv P$$

Communications entre deux machines

$$!sync(x, s, ev_1, ev_2).(\nu c)(\overline{ev_1}\langle x, c\rangle.c(y).\overline{ev_2}\langle y, s\rangle) \| \lfloor m_1 \rfloor_{c_1, ev_1} \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

$$\downarrow \; sync\ x, s, ev_1, ev_2$$

$$P_{sync} \| (\nu c)(\overline{ev_1}\langle x, c\rangle.c(y).\overline{ev_2}\langle y, s\rangle) \| \lfloor m_1 \rfloor_{c_1, ev_1} \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

$$\downarrow \; \tau$$

$$P_{sync} \| (\nu c)(c(y).\overline{ev_2}\langle y, s\rangle \| \lfloor m_1' \rfloor_{c_1, ev_1} \| \overline{c}\langle y\rangle) \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

$$\downarrow \; \tau$$

$$P_{sync} \| (\nu c)(\overline{ev_2}\langle y, s\rangle \| \lfloor m_1' \rfloor_{c_1, ev_1}) \| \lfloor m_2 \rfloor_{c_2, ev_2}$$

$$\downarrow \; \tau$$

$$P_{sync} \| \lfloor m_1' \rfloor_{c_1, ev_1} \| \lfloor m_2' \rfloor_{c_2, ev_2} \| \overline{s}\langle y_2 \rangle \xrightarrow{\overline{s}\ y_2} P_{sync} \| \lfloor m_1' \rfloor_{c_1, ev_1} \| \lfloor m_2' \rfloor_{c_2, ev_2}$$

Partage d'événements

- Il peut y avoir plusieurs occurrences d'une même spécification dans un processus,

$$\lfloor m_1 \rfloor_{c_1,E} \parallel \lfloor m_2 \rfloor_{c_2,E}$$

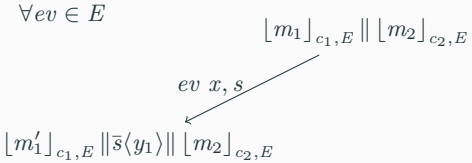
Partage d'événements

- Il peut y avoir plusieurs occurrences d'une même spécification dans un processus,
- elles ont la même collection de noms d'événements

$$\forall ev \in E \qquad \lfloor m_1 \rfloor_{c_1,E} \parallel \lfloor m_2 \rfloor_{c_2,E}$$

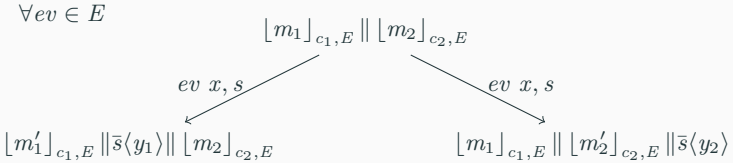
Partage d'événements

- Il peut y avoir plusieurs occurrences d'une même spécification dans un processus,
- elles ont la même collection de noms d'événements



Partage d'événements

- Il peut y avoir plusieurs occurrences d'une même spécification dans un processus,
- elles ont la même collection de noms d'événements



Partage d'événements et renommage

- On peut distinguer les événements de deux instances d'une même machine grâce aux **canaux privés**

$$((\nu\ ev)\ \lfloor m_1 \rfloor_{c_1,E} \parallel !ev'(x,s).\overline{ev}\langle x,s\rangle) \parallel \lfloor m_2 \rfloor_{c_2,E}$$

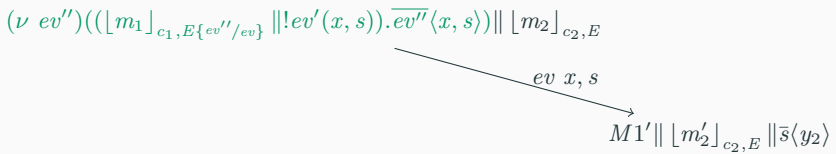
Partage d'événements et renommage

- On peut distinguer les événements de deux instances d'une même machine grâce aux **canaux privés** et à l'$\alpha-$**renommage**

$$
\begin{aligned}
&((\nu\ ev)\ \lfloor m_1 \rfloor_{c_1,E} \parallel !ev'(x,s).\overline{ev}\langle x,s\rangle) \parallel \lfloor m_2 \rfloor_{c_2,E} \\
\overset{\alpha}{=}\ &((\nu\ ev'')\ \lfloor m_1 \rfloor_{c_1,E\{^{ev''}/_{ev}\}} \parallel !ev'(x,s).\overline{ev''}\langle x,s\rangle) \parallel \lfloor m_2 \rfloor_{c_2,E}
\end{aligned}
$$

Partage d'événements et renommage

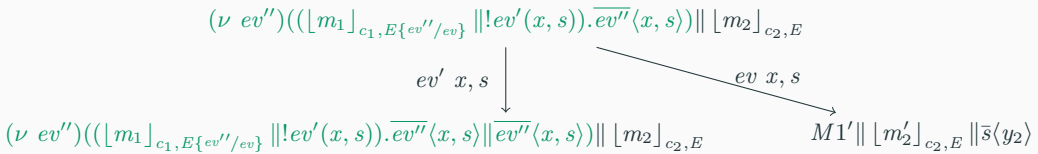
- On peut distinguer les événements de deux instances d'une même machine grâce aux **canaux privés** et à l'$\alpha-$**renommage**



$$M_1' \stackrel{def}{=} (\nu\ ev'')((\lfloor m_1 \rfloor_{c_1,E\{ev''/ev\}} \| !ev'(x,s)).\overline{ev''}\langle x,s\rangle)$$

Partage d'événements et renommage

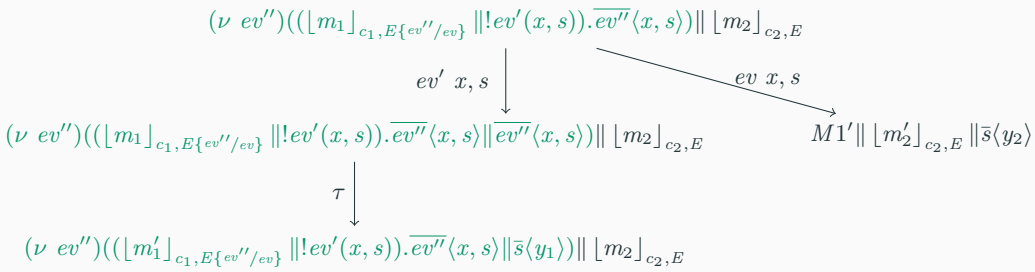
- On peut distinguer les événements de deux instances d'une même machine grâce aux **canaux privés** et à l'$\alpha-$**renommage**



$$M_1' \stackrel{def}{=} (\nu\ ev'')((\lfloor m_1 \rfloor_{c_1,E\{ev''/ev\}} \| !ev'(x,s)).\overline{ev''}\langle x,s\rangle)$$

Partage d'événements et renommage

- On peut distinguer les événements de deux instances d'une même machine grâce aux **canaux privés** et à l'$\alpha-$**renommage**



$$M_1' \stackrel{def}{=} (\nu\ ev'')((\lfloor m_1 \rfloor_{c_1,E\{ev''/ev\}} \| !ev'(x,s)).\overline{ev''}\langle x,s\rangle)$$

Initialisation dynamique

- On souhaite décrire dynamiquement l'initialisation d'une machine.

Initialisation dynamique

- On souhaite décrire dynamiquement l'initialisation d'une machine.
- En *LeanMachines*, il y a des événements d'initialisation.

$$P_1 \stackrel{\text{def}}{=} \lfloor m_0 \rfloor_{ctx, E \cup \{init\}}$$

Initialisation dynamique

- On souhaite décrire dynamiquement l'initialisation d'une machine.
- En *LeanMachines*, il y a des événements d'initialisation.

$$m_0, x \xrightarrow{ev} m', y$$

$$P_1 \stackrel{\text{def}}{=} \lfloor m_0 \rfloor_{ctx, E \cup \{init\}} \xrightarrow{ev\ x, s} \lfloor m' \rfloor_{ctx, E \cup \{init\}} \parallel \overline{s}\langle y \rangle$$

Initialisation dynamique

- On souhaite décrire dynamiquement l'initialisation d'une machine.
- En *LeanMachines*, il y a des événements d'initialisation.
- Il nous faut un moyen d'encoder une machine qui ne peut pas faire d'autres transitions avant son initialisation.

$$m_0, x \xrightarrow{ev} m', y$$

$$P_1 \stackrel{\text{def}}{=} \lfloor m_0 \rfloor_{ctx, E \cup \{init\}} \xrightarrow{ev\ x, s} \lfloor m' \rfloor_{ctx, E \cup \{init\}} \parallel \overline{s}\langle y \rangle$$

Initialisation dynamique

- On souhaite décrire dynamiquement l'initialisation d'une machine.
- En *LeanMachines*, il y a des événements d'initialisation.
- Il nous faut un moyen d'encoder une machine qui ne peut pas faire d'autres transitions avant son initialisation.

$$P_2 \stackrel{\text{def}}{=} (\nu E)\, \lfloor m \rfloor_{ctx,E}$$

Initialisation dynamique

- On souhaite décrire dynamiquement l'initialisation d'une machine.
- En *LeanMachines*, il y a des événements d'initialisation.
- Il nous faut un moyen d'encoder une machine qui ne peut pas faire d'autres transitions avant son initialisation.

$$P_2 \stackrel{\text{def}}{=} (\nu E)\, \lfloor m \rfloor_{ctx,E}$$

$$\nexists P' \ tq.\ P_2 \xrightarrow{\pi} P'$$

Initialisation dynamique

- On souhaite décrire dynamiquement l'initialisation d'une machine.
- En *LeanMachines*, il y a des événements d'initialisation.
- Il nous faut un moyen d'encoder une machine qui ne peut pas faire d'autres transitions avant son initialisation.

Pour une machine d'événement initial *init* ayant une collection d'événements E, on pose

$$m_0, x \xrightarrow{init} m_{init}, E$$

$$P_3 \stackrel{\text{def}}{=} (\nu E) \lfloor m_0 \rfloor_{ctx, E \cup \{init\}}$$

Initialisation dynamique

- On souhaite décrire dynamiquement l'initialisation d'une machine.
- En *LeanMachines*, il y a des événements d'initialisation.
- Il nous faut un moyen d'encoder une machine qui ne peut pas faire d'autres transitions avant son initialisation.

Pour une machine d'événement initial *init* ayant une collection d'événements E, on pose

$$m_0, x \xrightarrow{init} m_{init}, E$$

$$P_3 \stackrel{\text{def}}{=} (\nu E)\, \lfloor m_0 \rfloor_{ctx, E \cup \{init\}} \xrightarrow{init\ x, s} (\nu E)\, \lfloor m_{init} \rfloor_{ctx, E \cup \{init\}} \parallel \overline{s}\langle E \rangle$$

Initialisation dynamique

- On souhaite décrire dynamiquement l'initialisation d'une machine.
- En *LeanMachines*, il y a des événements d'initialisation.
- Il nous faut un moyen d'encoder une machine qui ne peut pas faire d'autres transitions avant son initialisation.

Pour une machine d'événement initial *init* ayant une collection d'événements E, on pose

$$m_0, x \xrightarrow{init} m_{init}, E$$

$$P_3 \overset{\text{def}}{=} (\nu E)\, \lfloor m_0 \rfloor_{ctx, E \cup \{init\}} \xrightarrow{init\ x, s} (\nu E)\, \lfloor m_{init} \rfloor_{ctx, E \cup \{init\}} \,\|\overline{s}\langle E \rangle$$

$$\Big\downarrow \overline{s}(E)$$

$$\lfloor m_{init} \rfloor_{ctx, E \cup \{init\}}$$

Conclusion et travaux futurs

- Extension du π-calcul afin de donner une sémantique aux machines de *LeanMachines*

Conclusion et travaux futurs

- Extension du π-calcul afin de donner une sémantique aux machines de *LeanMachines*
- Approche qui permet de composer plusieurs machines dans un même processus

Conclusion et travaux futurs

- Extension du π-calcul afin de donner une sémantique aux machines de *LeanMachines*
- Approche qui permet de composer plusieurs machines dans un même processus
- Permet d'exprimer des processus complexes (initialisation dynamique, indirections)

Conclusion et travaux futurs

- Extension du π-calcul afin de donner une sémantique aux machines de *LeanMachines*
- Approche qui permet de composer plusieurs machines dans un même processus
- Permet d'exprimer des processus complexes (initialisation dynamique, indirections)

Travaux futurs

Conclusion et travaux futurs

- Extension du π-calcul afin de donner une sémantique aux machines de *LeanMachines*
- Approche qui permet de composer plusieurs machines dans un même processus
- Permet d'exprimer des processus complexes (initialisation dynamique, indirections)

Travaux futurs

- Utiliser des outils des algèbres de processus pour vérifier des propriétés sur nos machines (systèmes de types)

Conclusion et travaux futurs

- Extension du π-calcul afin de donner une sémantique aux machines de *LeanMachines*
- Approche qui permet de composer plusieurs machines dans un même processus
- Permet d'exprimer des processus complexes (initialisation dynamique, indirections)

Travaux futurs

- Utiliser des outils des algèbres de processus pour vérifier des propriétés sur nos machines (systèmes de types)
- Définir une notion d'équivalence de processus

Conclusion et travaux futurs

- Extension du π-calcul afin de donner une sémantique aux machines de *LeanMachines*
- Approche qui permet de composer plusieurs machines dans un même processus
- Permet d'exprimer des processus complexes (initialisation dynamique, indirections)

Travaux futurs

- Utiliser des outils des algèbres de processus pour vérifier des propriétés sur nos machines (systèmes de types)
- Définir une notion d'équivalence de processus
- Encodage du $\pi-$*calcul* en *Lean4* pour étendre le *framework*

Bibliographie i

[CP26] Danaël Carbonneau and Frederic Peschanski, *LeanMachines : State-based Modeling with Refinement (a Lean4 Framework)*, Form. Asp. Comput. (2026), Just Accepted.

[SW01] Davide Sangiorgi and David Walker, *The Pi-Calculus : A Theory of Mobile Processes*, Cambridge University Press, Cambridge, 2001.