Laboratoire d'Informatique Signaux et Systèmes (I3S)
Université de Nice-Sophia-Antipolis
C.N.R.S - U.R.A. 1376
Thème Architectures Logicielles et Matérielles
Équipe Lambda-X
41 Bd. Napoléon III – 06041 NICE Cédex

λ-matrice

Modélisation et simulation algébrique et fonctionnelle des modèles d'états

Guilhem de Wailly et Fernand Boéri

Résumé

Dans ce rapport interne, nous présentons ici un formalisme algébrique et fonctionnel destiné à la description et à la résolution des systèmes basés sur un modèle d'état, et plus particulièrement, des systèmes de traitement du signal. Ce modèle original est proche du concept des graphes *Data Flow* et permet une représentation graphique des applications. Construit à l'aide de l'algèbre linéaire et du λ-calcul, proche de la sémantique dénotationnelle, il autorise la conception modulaire d'un système, la connaissance de ses caractéristiques — complétude, calculabilité, stabilité, mesurabilité — et sa résolution déterministe. L'implantation dans une architecture matérielle peut exploiter le parallélisme mis en évidence de manière naturelle.

Table des matières

1 Introduction

Un outil de modélisation des systèmes[1] de traitement du signal doit impérativement bénéficier d'un système de preuve permettant de connaître de manière précise ses caractéristiques. En effet, l'aboutissement de ce type d'outil est une implantation sur une architecture matérielle réduite, optimisée pour la vitesse et la sécurité de fonctionnement, éventuellement parallèle [AB86, CT93].

1. Dans ce qui suit, le mot *application* est réservé pour désigner l'application d'un opérateur à des arguments ; dans ce contexte, pour désigner une application de traitement du signal, on utilisera le mot *système*.

La connaissance des caractéristiques du système oriente la réalisation de l'architecture de manière adaptée. Notamment, la gestion de la mémoire pourra être dynamique ou statique, selon que le système est « stable » ou non.

Les applications de traitement du signal [Cha92, Kun80, Miq85] utilisent fréquemment des modules conçus séparément. Cette modularisation permet de préserver les investissements entrepris en autorisant la réutilisation des parties d'un système. L'ensemble de ces parties forme une bibliothèque. L'outil de conception doit donc aussi offrir la possibilité de concevoir ces modules et de les utiliser. Dès qu'il est question de modules, apparaissent les notions d'interface contractuelle, de compilation séparée et d'édition des liens, un peu à la manière des langages compilés actuels.

La forme de description la plus souvent utilisée pour les systèmes de traitement du signal est la représentation graphique. Dans cette forme de représentation, les boîtes sont des fonctions, et les traits sont des données. En fait, les boîtes sont soit des fonctions primitives, soit des modules. L'outil doit donc aussi faciliter la réalisation d'une sur-couche graphique.

Pour prendre en considération ces contraintes de manière efficace, les λ-matrices utilisent des outils à la fois primitifs, parfaitement connus et extrêmement puissants que sont l'algèbre linéaire [Mey92, NR85] et le λ-calcul [Kri90, Rev88]. L'algèbre linéaire est utilisée pour la description des objets manipulés dans les λ-matrices. Le λ-calcul, quant à lui, va permettre de concevoir les opérateurs manipulant ces objets de manière fonctionnelle.

L'utilisation conjointe de ces deux outils aboutit forme un formalisme proche de la sémantique dénotationnelle [Llo86, Mey92, Sto77, Str66] permettant la description des systèmes de traitement du signal entièrement fonctionnelle et formalisée. La résolution de ces systèmes obéit aussi à cette règle. Ainsi, les λ-matrices ont pour caractéristiques :

- formalisation algébrique entièrement exprimable avec le λ-calcul ;
- connexion implicite avec la théorie des graphes *Data Flow* [WA85, LM87] ;
- représentation intermédiaire à la fois de haut et de bas niveau, proche de la machine, proche de l'outil de conception en amont, avec un jeu de formes syntaxiques, ce qui permet une transition immédiate vers les outils de conception que sont λ-graphe et λ-flot [2] ;
- indépendance des objets manipulés et les opérateurs qui s'y appliquent ;
- possibilité de construire des structures dynamiques ;
- existence de critères dénommés $\Lambda(,)$ caractérisant les λ-matrices : complétude, calculabilité, stabilité, mesurabilité des temps de calcul ;

2. Ces outils ne seront pas décrits ici.

- modularisation des systèmes, possibilité de compilation séparée destructive et d'édition des liens ;
- parallélisme directement exploitable, car les dépendances fonctionnelles sont clairement isolées.

L'ensemble de ces caractéristiques font des λ-matrices un outil de modélisation à la fois simple, puissant et entièrement formalisé. Il est destiné à servir de base, d'une part, aux outils de conception – langage de haut niveau et langage graphique – destiné au concepteur de systèmes et d'autre part, aux outils de transformation de bas niveau – compilation et édition de liens, générateur d'architecture – destinés au concepteur d'architectures. Ces outils sortent cependant du cadre de rapport.

Après avoir définit brièvement les modèles d'état simples (cf. section 2) et composés (cf. section 3), les éléments de base des λ-matrices sont introduits (cf. section 4). Ensuite, la notion importante des environnements est décrite (cf. section 5). Alors, les opérateurs dynamiques caractérisant un système et capables de le faire évoluer dans le temps seront abordés (cf. section 6). Quelques compléments nécessaires à la construction de systèmes seront expliqués (cf. section 7). Pour finir, les possibilités d'implantation matérielles des λ-matrices seront introduites (cf. section 8).

Dans les annexes, on trouvera les compléments (cf. annexe A) et les exemples (cf. annexe B), les preuves formelles (cf. annexe C).

2 Modèle d'état simples

Le modèle d'état décrit ici est dit simple. Les fonctions des entrées et les fonction des états sont des composition de fonctions élémentaires, comme on peut le voir dans la figure 1.

Les valeurs traitées sont des vecteurs de valeurs de taille fixe, symbolisés sur la figure par un trait oblique portant la largeur du vecteur en indice. Les largeurs sont ici I, J et K.

Les variables d'état sont des opérateurs qui enregistrent sur un front de l'horloge la valeur en entrée pour la reproduire sur la sortie. Cette valeur reste inchangée, jusqu'au front d'horloge suivant. En automatisme, il s'agit du retard unitaire.

La valeur initiale est une pré-contrainte du système à sa mise en route. Elles ne jouent directement un rôle qu'au temps initial.

La boîte fonctionnelle de ce modèle calcule en permanence, en fonction des entrées, le vecteur de sortie et le vecteur matérialisant l'état du système. Cette boîte fonctionnelle est constituée de fonctions construites par composition, excluant toute équation de point-fixe.

L'horloge de ce système joue un triple rôle en agissant à la fois sur le convertisseur — ou le mécanisme d'acquisition — des valeurs d'entrée, sur la mise à disposition des valeurs de sortie et sur le recyclage du vecteur d'état. Notons qu'un

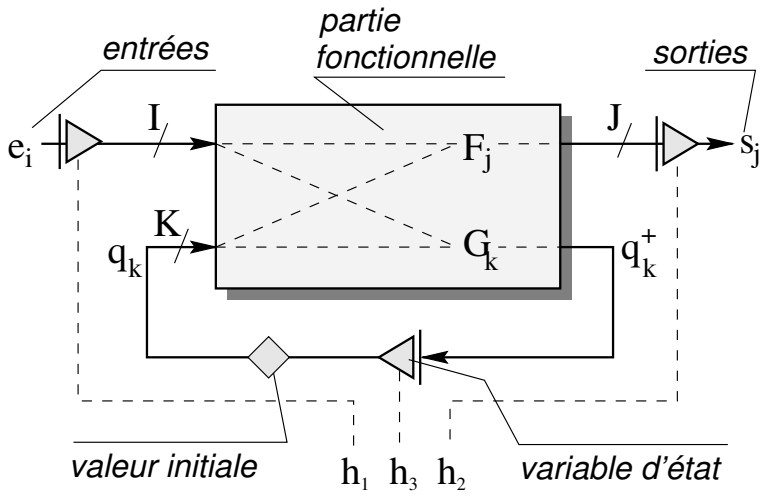


FIG. 1 - Représentation des systèmes.

système à plusieurs horloges fixes, souvent appelé filtre multicadence [Kun80], peut toujours se ramener à un modèle à une seule horloge. En effet, dans ce modèle, toutes les horloges sont des multiples constants d'une horloge de base. Il est donc toujours possible de synchroniser deux flots de valeurs régis par des horloges différentes par ajout ou suppression de valeurs

Il apparaît donc un cycle immuable dans le fonctionnement de ce modèle, que l'on peut résumer par : saisie des valeurs d'entrée (h_1), saisie des valeur de sortie (h_2), et saisie des valeurs d'état (h_3). Ce cycle se répète sans fin, à moins qu'une condition d'arrêt soit jointe au modèle.

Voici les équations décrivant entièrement ce modèle pour la variable $t = [1, \infty[$, représentant le temps :

$$\begin{cases} e_i(t) & = E_i(t) \\ s_j(t) & = F_j(e_1(t), \ldots, e_I(t), q_1(t), \ldots, q_K(t)) \\ q_k(1) & = Q_k(1) \\ q_k(t+1) & = G_k(e_1(t), \ldots, e_I(t), q_1(t), \ldots, q_K(t)) \end{cases} \tag{1}$$

Le système est entièrement décrit par la connaissances du nombre des entrée, I, du nombre des sorties J et du nombre des états K, des fonctions F_j et G_k, des valeurs initiale $Q_k(1)$ des états. Son fonctionnement dépend de la valeur des entrées $E_i(t)$. La valeurs des sortie au cours du temps est :

$$\begin{array}{ll} s_j(1) = & F_j(e_1(1), \ldots, e_I(1), q_1(1), \ldots, q_K(1)), \\ s_j(2) = & F_j(e_1(2), \ldots, e_I(2), q_1(2), \ldots, q_K(2)), \\ \ldots & , \\ s_j(n) = & F_j(e_1(n), \ldots, e_I(n), q_1(n), \ldots, q_K(n)). \end{array} \tag{2}$$

avec comme valeur d'état :

$$\begin{array}{ll} q_k(1) = & Q_k(1), \\ q_k(2) = & G_k(e_1(1), \ldots, e_I(1), q_1(1), \ldots, q_K(1)), \\ \ldots & , \\ q_k(n) = & G_k(e_1(n), \ldots, e_I(n), q_1(n), \ldots, q_K(n)). \end{array} \tag{3}$$

On peut donner une autre représentation de ce modèle. Il s'agit de la représentation par chronogrammes, comme on peut voir dans la figure 2.

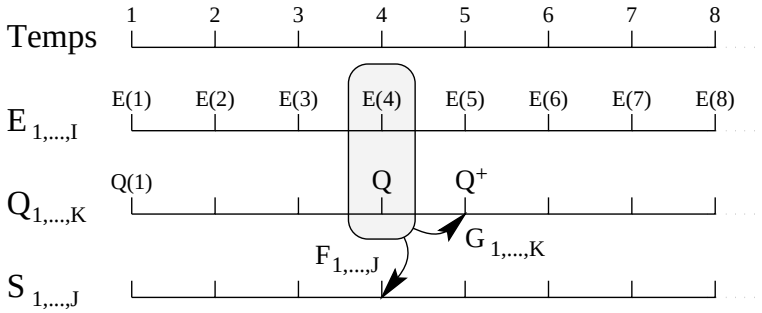


FIG. 2 - Chronogramme des équations du système.

Dans ce chronogramme, on voit clairement apparaître les dépendances fonctionnelles. Les sorties sont contemporaines aux entrées et aux états via les fonctions F_j. Les états sont calculés avec les fonctions G_k appliquées aux entrées et aux états de l'instant précédent.

Ainsi, si l'on connaît les valeurs d'entrée, les valeurs initiales des états, les fonctions F_i et G_k, le système est complètement déterministe, et on peut alors obtenir par calculs successifs les sorties à tout instant.

3 Modèle d'état composé

Le modèle d'état vu dans la section précédente peut être facilement modélisé d'une manière fonctionnelle. Cependant, avant d'entamer cette description, remarquons qu'il souffre de l'absence complète de modularisation. En effet, ces composants sont des fonctions au sens strict.

Pour permettre cette modularisation, il est nécessaire d'autoriser le concepteur à utiliser à l'intérieur même du modèle des variables d'état, comme on peut le voir dans l'exemple de la figure 3. Dans un tel modèle, une boîte représente soit une fonction pure, soit un modèle d'état. Aucun signe extérieur ne les distingue, si se n'est la documentation du module. Dès lors, la mise en équation triviale qui a été faite dans la section précédente n'apparaît plus aussi simplement.

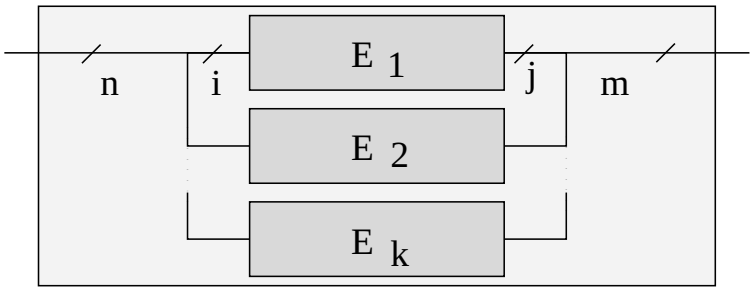


FIG. 3 - Représentation des systèmes complexes : boîte fonctionnelle constituée de modèles d'état.

La première solution est de conserver dans les modules un lien sur les variables d'états. Au moment de l'exécution du système, toutes ces variables sont rassemblées pour former un vecteur, et on se ramène à la description du modèle d'état simple vue dans la section précédente. On se heurte alors à au moins deux difficultés. Premièrement, si on désire prouver un système et sa résolution, il faut prouver le processus de transformation du modèle composé vers le modèle simple, et le modèle simple. Il s'agit en fait de prouver le compilateur de système et la machine d'exécution. Deuxièmement, la modularisation proposée n'est pas réelle, car on est obligé de conserver une trace de la structure des modules afin de récupérer au moment de l'exécution leurs variables d'état.

Les λ-matrices se proposent de résoudre ces problème en apportant un système de preuve au niveau du modèle composé, et permettent la caractérisation des modules séparément.

4 Les λ-matrices

4.1 Présentation

Le concept de base des λ-matrices est la transformation des équations du modèle d'état (1) en des équations de flots de données :

$$\begin{cases} q_k = Q_k(1)\, f_{by}\, G_k(e_1, \ldots, e_K, q_1, \ldots, q_K) \\ s_j = F_j(e_1, \ldots, e_K, q_1, \ldots, q_K) \end{cases} \tag{4}$$

et de les composer de manière à construire un modèle d'état complexe.

Dans ces équations, un « flot de données » constitué d'une valeur initiale — l'état — et d'une fonction de calcul des valeurs suivantes — le contrat — permet d'élaborer les variables d'état. Avec cette transformation, le concepteur est capable de décrire un système modulaire. Cette transformation des équations est la base des λ-matrices, qui, de plus, permettent :

– la *description* du système qui supporte deux niveaux, l'un enrichi syntaxi-

quement destiné au concepteur, et l'autre uniforme destiné à faciliter la représentation en machine[3], se fait à l'aide d'objets élémentaires combinés entre eux ;

- sa *caractérisation*, à l'aide des fonctions de critère ;
- sa *résolution*, avec plusieurs opérateurs spéciaux dits dynamiques.

4.1.1 Description

Afin de construire un système complexe, il faut disposer d'éléments primitifs, les *atomes*, et d'éléments composés, les *objets*. Les atomes sont :

- les *données* manipulées par le système, indépendantes du formalisme lui-même ;
- les *entiers* naturels ;
- les *opérateurs* disponibles pour manipuler les objets et les données ;
- les *symboles* qui sont des synonymes d'autres atomes ou objets.

Les données manipulées sont indépendantes des λ-matrices elles-mêmes. Il faut donc fournir une « algèbre des données » pour les utiliser. Avec les atomes il est possible de construire les objets composés suivants :

- les *alternatives* qui permettent d'effectuer des choix dans les données ;
- les *applications* qui permettent d'appliquer un opérateur à un certain nombre d'arguments ;
- les *définitions* qui permettent de donner un nom aux objets ;
- les *flots* qui permettent de construire les modèles d'état ;
- les *vecteurs* qui permettent de regrouper dans une structure commune des objets entre eux. Les vecteurs sont aussi associés à la construction des environnements.

4.1.2 Caractéristiques

Les caractéristiques d'un système sont déterminées en lui appliquant des fonctions dites *de critères*. Le résultat d'un critère est 0 si le système n'a pas la caractéristique recherchée, ou 1 s'il l'a. Nous avons les caractéristiques suivantes :

- *complétude* : une λ-matrice est complète lorsqu'elle ne contient pas de variable libre (cf. définition 5.5) ;

3. Le passage de l'un à l'autre est trivial et ne sera pas décrit.

- *calculabilité* : une λ-matrice est calculable lorsqu'elle est complète et qu'elle ne contient pas d'équation de point-fixe ;
- *stabilité* : une λ-matrice est stable lorsque elle est calculable et que sa dimension (cf. définition 4.3) est décroissante au cours du temps ;
- *mesurabilité* : une λ-matrice est mesurable lorsqu'elle est stable et que les temps de calculs sont décroissants.

Ces caractéristiques sont de plus en plus contraignantes pour le concepteur du système. En effet, les systèmes mesurable sont un sous-ensembles des systèmes stables, eux-même sous-ensemble des systèmes calculables, jusqu'aux système complets qui sont un sous-ensemble de l'ensemble des systèmes.

4.1.3 Résolution

Lors de la description du modèle d'état, nous avons vu que sa résolution comporte trois étapes. Ces étapes se retrouvent dans les λ-matrices : la saisie de la valeurs d'entrées et de sortie se nomme « évaluation », la saisie de la valeur d'état se nomme « régénération ».

Le processus de résolution d'une λ-matrice se doit d'être fonctionnel : il s'agit d'une fonction récursive à récursion de queue, qui est la vision fonctionnelle de l'itération. Ce processus alterne évaluations et régénérations. Pour pouvoir contrôler la durée de ce processus, une condition d'arrêt lui est adjoint (cf. section 6).

Comme le formalisme est fonctionnel, le système est un paramètre de opérateur de résolution. Le front d'horloge, ou le pas de résolution sont matérialisés par deux appels successifs à l'opérateur de résolution, avec un système régénéré. L'une des principale difficulté rencontrée est de montrer que si un critère est vérifié, il l'est aussi pour son régénéré. La stabilité, par exemple, signifie en fait que la λ-expression représentant le système ne doit pas « grossir » d'une régénération à l'autre.

Afin d'apporter ce genre de preuve, il est nécessaire de formaliser entièrement les λ-matrices. Dans cette optique, l'algèbre linéaire permet de décrire entièrement les éléments constituants. Quant au λ-calcul, son caractère fonctionnel permet d'établir des preuves solides.

4.2 Définition

Cette section se propose de donner une définition algébriques des termes des λ-matrices. Cette définition mathématique apporte avec elle sa précision et sa concision. Elle ouvre la voie à la définition de fonctions au sens mathématiques opérant sur ces termes.

Les termes définis ici ont au moins deux écritures : une écriture « syntaxique » qui permet de donner un sens au terme en le lisant, comme $s := v$ qui est la définition de s par v, et une écriture « uniforme », c'est à dire identique pour tous

les termes, comme $\{s, v\}_{\bar{d}}$. La première forme est proche du programmeur alors que la seconde est proche de la machine.

Voici la définition des termes des λ-matrices :

- soient $\mathbb{D}$ l'ensemble des données manipulées et $\mathbb{O}_D$ l'ensemble des opérateurs qui s'y rapportent [4] ;
- soient $\mathbb{N}$ l'ensemble des entiers naturels, $\mathbb{I}$ l'ensemble des symboles, $\mathbb{O}$ l'ensemble des opérateurs composé de l'ensemble des opérateurs spéciaux $\mathbb{O}_S$ et de l'ensemble des opérateurs sur les données $\mathbb{O}_D$;
- soit $\mathbb{C}$ l'ensemble *constructeur* tel que $\mathbb{C} = \mathbb{N} \cup \mathbb{D} \cup \mathbb{I} \cup \mathbb{O}$;
- soit $\mathcal{X}$ l'ensemble des éléments des λ-matrices, dénommés *acteurs*, défini plus bas ;
- soit $\mathbb{G}_{Nn} = \{C_1, C_2, \ldots, C_n\}$ un élément de l'ensemble des *acteurs énumérés* tel que $C_1, C_2, \ldots, C_n \notin \mathbb{G}_{Nn}$ [5], $Card(\mathbb{G}_{Nn}) = n$ et tel que ou bien $C_1, C_2, \ldots, C_n \in \mathcal{X}$, ou bien $n = 1$ et $C_1 \in \mathbb{C}$;
- soit $\mathbb{I}_n = \{1, 2, \ldots, n\}$ l'ensemble des *index* de $\mathbb{N}$ jusqu'à n tel que $\mathbb{I}_n \subset \mathbb{N}^*$;
- soit $\mathbb{G}_{NIn}$ le graphe de la famille de $\mathbb{G}_{Nn}$ indexée par $\mathbb{I}_n$ par la relation $\iota_n = \{\mathbb{I}_n, \mathbb{G}_{Nn}, \mathbb{G}_{NIn}\}$. $\mathbb{G}_{NIn}$ est de la forme $\{(1, C_1), (2, C_2), \ldots, (n, C_n)\}$ et on l'écrira $\{C_1, C_2, \ldots, C_n\}$ ou $C_n \star \ldots \star C_2 \star C_1$;
- soit $\mathbb{G}_{NI}$ l'ensemble des *acteurs énumérés et indexés* $\mathbb{G}_{NIn}$ pour tout $n \in \mathbb{N}^*$. On a $\mathbb{G}_{NI} = \cup_{n \in \mathbb{N}^*} \mathbb{G}_{NIn}$;
- soit $\mathbb{T}$ l'ensemble des *types* tel que $\mathbb{T} = \{\top, \bot, \bar{a}, \bar{p}, \bar{d}, \bar{n}, \bar{s}, \bar{o}, \bar{t}, \bar{i}, \bar{v}\}$;
- à chaque élément de $\mathbb{G}_{NI}$, on associe un type de $\mathbb{T}$ pour obtenir les ensembles $\mathbb{G}_{NITt}$ notés $\{C_1, C_2, \ldots, C_n\}_t$;
- on définit $\mathbb{G}_{NIT}$ l'ensemble des *acteurs énumérés, indexés et typés* tel que $\mathbb{G}_{NIT} = \cup_{t \in \mathbb{T}} \mathbb{G}_{NITt}$. On a par définition $\mathcal{X} \subset \mathbb{G}_{NIT}$;
- on définit $\mathcal{A}$ l'ensemble des *atomes* de $\mathcal{X}$ comme un sous-ensemble de $\mathcal{X}$ par l'union des ensembles suivants :

4. Ces opérateurs doivent doivent avoir certaines propriétés, comme nous le verrons par la suite.

5. Cette condition est très importante car c'est elle qui élimine les cycles dans la structure des matrices.

nom	syntaxe	définition
donnée		$\{x \in \mathcal{D} \mid x = \{d\}_{\bar{\imath}}\}$
entier	$1, 2, \ldots$	$\{x \in \mathcal{N} \mid x = \{n\}_{\bar{n}}\}$
indéfini	$\bot$	$\{0\}_{\bot}$
opérateur	$+, -$	$\{x \in \mathcal{O} \mid x = \{o\}_{\bar{o}}\}$
sur-défini	$\top$	$\{0\}_{\top}$
symbole	foo, baz	$\{x \in \mathcal{I} \mid x = \{s\}_{\bar{i}}\}$

pour tout $n \in \mathbb{N}$, $d \in \mathbb{D}$, $s \in \mathbb{I}$ et $o \in \mathbb{O}$. On remarque que les atomes sont des ensembles énumérés, indexés et typés dont les éléments n'appartiennent pas à $\mathcal{X}$;

– on définit les *objets* suivants comme des sous-ensembles de $\mathcal{X}$:

nom	syntaxe	définition
alternative	$c \to a, s$	$\{x \in \mathcal{T} \mid x = \{c, a, s\}_{\bar{a}}\}$
application	$o(a_1, \ldots, a_k)$	$\{x \in \mathcal{P} \mid x = \{o, a_1 \ldots a_k\}_{\bar{p}}\}$
définition	$n := v$	$\{x \in \mathcal{F} \mid x = \{s, v\}_{\bar{d}}\}$
extraction	$v \cdot i$	$\{x \in \mathcal{E} \mid x = \{v, i\}_{\bar{e}}\}$
flot	$e\, f_{by}\, c$	$\{x \in \mathcal{S} \mid x = \{e, c\}_{\bar{s}}\}$
vecteur	$\left[\begin{smallmatrix} a_1 \\ \ldots \\ a_k \end{smallmatrix}\right]$	$\{x \in \mathcal{V} \mid x = \{a_1, \ldots, a_k\}_{\bar{v}}\}$

pour tout $n \in \mathcal{I}$, $a, a_1, a_2, \ldots, a_k, c, e, i, s, v \in \mathcal{X}$, $o \in \mathcal{O}$. Les objets sont des ensembles dont les composantes sont des éléments de $\mathcal{X}$ [6] [7] ;

– on a $\mathcal{X} = \mathcal{A} \cup \mathcal{F} \cup \mathcal{T} \cup \mathcal{S} \cup \mathcal{E} \cup \mathcal{P} \cup \mathcal{V}$, l'ensemble des *acteurs*, termes des λ-matrices (cf. exemple B) ;

– on appelle *système* les éléments de l'ensemble $\mathcal{V}$ de $\mathcal{X}$;

– on appelle *acteur figé* un élément d'un sous-ensemble des acteurs $\mathcal{X}$ noté $\overline{\mathcal{X}} = \mathcal{V} + \mathcal{A} - \mathcal{I}$ composé des vecteur et des atomes sans les symboles, et dont tous les éléments sont des acteurs figés ;

– on appelle *acteur neutre* un élément d'un sous-ensemble de $\mathcal{X}$ noté $\mathbb{X}_n = \mathcal{V} + \mathcal{P} + \mathcal{A} - \mathcal{I}$ composé des vecteurs, des applications et des atomes sans les symboles, et dont tous les éléments sont des acteurs neutres. Les acteur figés sont un sous-ensemble des acteurs neutres.

6. Par convention, les opérateurs associent à droite, ce qui signifie que $a\, f_{by}\, b\, f_{by}\, c$ doit se lire $a\, f_{by}\, (b\, f_{by}\, c)$.

7. Dans ce qui suit, la notation $f(a_1, a_2, \ldots, a_k)$ représente une application et $f(a_1, a_2, \ldots, a_k)$ son résultat. Certain résultats seront notés sous une forme abrégée comme $f_E u$ qui représente $f(u, E)$.

4.3 Applications de bases

Cette définition des termes algébrique permet d'élaborer un premier jeu d'opérateurs, ou applications, de base. Leur caractère primitif provient du fait qu'ils sont issus directement de la structure des termes des λ-matrices.

Pour tout acteur $x = \{c_1, c_2, \ldots, c_k\}_t \in \mathcal{X}, k \in \mathbb{N}^*$, on définit les applications suivantes :

- *Type* : retourne le type d'un acteur.

$$\begin{aligned} \tau() : \mathcal{X} &\longrightarrow \mathbb{T} \quad | \quad | \\ \tau(x) &= t \end{aligned} \tag{5}$$

- *Dimension* : retourne la dimension d'un acteur.

$$\begin{aligned} |\,| : \mathcal{X} &\longrightarrow \mathbb{N} \quad | \quad | \\ |x| &= \begin{cases} 1, \text{ si } x \in \mathcal{A}, \\ \sum_{i=1}^{n} |c_i|\,, \\ \quad \text{si } x = \{c_1, c_2, \ldots, c_n\}_t \notin \mathcal{A}. \end{cases} \end{aligned} \tag{6}$$

  La dimension d'un acteur peut être directement assimilée à la place occupée lorsqu'il sera représenté en mémoire. Pour simplifier, on remarque que la dimension des données vaut 1. En fait, elles doivent avoir une dimension constante. Cette constante devient alors la dimension des atomes.

- *Index* : retourne l'acteur ayant une certaine valeur d'index inclus dans un acteur, ou $\perp$ si l'index est trop élevé.

$$\begin{aligned} \delta(,) : \mathbb{N} \times \mathcal{X} &\longrightarrow \mathcal{X} \quad | \quad | \\ \delta(i, x) &= \begin{cases} \perp, \text{ si } i > n \;\; ou \;\; x \in \mathcal{A}, \\ C_i, \text{ si } i \leq n \;\; et \;\; x \in \mathcal{X} - \mathcal{A}. \end{cases} \end{aligned} \tag{7}$$

D'après les définitions précédentes, on notera que tout élément de $\mathcal{X}$ possède un type et une dimension uniques. La dimension des atomes vaut 1, alors que la dimension des objets est égale à la sommes de la dimension de leur composantes augmentée de 1. De plus, tout objet de $\mathcal{X}$ peut être indéxé : la valeur indexée est elle-même un élément de $\mathcal{X}$.

5 Environnement

D'une manière heuristique, un environnement est une structure permettant de contenir des associations entre un nom et une valeur. Dès lors, partout dans cet environnement, il est possible d'utiliser le nom comme un synonyme de la

valeur. Lorsqu'un environnement est contenu dans un autre, il est appelé « sous-environnement » ou « environnement fils » ; les associations de l'environnement parent sont accessibles dans le sous-environnement, mais l'inverse est faux.

Dès qu'il existe la notion de hiérarchie des environnements, la liaison des données peut être dynamique ou statique. La première est utilisée dans le langage LISP ; elle ne permet pas une compilation efficace [ASU86, FM91, Jon87]. Nous lui préférerons la seconde, utilisée dans le langage SCHEME et ses dérivés [ASS87, CR90] (cf. complément A.1). Dans le cadre de la liaison statique, les variables libres de la valeur associée à un nom sont liées dans l'environnement de la valeur et non dans l'environnement où elle est utilisée.

5.1 Outils

Cette section présente le matériel de base utilisé pour la définition des environnements. Ces outils permettent d'établir des dépendances entre acteurs basées sur l'inclusion, définie ci-dessous.

- *Appartenance* $\succeq$: un acteur appartient à un autre lorsqu'au moins un index permet de l'obtenir.

$$\begin{array}{l} \forall x, y \in \mathcal{X}, \\ x \succeq y \Leftrightarrow \exists k_1, k_2, \ldots, k_n \in \mathbb{N}, n \in \mathbb{N}^* \quad | \\ \delta(k_1, y) = \ldots = \delta(k_n, y) = x. \end{array} \tag{8}$$

- *Inclusion* $\succ$: un acteur est inclus dans un autre lorsqu'il existe au moins une liste d'acteurs en relation d'appartenance permettant de lier le premier au dernier.

$$\begin{array}{l} \forall x, y \in \mathcal{X}, \\ x \succ y \Leftrightarrow \exists z_1, z_2, \ldots, z_n \in \mathcal{X}, n \in \mathbb{N} \quad | \\ x \succeq z_1, z_1 \succeq z_2, \ldots, z_n \succeq y; \end{array} \tag{9}$$

  Par définition, $z_1, z_2, \ldots, z_n$ forment un *chemin* de x dans y. Notons que les chemins sont des éléments des acteurs énumérés et indexés $\mathbb{G}_{NI}$.

Les acteurs ne peuvent créer des cycles dans leur structure, car un acteur ne peut s'appartenir lui-même (cf. preuve C.1).

5.2 Définitions

Les outils de base introduits dans la section précédente permettent maintenant de définir les éléments de l'ensemble des environnements.

1. Dans l'ensemble des chemins $\mathbb{G}_{NI}$, il existe un sous-ensemble d'éléments dont les composantes sont des vecteurs ; cet ensemble Γ est l'ensemble des *environnements* tel que $E = \{v_1, v_2, \ldots, v_k\} \in \Gamma \Rightarrow \forall i \in [1, k], \tau(v_i) = \bar{v}$, qui est le type des vecteurs.

2. Si $z_1, z_2, \ldots, z_n, y$ est le chemin de x dans y et $y \in \mathcal{V}$ et $z_1, z_2, \ldots, z_n \notin \mathcal{V}$ alors y est l'*environnement immédiat* de x dans y.

3. Si $z_1, z_2, \ldots, z_n, y$ est le chemin de x dans y et $y, z_\alpha \in \mathcal{V}$, avec $\alpha \in [1, n]$, et $z_1, z_2, \ldots, z_{\alpha-1}, z_{\alpha+1}, \ldots, z_n \notin \mathcal{V}$ alors $z_\alpha \star y$ est l'*environnement* de x dans y.

4. Si $z_1, z_2, \ldots, z_n, y$ est l'environnement de x dans y, alors $z_1, z_2, \ldots, z_n$ sont des *sous-environnements* de y et y est leur *environnement parent.*

Par abus de notation, il est convenu que l'ensemble des vecteurs est un sous-ensemble des environnements $\mathcal{V} \subset \Gamma$.

5.3 Valeur associée

Étant donné un environnement et un symbole, il est possible de déterminer la valeur associée à ce symbole dans cet environnement. Cette recherche de la valeur associée se fait de manière apparemment récursive. En fait il s'agit d'un processus itératif, dans le sens où la totalité du problème se réduit en la résolution d'un problème plus simple (cf. complément A.2). Voici la définition de cet opérateur :

$$\begin{aligned} &\rho(,) : \mathcal{I} \times \Gamma \longrightarrow \mathcal{X} \quad | \quad | \\ &\quad \rho(s, E\star\begin{bmatrix} C_1 \\ \vdots \\ C_k \end{bmatrix}) \quad \text{(vecteur)} \\ &\qquad = \begin{cases} v_\alpha, \text{ si } c_i = \ s := v_i, \forall i \in [\alpha, \alpha + n], n \in \mathbb{N}, \\ \rho(s, E), \text{ sinon,} \end{cases} \\ &\quad \rho(s, \perp) = \perp. \quad \text{(indéfini)} \end{aligned} \tag{10}$$

On remarque que si un environnement associe plusieurs fois le même symbole, c'est la première définition — celle qui a l'index le plus faible — qui est prise en compte.

Il est intéressant de noter que seule les définitions incluses directement dans les environnements sont prises en compte. Si une définition est située dans un autre objet, elle est correcte dans sa syntaxe car l'objet décrit appartient à $\mathcal{X}$, mais n'a pas de sens dans la mesure où la définition ne sera jamais prise en compte.

La recherche d'une valeur associée à un symbole se termine toujours, soit par la valeur associée, soit par $\perp$ (cf. preuve 7).

De plus, si une variable est liée dans un environnement, elle l'est identiquement dans l'environnement étendu à droite, ce qui s'écrit :

$$\forall E, F \in \Gamma, \forall s \in \mathcal{I}, \rho(s, E) = u \neq \perp \Rightarrow \rho(s, F\star E) = u. \tag{11}$$

Cette propriété est la base de la modularisation des λ-matrices car les variables liées dans un module restent liées aux même valeurs dans un système utilisant ce module.

5.4 Environnement associé

On définit ici un opérateur qui permet de retrouver l'environnement de définition d'un symbole utilisé dans un environnement. Cet opérateur sera utilisé pour effectuer des liaisons statiques plutôt que dynamique (cf. complément A.1) en évaluant les valeurs associées aux symboles dans leur environnement d'origine plutôt que dans l'environnement du symbole. Sa définition est :

$$\begin{aligned}
&\mu(,) : \mathcal{I} \times \Gamma \longrightarrow \Gamma \quad | \qquad | \\
&\quad \mu(s, E\star\begin{bmatrix} C_1 \\ \vdots \\ C_k \end{bmatrix}) \quad \text{(vecteur)} \\
&\quad = \begin{cases} E\star\begin{bmatrix} C_1 \\ \vdots \\ C_k \end{bmatrix}, \text{ si } c_\alpha = \ s{:=}v, \forall \alpha \in [1,k], \\ \mu(s,E), \text{ sinon,} \end{cases} \\
&\quad \mu(s, \perp) = \perp. \quad \text{(indéfini)}
\end{aligned} \tag{12}$$

Pour que les λ-matrices soient à liaison dynamique plutôt que statique, il suffit de changer la définition de cet opérateur en

$$\mu(s, E) = E, \forall s \in \mathcal{I}, \forall E \in \Gamma. \tag{13}$$

Ainsi, l'environnement courant de la valeur associée au symbole s est l'environnement courant de s, ce qui provoque une liaison dynamique.

5.5 Variable libre

On appelle *variable libre* s dans un environnement E toute variable telle que $\rho(s, E) = \perp$.

Une λ-matrice contenant des variables libres est dite *incomplète* (cf. définition 6.5). Une telle matrice est un module destiné à être incorporer dans des systèmes ultérieurement, à des fins de structuration ou de réutilisation. Les variables libres devront alors être définies dans le système hôte, ou dans le cas contraire, l'ensemble forme un autre module (cf. section 7). Seuls les systèmes complets peuvent être résolus.

5.6 Environnement courant

Cette section et la suivantes servent de transition pour introduire les opérateurs dynamiques. Ainsi, les principes qui y sont décrits ne seront pas utilisés ailleurs. Cependant, ces notions sont très importantes.

A tout terme inclus dans un système, on peut associer un environnement dit *environnement courant*. Le résultat de cette association forme un couple dont les éléments appartiennent à un ensemble noté $\mathcal{X}.\Gamma$.

Voici la définition de la relation :

$$\begin{array}{l} \epsilon(,) : \mathcal{X} \times \Gamma \longrightarrow \mathcal{X}.\Gamma \quad | \quad | \\ \quad \epsilon(u, E) = u.E, \forall u \in \mathcal{A}, \quad \text{(atome)} \\ \quad \epsilon(\{c_1, ..., c_k\}_t, E) \\ \quad\quad = \{\epsilon(c_1, E'), ..., \epsilon(c_k, E')\}_t.E, \\ \quad \text{avec } E' = E \star \{c_1, ..., c_k\}_t, \quad \text{(vecteur)} \\ \quad \text{ou } E' = E. \quad \text{(défaut)} \end{array} \tag{14}$$

Comme les acteurs ne forment pas de cycle (cf. définition 4.2), on en déduit que toutes les composantes d'un système donné a un environnement courant unique.

6 Opérateurs dynamiques

De manière à joindre à chaque acteur son environnement courant, la méthode utilisée dans la section précédente est de modifier la structure de chaque acteur pour lui accoler son environnement courant. Ici, une autre méthode va être utilisée.

En effet, plutôt que de modifier une λ-matrice pour adjoindre à chaque terme son environnement courant, le formalisme utilise des opérateurs dits « dynamiques » à deux paramètres : un acteur et son environnement courant. Ce dernier est construit au fur et à mesure : lorsqu'un vecteur est traité, chacune de ses composantes sont traitées dans l'environnement étendu par le vecteur. La définition générique des ces opérateurs est :

$$\begin{array}{l} dyn : \mathcal{X} \times \Gamma \times ... \longrightarrow \mathcal{X} \quad | \quad | \\ \quad dyn(u, E, ...) = f(u, E, ...), \forall u \in \mathcal{A}, \quad \text{(atome)} \\ \quad dyn(\{c_1, ..., c_k\}_t, E, ...) \\ \quad\quad = f(\{dyn(c_1, E', ...), ..., dyn(c_k, E', ...)\}_t, E, ...), \\ \quad \text{avec } E' = E \star \{c_1, ..., c_k\}_t, \quad \text{(vecteur)} \\ \quad \text{ou } E' = E. \quad \text{(défaut)} \end{array} \tag{15}$$

où f est une fonction dépendante de l'opérateur dynamique. Ainsi, dans tous les cas, l'environnement courant est E, le second paramètre de la fonction f.

6.1 Réduction

L'opérateur de réduction réduit les applications. Il est une primitive de l'opérateur d'évaluation décrit dans la section suivante. Sa définition est :

$$\begin{array}{l} \triangleleft : \mathcal{P} \longrightarrow \mathcal{X} \quad | \quad | \\ \quad \triangleleft\, u = \perp, \quad \text{(défaut)} \\ \quad \triangleleft\, o(a_1, a_2, \ldots, a_k) \quad \text{(données)} \\ \quad\quad = o(\, \triangle_E\, ak_1, \triangle_E\, ak_2, \ldots, \triangle_E\, ak_) = x, \\ \quad\quad \forall u \in \mathcal{X}, \\ \quad\quad \forall x \in \mathcal{D} + \{\perp\}. \end{array} \tag{16}$$

L'ensemble de données doit être distinct des autres ensembles : en effet, si cet ensemble était par exemple identique à l'ensemble des symboles, le résultat des réductions risqueraient de corrompre le système.

6.2 Évaluation

L'opérateur d'évaluation peut être vu comme donnant une « valeur » aux termes des λ-matrice. Il prend une « photographie » du système à un instant donné. Il en résulte une perte d'informations par rapport au système initial, qui lui contient des valeurs, et leur mode d'obtention.

La définition de l'opérateur d'évaluation est la suivante :

$$
\begin{aligned}
&\triangle : \mathcal{X} \times \Gamma \longrightarrow \mathcal{X} \quad | \quad | \qquad (17)\\
&\triangle_E\, c \to a, s = \begin{cases} \triangle_E\, a, \text{ si } \triangle_E\, c \neq \bot, \\ \triangle_E\, s, \text{ sinon,} \end{cases} \quad \text{(alternative)}\\
&\triangle_E\, o(a_1,\, a_2,\, \ldots,\, a_k) \quad \text{(application)}\\
&\quad = \triangleleft\, o(a_1,\, a_2,\, \ldots,\, a_k),\\
&\triangle_E\, x = x, \quad \text{(atome)}\\
&\triangle_E\ \ s{:=}v = \triangle_E\, v, \quad \text{(définition)}\\
&\triangle_E\ \ u \cdot i \quad \text{(extraction)}\\
&\quad = \begin{cases} \delta(\triangle_E\, i,\, \triangle_E\, u), \text{ si } \triangle_E\, u \in \mathcal{V} \\ \bot, \text{ sinon,} \end{cases}\\
&\triangle_E\, e\, f_{by}\, c = \triangle_E\, e, \quad \text{(flot)}\\
&\triangle_E\, s = \triangle_{E_v}\, v, \quad \text{(symbole)}\\
&\quad \text{avec } v = \rho(s, E) \text{ et } E_v = \mu(s, E),\\
&\triangle_E \begin{bmatrix} c_1 \\ \vdots \\ c_k \end{bmatrix} = \begin{bmatrix} \triangle_{E'}\, c_1 \\ \vdots \\ \triangle_{E'}\, c_k \end{bmatrix}, \quad \text{(vecteur)}\\
&\quad \text{avec } E' = E \star \begin{bmatrix} c_1 \\ \vdots \\ c_k \end{bmatrix}.
\end{aligned}
$$

Remarquons que l'évaluation d'un symbole dans un environnement peut ne pas se terminer. En effet, un cycle peut être crée. Nous montrons plus bas comment le critère de calculabilité détecte ces cycles (cf. définition 6.5)

Dans le cas de l'évaluation des symboles, le fait que l'évaluation de la valeur associée au symbole se fasse dans l'environnement de définition du symbole E_v et non dans l'environnement E d'utilisation du symbole et la caractéristique qui implique que les λ-matrices sont à liaison statique [ASS87] (cf. complément A.1).

Le codomaine de l'évaluation est l'ensemble des acteurs figés $\overline{\mathcal{X}}$ (cf. preuve 9) :

$$
\begin{aligned}
&\forall u \in \mathcal{X} \quad | \quad \triangle_E\ u \in \mathcal{X},\\
&v \succ \triangle_E\, u \Rightarrow v \in \mathcal{V} + \mathcal{A} - \mathcal{I} = \overline{\mathcal{X}}.
\end{aligned} \qquad (18)
$$

Les $k^{\text{ièmes}}$ évalués[8] d'un acteur figés sont identiques (cf. preuve 10), ce qui s'énonce :

$$\forall u \in \overline{\mathcal{X}} \Rightarrow u = \triangle_\bot\, u. \qquad (19)$$

8. Les expressions $f^k x$ et $(f)^k x$ signifie que l'opérateur f est appliqué k fois à l'argument x.

6.3 Régénération

L'opérateur de régénération construit un nouveau système dont l'état des flots à été mis à jour. Par analogie avec le modèle d'état (cf. section 2), il correspond au calcul global de toutes les variables d'état pour passer à l'instant suivant. Toutes les composantes sont régénérées une à une.

$$
\begin{array}{l}
\bigtriangledown_{:} \mathcal{X} \times \Gamma \longrightarrow \mathcal{X} \quad | \quad | \\
\quad \bigtriangledown_{E} x = x, \quad \text{(atome)} \\
\quad \bigtriangledown_{E} e\, f_{by}\, c = \bigtriangleup_{E} c\, f_{by}\ \bigtriangledown_{E}\ c, \quad \text{(flot)} \\
\quad \bigtriangledown_{E} \{c_1, c_2, ..., c_n\}_t \quad \text{(objet)} \\
\quad\quad = \{\bigtriangledown_{E} c_1,\ \bigtriangledown_{E} c_2, ...,\ \bigtriangledown_{E} c_n\}_t, \\
\quad \bigtriangledown_{E} \left[\begin{array}{c} c_1 \\ \cdots \\ c_k \end{array}\right] = \left[\begin{array}{c} \bigtriangledown_{E'} c_1 \\ \cdots \\ \bigtriangledown_{E'} c_k \end{array}\right], \quad \text{(vecteur)} \\
\quad\quad \text{avec } E' = E \star \left[\begin{array}{c} c_1 \\ \cdots \\ c_k \end{array}\right].
\end{array}
\tag{20}
$$

Remarquons que la régénération d'un flot produit un flot dont l'état est l'évalué du contrat et dont le contrat est son régénéré. Ainsi, la régénération d'un acteur produit un acteur de même structure qui diffère seulement par la valeur des états des flots.

L'évaluation du contrat dans la régénération du flot permet de placer une valeur contemporaine au pas de régénération, et donc de réaliser un retard unitaire. La régénération d'un flot pourrait être

$$
\bigtriangledown_{E} e\, f_{by}\, c = c\, f_{by}\ \bigtriangledown_{E}\ c, \tag{21}
$$

qui réalise alors un simple changement d'expression du flot.

On appelle l'environnement régénéré de E noté $\overset{\bigtriangledown}{E}$ l'environnement défini par :

$$
\begin{array}{l}
\forall E \in \Gamma \quad | \quad E = e_n \star \ldots \star e_2 \star e_1 = e_{1...n}, \\
\overset{\bigtriangledown}{E} = \overset{\bigtriangledown}{e}_n \star \ldots \star \overset{\bigtriangledown}{e}_2 \star \overset{\bigtriangledown}{e}_1 = \overset{\bigtriangledown}{e}_{1...n} \quad | \\
\overset{\bigtriangledown}{e}_\alpha = \bigtriangledown_{e_{1...\alpha-1}} e_\alpha, \forall \alpha \in [1...n].
\end{array}
\tag{22}
$$

La valeur associée à un symbole dans un environnement régénéré est égale au régénéré de cette valeur dans l'environnement d'origine (cf. preuve 8), ce qui s'énonce :

$$
\begin{array}{l}
\forall E \in \Gamma, \\
\left.\begin{array}{r} \mu(s, E) = E_v \neq \perp \\ \rho(s, E) = v \neq \perp \end{array}\right\} \Rightarrow \left\{\begin{array}{l} \mu(s, \overset{\bigtriangledown}{E}) = \overset{\bigtriangledown}{E}_v, \\ \rho(s, \overset{\bigtriangledown}{E}) = \bigtriangledown_{E_v} v. \end{array}\right.
\end{array}
\tag{23}
$$

L'environnement régénéré permet de considérer l'environnement d'un acteur aprés une régénération global du système.

6.4 Résolution

L'opérateur de résolution permet l'écriture d'une équation de point-fixe dans le formalisme des λ-matrice. Il régénère un système tant qu'une condition limite n'est pas atteinte. La durée de ce processus est inconnue et elle peut être infinie. Ce processus n'est donc pas borné en temps. Par contre, si le système est bornée en temps, chaque étapes de la résolution l'est aussi. La définition de cet opérateur est :

$$\begin{aligned} \triangleright &: \mathcal{X} \times \mathcal{N} \times \Gamma \longrightarrow \mathcal{X} \quad | \quad | \\ &\triangleright_{i\,E} s = \triangle_E \ s \cdot i \stackrel{?}{=} \perp \rightarrow \perp, \triangleright_{\substack{\triangledown \\ i\ E}} \ \triangledown_E s. \end{aligned} \tag{24}$$

L'évaluation du système est forcée dans l'expression de la condition de l'alternative, réalisant ainsi les sorties. La définition de cet opérateur est écrit de manière à être un acteur des λ-matrices. Il suppose que les opérateurs de résolution $\triangleright$ et d'évaluation $\triangle$ sont des éléments des opérateur spéciaux, donc accessibles au concepteur du système.

L'opérateur de résolution engendre un processus itératif, stable et non borné en temps (cf. complément A.2).

6.5 Complétude

La complétude vérifie qu'à tout symbole appartenant à un système, autre que le nom d'une définition, correspond une valeur dans son environnement courant (cf. preuve 11), ce qui s'énonce :

$$\begin{aligned} &\forall u \in \mathcal{X}, \Lambda_l(u, \perp) = 1 \\ &\Rightarrow \forall s \in \mathcal{I} \quad | \quad s \succ u, s \notin \mathcal{F}, \rho(s, E) \neq \perp. \end{aligned} \tag{25}$$

où E est l'environnement courant de s dans u.

Pour détecter l'absence de définition, nous parcourons l'ensemble des composantes d'un système en mettant à jour l'environnement dynamique qui leur est associé. Nous recherchons les symboles qui se trouvent ailleurs que dans une définition ; si la recherche de la valeur associée à ce symbole échoue, c'est à dire qu'il s'agit d'une variable libre, le critère est annulé.

La définition du critère est donc :

$$\Lambda_l(,) : \mathcal{X} \times \Gamma \longrightarrow \{0, 1\} \quad | \quad | \tag{26}$$

$$
\begin{array}{l}
\Lambda_l(x,E) = 1, \quad \text{(atome)} \\
\Lambda_l(\ s{:=}v, E) = \Lambda_l(v,E), \quad \text{(définition)} \\
\Lambda_l(\{c_1,c_2,...,c_n\}_t, E) = \prod_{i=1}^{n} \Lambda_l(c_i,E), \quad \text{(objet)} \\
\Lambda_l(s,E) = \left\{ \begin{array}{l} 0, \text{ si } \rho(s,E) = \bot, \\ 1, \text{ sinon,} \end{array} \right. \quad \text{(symbole)} \\
\Lambda_l(\left[\begin{array}{c} c_1 \\ \vdots \\ c_k \end{array} \right], E) = \prod_{i=1}^{k} \Lambda_l(c_i,E'), \quad \text{(vecteur)} \\
\quad \text{avec } E' = E\star\left[\begin{array}{c} c_1 \\ \vdots \\ c_k \end{array} \right].
\end{array}
$$

Pour tout acteur complet dans un environnement, si son évalué existe[9], il est complet (cf. preuve 19), et que si son régénéré existe, ils est complet (cf. preuve 12), ce qui s'énonce :

$$
\begin{array}{l}
\forall u \in \mathcal{X} \quad | \quad \exists!\ \triangle_E\ u \in \mathcal{X}, \exists!\ \bigtriangledown_E\ u \in \mathcal{X}, \\
\Lambda_l(u,E) = 1 \Rightarrow \left\{ \begin{array}{l} \Lambda_l(\ \triangle_E\ u, E) = 1, \\ \Lambda_l(\ \bigtriangledown_E\ u, \overset{\bigtriangledown}{E}) = 1. \end{array} \right.
\end{array} \tag{27}
$$

Nous en déduisons que la complétude se conserve au cours des régénérations.

6.6 Détecteur de cycle

Nous présentons ici un opérateur à valeur booléenne permettant de savoir si un cycle symbolique est créé dans un acteur sur un symbole dans un environnement (cf. complément A.3.3). Cet opérateur est une primitive du critère de calculabilité décrit dans la section suivante.

Nous avons :

$$
\begin{array}{l}
\gamma(,,) : \mathcal{I} \times \mathcal{X} \times \Gamma \longrightarrow \{0,1\} \quad | \quad | \\
\quad \gamma(s,x,E) = 1, \quad \text{(atome)} \\
\quad \gamma(s,\ s'{:=}v, E) = \gamma(s,v,E), \quad \text{(définition)} \\
\quad \gamma(s, e\ f_{by}\ c, E) = \gamma(s,e,E), \quad \text{(flot)} \\
\quad \gamma(s,\{c_1,c_2,\ldots,c_n\}_t, E) = \prod_{i=1}^{n} \gamma(s,c_i,E), \quad \text{(objet)} \\
\quad \gamma(s,s',E) \quad \text{(symbole)} \\
\qquad = \left\{ \begin{array}{l} 0, \text{ si } s' = s, \\ \gamma(s,\rho(s',E),\mu(s',E)), \text{ sinon,} \end{array} \right. \\
\quad \gamma(s,\left[\begin{array}{c} c_1 \\ \vdots \\ c_n \end{array} \right], E) \quad \text{(vecteur)} \\
\qquad = \left\{ \begin{array}{l} \prod_{i=1}^{n} \gamma(s,c_i,E'), \text{ si } \rho(s,\left[\begin{array}{c} c_1 \\ \vdots \\ c_n \end{array} \right]) = \bot, \\ 1, \text{ sinon,} \end{array} \right. \\
\qquad \text{avec } E' = E\star\left[\begin{array}{c} c_1 \\ \vdots \\ c_n \end{array} \right].
\end{array} \tag{28}
$$

9. La clause d'existence est nécessaire car un acteur complet n'est pas forcément calculable, ce qui signifie que son évalué peut ne pas exister, et donc son régénéré non plus.

Aucun évalué s'il existe ne crée de cycle sur un symbole dans un environnement (cf. preuve 13), et que si aucun cycle n'existe sur un symbole dans l'acteur d'un environnement, alors aucun cycle n'existe non plus sur ce symbole dans le régénéré de l'acteur dans l'environnement régénéré (cf. preuve 14).

6.7 Calculabilité

Une λ-matrice est calculable lorsqu'elle est complète et qu'elle ne contient pas d'équation de point-fixe. Une équation de point-fixe est créée lorsque la valeur de la définition d'un symbole utilise ce symbole, comme dans $x := f(x)$ (cf. exemple B.3.2), formant ainsi un cycle symbolique. Les flots, cependant, peuvent créer des cycles symboliques avec leur contrat, décrivant ainsi une équation récurrente. Autrement dit, l'évalué d'une λ-matrice calculable existe (cf. preuve 15), ce qui s'énonce :

$$\forall u \in \mathcal{X}, \Lambda_c(u, E) = 1 \Rightarrow \exists!\ \triangle_E\ u \in \mathcal{X}. \tag{29}$$

La définition de ce critère s'appuie sur l'opérateur de détection de cycle vu précédemment, rendant sa définition particulièrement simple :

$$\begin{array}{l} \Lambda_c(,) : \mathcal{X} \times \Gamma \longrightarrow \{0,1\} \quad | \qquad | \\ \quad \Lambda_c(x, E) = 1, \quad \text{(atome)} \\ \quad \Lambda_c(\ s{:=}v, E) \quad \text{(définition)} \\ \qquad = \left\{ \begin{array}{l} \Lambda_c(v, E), \text{ si } \gamma(s, v, E) = 1, \\ 0, \text{ sinon,} \end{array} \right. \\ \quad \Lambda_c(\{c_1, c_2, ..., c_n\}_t, E) = \prod_{i=1}^{n} \Lambda_c(c_i, E), \quad \text{(objet)} \\ \quad \Lambda_c(\left[\begin{array}{c} c_1 \\ \vdots \\ c_k \end{array} \right], E) = \prod_{i=1}^{k} \Lambda_c(c_i, E'), \quad \text{(vecteur)} \\ \qquad \text{avec } E' = E \star \left[\begin{array}{c} c_1 \\ \vdots \\ c_k \end{array} \right]. \end{array} \tag{30}$$

Pour tout acteur calculable, son évalué est calculable (cf. preuve 19) ainsi que son régénéré (cf. preuve 16), ce qui s'énonce :

$$\forall u \in \mathcal{X}, \Lambda_c(u, E) = 1 \Rightarrow \left\{ \begin{array}{l} \Lambda_c(\ \triangle_E\ u, E) = 1, \\ \Lambda_c(\ \bigtriangledown_E\ u, \overset{\bigtriangledown}{E}) = 1. \end{array} \right. \tag{31}$$

Nous en déduisons que la calculabilité se conserve au cours des régénérations.

6.8 Norme

La norme d'un acteur s'apparente à sa dimension. On peut l'assimiler à la phrase « *quelle sera la dimension maximale de l'évalué de ...* » . Par définition, la norme d'un acteur non calculable dans un environnement est infinie. Voici sa

définition :

$$
\begin{array}{l}
\sigma(,) : \mathcal{X} \times \Gamma \longrightarrow \mathbb{N} \quad | \quad | \\
\quad \sigma(c \rightarrow a, s, E) = \sup(\sigma(a, E), \sigma(s, E)), \quad \text{(alternative)} \\
\quad \sigma(o(a_1,\, a_2,\, \ldots,\, a_k), E) = 1, \quad \text{(application)} \\
\quad \sigma(x, E) = 1, \quad \text{(atome)} \\
\quad \sigma(\ s{:=}v, E) = \sigma(v, E), \quad \text{(définition)} \\
\quad \sigma(\ u \cdot i, E) \quad \text{(extraction)} \\
\qquad = \left\{ \begin{array}{l} \left\{ \begin{array}{l} \sup\limits_{i=1}^{n}(\sigma(c_i, E')), \text{ si } i \notin \mathcal{N}, \\ \quad \sigma(\delta(i, m), E'), \text{ sinon,} \end{array} \right\} \text{,si } u = \left[\begin{array}{c} c_1 \\ \vdots \\ c_k \end{array} \right], \\ \sigma(\ \rho(u, E) \cdot i, \mu(u, E)), \text{ si } u \in \mathcal{I}, \\ \perp, \text{ sinon,} \end{array} \right. \\
\quad \sigma(e\, f_{by}\, c, E) = \sigma(e, E), \quad \text{(flot)} \\
\quad \sigma(s, E) = \sigma(v, E_v), \quad \text{(symbole)} \\
\qquad \text{avec } v = \rho(s, E) \text{ et } E_v = \mu(s, E), \\
\quad \sigma(\left[\begin{array}{c} c_1 \\ \vdots \\ c_k \end{array} \right], E) = \sum\limits_{i=1}^{k} \sigma(c_i, E'), \quad \text{(vecteur)} \\
\qquad \text{avec } E' = E \star \left[\begin{array}{c} c_1 \\ \vdots \\ c_k \end{array} \right].
\end{array}
\tag{32}
$$

La norme d'une alternative est la plus grande norme de la clause « si » et de la clause « sinon ». Ainsi, la norme d'une alternative est toujours supérieure au sens large à la dimension de son évalué.

La norme d'une extraction est plus complexe. En effet, il faut d'abord considérer le fait que seul l'extraction d'un vecteur ou d'un symbole représentant en final un vecteur est sémantiquent corecte. Si le module de l'extraction est un symbole, on remplace le symbole par sa valeur. Si le module n'est ni un symbole ni un vecteur, la norme de l'extraction n'est pas définie.

Si le module est un vecteur, l'extraction peut être déterministe, c'est le cas ou l'index est un entier ou indéterministe dans les autres cas. Si l'extraction est déterministe, sa norme est la norme de la composante dans son environnement. Si elle ne l'est pas, sa norme est la plus grande norme de toutes les composantes du vecteur. Cette propriété est utilisée pour la définition du critère de la stabilité (cf. section 6.9).

6.9 Stabilité

Un système est stable lorsqu'il est calculable et que la dimension de son régénéré est inférieure à sa dimension (cf. preuve 24) :

$$
\begin{array}{l}
\forall u \in \mathcal{X}, \forall E \in \Gamma, \\
\Lambda_s(u, E) = 1 \Rightarrow |u| \geq |\bigtriangledown_E u| .
\end{array}
\tag{33}
$$

En règle générale, un acteur est stable lorsque toutes ses composantes sont stables. Pour les flots, la condition supplémentaire est que la norme de son état

doive être égale à la norme de son contrat et que son état soit un acteur figé. Ainsi, la dimension du flot au cours de régénération diminue.

Les alternatives doivent avoir des clauses appartenant aux acteurs neutres et dont les normes sont identiques. De ce fait, leur évaluation produit toujours un acteur de même dimension, quelquesoit la valeur de la condition. Cette propriété se conserve au cours de régénérations.

Pour que les extractions produisent un résultat déterministe, la condition est que leur valeur d'index soit un entier. Alors se sera toujours la même composant de l'indéxé qui sera sélectionnée.

Nous obtenons donc la définition :

$$
\begin{array}{l}
\Lambda_s(,) : \mathcal{X} \times \Gamma \longrightarrow \{0,1\} \quad | \quad | \\
\quad \Lambda_s(c \rightarrow a, s, E) \quad \text{(alternative)} \\
\quad\quad = \left\{ \begin{array}{l} 1, \text{ si } a, s \in \mathbb{X}_n \text{ et } \sigma(a,E) = \sigma(s,E), \\ 0, \text{ sinon,} \end{array} \right. \\
\quad \Lambda_s(x, E) = 1. \quad \text{(atome)} \\
\quad \Lambda_s(\ u \cdot i, E) = \left\{ \begin{array}{l} \Lambda_s(u,E), \text{ si } i \in \mathcal{N}, \\ 0, \text{ sinon,} \end{array} \right. \quad \text{(extraction)} \\
\quad \Lambda_s(e\, f_{by}\, c, E), \quad \text{(flot)} \\
\quad\quad = \left\{ \begin{array}{l} \Lambda_s(c,E), \text{ si } e \in \overline{\mathcal{X}} \text{ et } \sigma(e,E) \geq \sigma(c,E), \\ 0, \text{ sinon,} \end{array} \right. \\
\quad \Lambda_s(\{c_1, c_2, \ldots, c_n\}_t, E) = \prod_{i=1}^{n} \Lambda_s(c_i, E), \quad \text{(objet)} \\
\quad \Lambda_s(\left[\begin{array}{c} c_1 \\ \vdots \\ c_n \end{array} \right], E) = \prod_{i=1}^{n} \Lambda_s(c_i, E'), \quad \text{(vecteur)} \\
\quad\quad \text{avec } E' = E \star \left[\begin{array}{c} c_1 \\ \vdots \\ c_n \end{array} \right].
\end{array}
\tag{34}
$$

Pour tout acteur stable dans un environnement, son régénéré est stable dans l'environnement régénéré (cf. preuve 26), ce qui s'énonce :

$$
\forall u \in \mathcal{X}, \Lambda_s(u,E) = 1 \Rightarrow \left\{ \begin{array}{l} \Lambda_s(\ \triangle_E\, u, E) = 1, \\ \Lambda_s(\ \bigtriangledown_E\, u, \overset{\bigtriangledown}{E}) = 1. \end{array} \right. \tag{35}
$$

Nous en déduisons que la stabilité conserve au cours des régénérations.

6.10 Propriétés

Ces propriétés sont vraies pour tous les critères.

- si un critère est vérifié dans un environnement, il l'est aussi dans cet environnement étendu à gauche :

$$
\Lambda(u,E) = 1 \Rightarrow \Lambda(u, F \star E, \tag{36}
$$

)

– si un critère est vérifié pour un système, il l'est pour son évalué [10] :

$$\Lambda(u, E) = 1 \Rightarrow \Lambda(\ \triangle_E\ u, E) = 1 \tag{37}$$

– si un critère est vérifié pour un système, il l'est pour son régénéré :

$$\Lambda(u, E) = 1 \Rightarrow \Lambda(\ \bigtriangledown_E\ u, E) = 1 \tag{38}$$

– si un critère est vérifié dans l'environnement vide, il l'est pour tous ses régénérés dans le même environnement :

$$\Lambda(u, \perp, =)1 \Rightarrow \Lambda(\ \bigtriangledown_\perp\ u, \perp, \tag{39}$$

)

Ainsi, si la description d'un système vérifie un critère, sa résolution le vérifie aussi. Par exemple, si un système est stable, sa résolution l'est aussi, ce qui signifie que le processus de calcul engendré par le système a une occupation en mémoire constante. Il en va de même pour tous les critères.

7 Construction des systèmes

Les λ-matrices sont un outil puissant de structuration des systèmes. En effet, elles permettent de concevoir des modules séparément, et de les rassembler par la suite de manière à former un système complet et autonome.

Un module peut être vue comme une λ-matrice indépendante connectée à d'autres dans un environnement de travail. La connection mise en œuvre permet d'établir deux sortes de liens :

– Les liens allant de l'« extérieur vers l'intérieur » du module. Ils sont assimilables aux *paramètres* des fonctions des langages traditionnels. Les λ-matrices ne possède pas quant à elles un tel mécanisme d'abstraction [11]. Ce type de liens est donc établi par la présence de variables libres dans un module, qui trouveront leur valeur dans le système qui les intégrera.

– Les liens allant de l'« intérieur vers l'extérieur » du module. Ils correspondent à la *valeur de retour* des fonctions. Dans les langages traditionnels, une fonction ne possède qu'une seule valeur de retour. Cette contrainte est trop limitative dans le cadre d'utilisation des λ-matrices, destinées à représenter, entre autre, des circuits numériques aux bus parallèles. Cette connection de l'intérieur d'un module vers l'extérieur est établie, dans les λ-matrice, par l'objet d' « extraction ». Ainsi, il est possible de concevoir un module ayant plusieurs sorties.

10. Il faut rajouter la clause d'existence pour le critère de complétude.

11. L'abstraction des λ-matrices est l'un des objets du langage λ-flot qui en est sur-couche syntaxique.

Un module possède en général des variables libres destinées à établir une connection rentrante. Il est donc utile de concevoir une matrice d'interface permettant le changement de nom de ces variables libres, et ceci afin de permettre d'utiliser plusieurs fois le même module dans un même environnement. La mise en place de cette interface de connection est l'objet du langage λ-flot, qui n'est pas décrit ici.

Possédant des variables libres, un module n'est pas complet, ce qui entraîne qu'il n'est ni calculable ni stable ni mesurable. Il est donc aussi utile de modifier le critère de complétude de manière à ce qu'il tienne compte de la possibilité de paramètrer d'une matrice ; il fera alors une différence entre un paramètre et une variable libre. Seules ces dernières annulent alors la complétude. Dès lors, il est possible de calculer les critère d'un module séparément du système hôte.

8 Vers une architecture matérielle

L'implantation sur une architecture matérielle des λ-matrices est grandement facilitée car le formalisme est en fait assez proche des architectures. Cependant, pour une implantation efficace, il sera nécessaire d'apporter un certains nombre d'optimisations :

- *compilation symbolique* visant à remplacer les symboles par des adresses en mémoire.
- *régénération partielle* ne modifiant que l'état des flots sans régénérer tous le reste ;
- *évaluation paresseuse* par l'implantation d'un indicateur d'évaluation pour chaque objet ;

De plus le parallélisme apparaît naturellement : entre deux régénérations, tous les calculs peuvent être effectués en parallèle. La structure même des λ-matrices permet d'éviter les conflits d'accès aux ressources. Le seul risque est de calculer plusieurs fois la même chose, inconvénient partiellement contourné par l'indicateur de calcul effectué.

Selon les caractéristiques de la matrice, mises en évidences par les critères, l'implantation matérielle sera différente. Par exemple, si la matrice n'est pas stable en mémoire, il faudra prévoir une gestion de la mémoire dynamique de types *glanage de cellules* souvent implantés pour les langages de la famille de LISP. Par contre, si elle est stable, un gestion statique pourra être envisagée sur une quantité de mémoire optimale connue.

9 Conclusion

Les outils dérivés du λ-calcul, comme la sémantique dénotationnelle, sont souvent trop puissants dans le cadre de l'objectif à atteindre ici. En effet, les appli-

cations reposant sur les modèles d'états, et plus particulièrement les applications de traitement de signal, sont très homogènes quant à leur description et à leur fonctionnement. L'outil proposé dans ces pages à pour but de limiter la puissance des outils cités, en offrant une sur-couche bien spécifique.

Cependant, ces limitations ne grèvent en rien la capacité à décrire des applications complexes et modulaires. La modularisation permet d'envisager une compilation séparée et une édition des liens finale des modules, un peu à la manière des langages actuels, dont on connaît l'importance qualitative qu'ils ont eu en Génie Logiciel.

Mais l'apport principal est la possibilité de trier les applications par classes, selon leur caractéristiques. Ces caractéristiques permettent de déterminer le comportement de l'application au cours de son fonctionnement, et donc de choisir une implantation adaptée. Les critères clefs sont la consommation en mémoire et en temps de calcul.

Dérivées d'outils proches des mathématiques appliquées, les λ-matrices reposent intégralement sur un système de preuves formelles. Ce système s'appuie sur une description algébrique de leurs constituants et sur la description fonctionnelle des opérateurs agissants sur ces constituant.

Une autre voie importante à examiner est l'exploitation du parallélisme, qui est clairement identifié et séparé des actions séquentielles. Entre deux régénérations, tous les calculs peuvent être fait en parallèle avec des propriètés interressantes : les interblocages sont impossibles et les opérations d'écritures et de lecture dans la mémoire peuvent être séparées.

Références

[AB82] M. Auguin and F. Boéri. Efficient multiprocessor architecture for digital processing. *ICASSP*, pages 675–679, 1982.

[AB86] M. Auguin and F. Boéri. The opsila computer. *in Parallel Algorithms and architectures, Eds. M. Cosnard et coll., North Holland*, pages 143–154, 1986.

[ASS87] H. Abelson, G.J. Susman, and J. Susman. *Structure and Interpretation of Computer Programs.* MIT Press, 1987.

[ASU86] A. Aho, R. Sethi, and J. Ullman. *Compilers.* Addison-Wesley Publishing Company, Inc, 1986.

[AW76] E.A. Ashcroft and W.W. Wadge. LUCID, a formal system for writing and proving programs. *SIAM j. Comput*, 5(3):336–354, September 1976.

[AW77] E. A. Ashcroft and W. W. Wadge. Lucid, a Nonprocedural Language with Iteration. *j-CACM*, 20(7):519–526, July 1977.

[Bac78] J. Backus. Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *j-CACM*, 21(8):613–641, aug 1978.

[BL90] A. Benveniste and P. LeGuernic. Hybrid dynamical systems theory and the SIGNAL language. *IEEE Transactions*, 35:535–546, 1990.

[Cha92] R. Chassaing. *Digital Signal Processing with C and the TMS320C30.* Topics in Digital Signal Processing, 1992.

[Chu51] A. Church. *The Calculi of Lambda-conversion.* Annals of Mathematical Studies, vol. 6, Princeton University Press, Princeton (N.J.), 1951.

[CR90] W. Clinger and J. Rees. Revised^4 report on the algoritmic language scheme. Technical report, MIT Artificial Intelligence Laboratory, CSDTR 174, october 1990.

[CT93] M. Cosnard and D. Trystram. *Algorithmes et architectures parallèles.* InterÉdition, 1993.

[FM91] P. Fradet and D. Le Métayer. Compilation of functional languages by program transformation. In *Transaction on Programming Languages and Systems*, volume 13,1, pages 21–51. ACM, 1991.

[HCRP91a] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. Programmation et vérification des systèmes réactifs : le langage LUSTRE. *Techniques et Sciences Informatiques*, 10(2):139–158, 1991.

[HCRP91b] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language LUSTRE. In *Proceedings of the IEEE*, pages 1305–1319, 1991. Published as Proceedings of the IEEE, volume 79, number 9.

[Ive62] K. Iverson. *A Programming Language.* Wiley, New York, 1962.

[Jon87] S.L. Peyton Jones. *The implementation of Functional Programming Languages.* Prentice Hall International, 1987.

[Kah74] G. Kahn. The semantics of a simple language for parallel programming. In *IFIP 74 Congress.* North Holland, Amsterdam, 1974.

[Kri90] J.L. Krivine. *Lambda-Calcul, types et modèles.* Masson, 1990.

[Kun80] M. Kunt. *Traité d'Électricité-Traitements numérique des signaux.* Edition Georgi, 1980.

[Lan66] P.J. Landin. The next 700 programming languages. *Communication of ACM*, 9:157–166, march 1966.

[LB90] E.A. Lee and J.C. Bier. Architectures for statically scheduled dataflow. *Journal of parallel and Distributed Computing*, 10:333–348, 1990.

[Lee91] E.A. Lee. Consistency in dataflow graphs. *IEEE Transactions on Parallel and Distributed systems*, 2(2):223–235, April 1991.

[Llo86] A. Lloyd. *A pratical introduction to denotational semantics.* Cambridge Computer Science Texts 23, 1986.

[LM87] E.A. Lee and D.G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, September 1987.

[Mey92] B. Meyer. *Introduction à la théorie des langages de programmation.* Inter Edition, 1992.

[Miq85] R. Miquel. *Le Filtrage Numérique par microprocesseurs.* édiTESTS, 1985.

[NR85] M. Nivat and J.C. Reynolds. *Algebraic methods in semantics.* Cambridge University Press, 1985.

[Rev88] G.E. Revesz. *Lambda-Calculus, Combinators, and Functional Programming.* Cambrige University Press, 1988.

[Sto77] J.E. Stoy. *Denotational Semantics: The Scott — Strachey Approach to Programming Language Semantics.* MIT Press Series in Computer Science, 1977.

[Str66] C. Strachey. *"Towards a Formal Semantics", Formal Language Description Languages for Computer Programming.* éd. Tom B. Steel Jr., pp. 198-220, North-Holland Puplishing Co., Amsterdam, 1966.

[WA85] W.W. Wadge and E.A. Ashcroft. *Lucid, the DataFlow Programming Language.* Academic Press, 1985.

— ANNEXES —

A Éléments de construction

A.1 Liaison statique *vs.* liaison dynamique

Cette section montre la différence entre la liaison statique et la liaison dynamique dans les environnements des λ-matrices sur un exemple. Considérons le système S :

$$\left[\begin{array}{l} a{:=}3 \\ b{:=}a \\ \left[\begin{array}{l} a{:=}4 \\ c{:=}{+}(a,\, b) \end{array}\right] \end{array}\right]$$

Dans le sous-environnement de S, le symbole a de l'application est synonyme de 3 ou de 4, selon la nature de la liaison. Évaluons ce système :

$$\begin{aligned} \triangle_{\perp} S &= \left[\begin{array}{l} \triangle_S\ a{:=}3 \\ \triangle_S\ b{:=}a \\ \triangle_S \left[\begin{array}{l} a{:=}4 \\ c{:=}{+}(a,\, b) \end{array}\right] \end{array}\right] \\ &= \left[\begin{array}{l} \triangle_S\, 3 \\ \triangle_S\, a \\ \left[\begin{array}{l} \triangle_{S'}\ a{:=}4 \\ \triangle_{S'}\ c{:=}{+}(a,\, b) \end{array}\right] \end{array}\right], \\ &\quad \textit{avec}\ \ S' = S\star\left[\begin{array}{l} a{:=}4 \\ c{:=}{+}(a,\, b) \end{array}\right] \\ &= \left[\begin{array}{l} 3 \\ \triangle_S\, a \\ \left[\begin{array}{l} \triangle_{S'}\, 4 \\ {+}(\triangle_{S'}\, a,\ \triangle_{S'}\, b) \end{array}\right] \end{array}\right] \end{aligned}$$

Par définition, l'évaluation d'un symbole est $\triangle_E\, s = \triangle_{E_v}\, v$, avec $v = \rho(s, E)$ et $E_v = \mu(s, E)$.

L'expression $\triangle_S\, a$ vaut sans ambiguïté 3 qui est la valeur associée à a dans l'environnement S. De même, pour $\triangle_{S'}\, a$ on obtient 4 qui la valeur de a dans S'.

Dans le cas de $\triangle_{S'}\, b$, nous obtenons comme valeur associée le symbole a. Donc $\triangle_{S'}\, b = \triangle_E\, a$. Si l'environnement E est identique à S', nous aurions comme

résultat final la valeur 4 qui est la valeur associée à a dans S'. Si E est égale à S, nous avons comme résultat final la valeur 3 qui est la valeur associée à a dans l'environnement S.

Dans le premier cas, nous avons une liaison dynamique, c'est à dire que l'évaluation d'un symbole est l'évaluation de sa valeur associée dans l'environnement de ce symbole. Par contre, dans le second cas, nous avons une liaison statique, c'est à dire que l'évaluation d'un symbole revient à l'évaluation de sa valeur associée dans l'environnement de définition de ce symbole.

Dans les λ-matrices, E vaut $\mu(s, S')$ qui est S. Nous avons donc une liaison statique.

Une des différences importante entre les deux types de liaison provient de l'unicité de l'environnement de définition et de la multiplicité de l'environnement d'utilisation. Dans le premier cas, l'unicité permet une compilation qui par définition est statique, alors que dans le second cas, la multiplicité impose à la compilation de conserver ce caractère multiple, et donc d'être moins efficace.

A.2 Fonction récursives *vs.* fonctions itératives

Nous montrons comment on peut formaliser les fonctions récursives et itératives et quelles sont les conséquences sur la consommation en mémoire des processus respectifs engendrés.

Une fonction récursive est une fonction dont l'évaluation du résultat est une fonction de cette fonction, comme dans :

$$f(x) = g(h(x), f(i(x)))$$

Par exemple, le célèbre exemple de la factorielle donne :

$$n! = n \stackrel{?}{=} 0 \rightarrow 1, n.(n-1)!$$

dans lequel la fonction g joue le double rôle de condition d'arrêt et de multiplieur.

Remarquons que la fonction i transforme x, sans quoi nous aurions une équation de point-fixe de type $x = f(x)$. La fonction h transforme x et le garde en réserve pour l'intégration effectuée par la fonction g.

Le problème inhérent à l'évaluation de fonction récursive est la consommation indéterministe de ressource. En effet, le fait que le « problème » $f(x)$ se traduit en un sous-problème $g(\alpha, f(\beta))$ oblige la conservation de la valeur α. Dans les cas simples, il existe des moyens de convertir une définition de fonction récursive en fonction itérative, comme on peur le voir dans [ASS87]. Mais ces moyens ne peuvent être appliqués qu'au coup par coup, selon le problème à traiter.

La définition d'une fonction itérative repose sur le modèle :

$$f(x) = f(i(x))$$

Dans ce modèle, le problème $f(x)$ se résume en un sous-problème $f(\alpha)$. Ce processus n'est toujours pas déterministe en temps, mais il l'est en consommation de ressource. Par exemple, en reprenant la définition de la factorielle, on obtient :

$$n! = f(n, 1)$$

avec la fonction itérative :

$$f(n, r) = n \stackrel{?}{=} 0 \rightarrow r, f(n - 1, r * n)$$

A.3 Dépendances symboliques

Dans cette section nous décrivons un outil graphique pour représenter les liens symboliques entre les différents éléments d'un système. Cet outil a été utilisé pour concevoir le détecteur de cycles (cf. section 6.6).

A.3.1 Définitions

Cet outil permet de représenter le nom des définitions, les dépendances entre ces noms, produites par un flot ou non, et les environnements ainsi que leur hiérarchie.

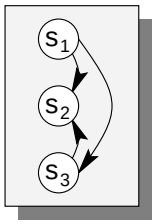
- les définitions sont représentées par le nom de la définition entouré d'un cercle ;
- les liens simples par une flèche entre deux définitions ;
- les liens crées par le contrat des flots sont représentés par une flèche en tirets ;
- les environnements sont des rectangles entourant des définitions ;
- les définitions anonymes sont représentées par un gros point noir ;
- l'appartenance d'un environnement à un autre est représentée par une flèche arrivant à l'environnement fils.

A.3.2 Exemples

Cette représentation ne s'attache qu'aux symboles et à leur utilisation. Il est donc normal que plusieurs acteurs puissent être représentés de la même manière.

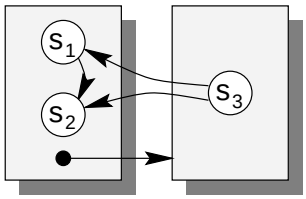
Ainsi la figure 4 présente un exemple simple de dépendance symbolique sans flot ni sous-environnement. Par exemple, la valeur associée à la définition de s_1 utilise de manière directe les symboles s_2 et s_3. Dans ce contexte, si la valeur de s_1 utilise de manière directe le symbole s_1, un cycle symbolique est créé.

Dans la figure 5 on aperçoit une hiérarchie d'environnements. Dans une hiérarchie, les dépendances symboliques ne peuvent être que dans le sens contraire



$$\left[\begin{array}{l} s_1 := f(s_2,\ s_3) \\ s_2 := \alpha \\ s_1 := g(s_2) \end{array}\right]$$

FIG. 4 - *Dépendances symboliques dans un seul environnement, sans flot.*

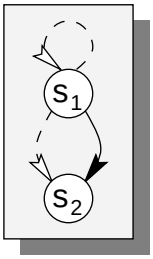


$$\left[\begin{array}{l} s_1 := f(s_2) \\ s_2 := \alpha \\ \left[\begin{array}{l} s_3 := g(s_1,\ s_2) \end{array}\right] \end{array}\right]$$

FIG. 5 - *Dépendances symboliques entre deux environnements.*

à la hiérarchie. Par exemple la valeur associée à s_3 « voit » les symboles s_1 et s_2, mais ces derniers n'ont pas accès à s_3.

Dans la figure 6 nous avons deux dépendances symboliques dues au contrat d'un flot, apparaissant en tirets. Le contrat des flots peut créer des cycles symboliques car il s'agit alors d'une équation récurrente et non d'une équation de point-fixe.



$$\left[\begin{array}{l} s_1 := s_2\ f_{by}\ f(s_1,\ s_2) \\ s_2 := \alpha \end{array}\right]$$

FIG. 6 - *Dépendances symboliques dues au contrat d'un flot.*

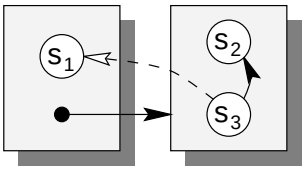
La figure 7 nous montre un flot dont le contrat crée un sous-environnement.

La figure 8 montre les dépendances lorsque la valeur d'une définition est un vecteur.

A.3.3 Détection de cycle

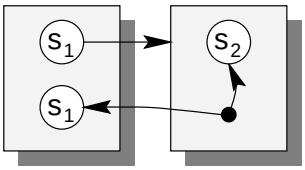
Nous présentons ici une méthode empirique pour construire le détecteur de cycle. On recherche un cycle sur un symbole s pour un acteur u dans un environnement E. On parcourt les composantes de u, et selon les acteurs rencontrés :

1. atome : un atome est acyclique ;
2. définition : continuer la recherche dans la valeur de la définition ;



$$\left[\; s_1 := \alpha\, f_{by} \left[\begin{array}{l} s_2 := \alpha \\ s_3 := f(s_1,\, s_2) \end{array}\right] \right]$$

FIG. 7 - *Dépendances symboliques dues au contrat d'un flot via un sous-environnement.*



$$\left[\begin{array}{l} s_1 := \left[\begin{array}{c} f(s_2,\, s_3) \\ s_2 := \alpha \end{array}\right] \\ s_3 := \beta \end{array}\right]$$

FIG. 8 - *Dépendances symboliques dues à un vecteur comme valeur d'une définition.*

3. flot : continuer la recherche dans l'état du flot, car le contrat ne crée pas de cycle ;
4. symbole :
   - si ce symbole est identique à s, il y a un cycle ;
   - sinon, poursuivre la recherche dans la valeur associée à ce symbole dans son environnement associé ;
5. vecteur : poursuivre la recherche dans toutes les composantes, sauf si le symbole est défini dans ce vecteur.

Le détecteur de cycle est l'opérateur de base des critères de calculabilité (cf. définition 6.7) et de stabilité (cf. définition 6.9).

B Exemples

B.1 Commentaires sur les acteurs

La définition des termes des λ-matrices amène un certain nombre de commentaires. Tout d'abord, on remarque que dans la définition même des termes, il existe déjà un certain nombre de contraintes. Par exemple, une définition est un acteur constitué d'un symbole et d'un acteur. Ces contraintes sont « statiques » dans la mesure où les termes qui ne s'y plient pas ne sont pas des acteurs. Mais il existe aussi un certain nombre de contraintes dites « dynamiques » ou sémantiques qui apparaissent dans les opérateurs de transformation définies dans la suite. Ces contraintes agissent sur les objets des acteurs :

- *alternative* : elle permet de construire des tests en fonction de la condition. Elle est constitués d'une condition, d'une clause « si » et d'une clause « sinon » ;
- *application* : le premier terme est soit un opérateur standard défini dans le formalisme, soit une fonction faisant partie de l'algèbre des données, définie par l'utilisateur. Les autres termes, en nombre variable, sont des acteurs qui forme la liste des arguments de l'application ;
- *définition* : permet de donner un nom aux acteurs. Les définitions introduisent la notions des environnements. Le premier terme est un symbole, le second est un acteur qui représente la valeur associée au symbole ;
- *extraction* : permet de lire une valeur dans un module. Un module est soit un vecteur, soit un symbole qui fait rérérence directement ou indirectement à un vecteur. Il s'agit là d'une contrainte dynamique, car elle apparaît dans l'opérateur d'évaluation et dans le calcul de la norme. Les extractions jouent le rôle des valeurs de retour des langages traditionnels et sont l'une des clef, avec les environnements, de la modularisation des λ-matrices ;
- *flot* : il peut être plus ou moins assimilé à une variable d'état. Il introduit l'opérateur de retard unitaire de l'automatisme ;
- *vecteur* : permet le regroupement dans une structure d'un nombre variable d'acteurs. Par définition, le vecteur vide n'existe pas. Il joue aussi le rôle d'environnement, l'autre clef dans la modularisation des λ-matrices avec l'extraction.

B.2 Éléments des λ-matrices

B.2.1 Terme de $\mathcal{X}$

Voici l'exemple d'un terme des λ-matrices décrit sous sa forme syntaxique, algébrique et sous forme de diagramme.

Soit $S = \left[\begin{array}{l} a{:=}2 \\ +(a,\ b) \\ \left[\begin{array}{l} a{:=}5 \\ b{:=}7 \\ +(a,\ b,\ c) \end{array}\right] \\ b{:=}13 \end{array}\right]$

Cette écriture utilise les éléments de $\mathcal{X}$ suivants :

$S = \{u_1, u_2, u_3, u_4\}_{\bar{v}}$
$u_1 = \{a, 2\}_{\bar{d}}$
$u_2 = \{+, a, b\}_{\bar{p}}$
$u_3 = \{v_1, v_2, v_3\}_{\bar{v}}$
$u_4 = \{b, 13\}_{\bar{d}}$
$v_1 = \{a, 5\}_{\bar{d}}$
$v_2 = \{e, 7\}_{\bar{d}}$
$v_3 = \{+, a, b, c, d\}_{\bar{p}}$

On peut maintenant dessiner le diagramme de s :

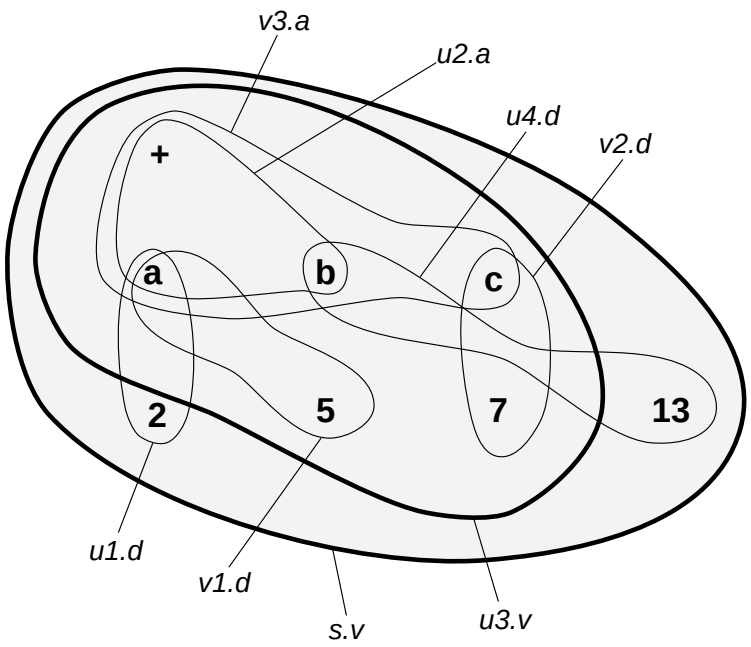


FIG. 9 - Diagramme de S. Les vecteurs sont entourés d'un trait épais. Le type de l'ensemble est accolé à son nom. Les index des composants n'apparaissent pas.

B.2.2 Système

Reprenons l'exemple donné dans la figure 1 en supposant que nous manipulons des réels, pour simplifier. Pour construire la λ-matrice associée, nous créons deux

définitions, s et q. Commençons par la plus simple des deux, c'est à dire s :

$$s := F(e,\, q).$$

Cette écriture immédiate définit s comme étant l'application de F à e et q. La variable q est définie comme un flot :

$$q := 0\, f_{by}\, G(e,\, q).$$

Cette écriture est intéressante car elle montre comment les λ-matrices permettent de créer des cycles sémantiques en utilisant les symboles. Notons que ces cycles ne sont valides qu'à travers l'utilisation des flots. La valeur initiale de q a été fixée à 0. Il ne reste maintenant plus qu'à rassembler ces deux définitions dans une structure de vecteur pour obtenir le système :

$$\left[\begin{array}{l} s := F(e,\, q) \\ q := 0\, f_{by}\, G(e,\, q) \end{array}\right].$$

Cette matrice est la description du système. Avant d'aller plus en avant, on peut établir ses propriétés en y appliquant les fonctions de critère $\Lambda(,)$. Ces fonctions booléennes permettent de caractériser un système. Appelons M le système ; il vient :

- *complétude* : $\Lambda_l(M, \perp) = 1$, car M ne contient aucune variable libre ;
- *calculabilité* : $\Lambda_c(M, \perp) = 1$, car M ne contient aucune équation de point-fixe ;
- *stable* : $\Lambda_s(M, \perp, \perp) = 1$, car la représentation en mémoire de M au cours du temps est stable ;

Comme le système est complet, la première transformation que l'on peut effectuer est la suppression des formes syntaxiques, comme par exemple $:=$, f_{by}. En supposant F et G des opérateurs primitifs, nous obtenons la λ-matrice :

$$\left[\begin{array}{l} \{s, \{F, e, q\}_{\bar{p}}\}_{\bar{d}} \\ \{q, \{0, \{G, e, q\}_{\bar{p}}\}_{\bar{s}}\}_{\bar{d}} \end{array}\right].$$

Cette λ-matrice est prête à être résolue. Cette résolution passe par l'opérateur récursif de résolution $\triangleright$ qui effectue un certain nombre de régénérations du système. Ce processus s'arrête lorsqu'une des composantes du système devient invalide, cette composante étant passée en argument $\triangleright$.

B.3 Critères

Ici, nous donnons un certain nombre d'exemples d'acteurs ne vérifiant pas les critères.

B.3.1 Complétude

Rappelons qu'un système n'est pas complet lorsqu'il contient des noms de variables non-définis dans l'environnement courant, ou dans la hiérarchie des environnements parents. Par exemple, le système suivant n'est pas complet parce que la variable x n'est pas définie :

$$\left[\ s := 1\ f_{by}\ +(s,\ 1,\ x)\ \right]$$

Dans le cas suivant, x, définie dans un sous-environnement, est inaccessible à l'environnement parent :

$$\left[\begin{array}{l} s := 1\ f_{by}\ +(s,\ 1,\ x) \\ \left[\ x := 3\ \right] \end{array}\right]$$

Les systèmes suivants sont complets car la variable x est accessible dans les deux cas :

$$\left[\begin{array}{l} x := 3 \\ s := 1\ f_{by}\ +(s,\ 1,\ x) \end{array}\right]$$

$$\left[\begin{array}{l} x := 3 \\ \left[\ s := 1\ f_{by}\ +(s,\ 1,\ x)\ \right] \end{array}\right]$$

B.3.2 Calculabilité

Un système n'est pas calculable lorsqu'il contient des équations de point-fixe qui interviennent lorsque la valeur associée à un symbole dans une définition utilise ce symbole. Par exemple :

$$\left[\ s := +(s,\ 1)\ \right]$$

Dans ce cas, le cycle est créé à l'aide de deux variables.

$$\left[\begin{array}{l} s := +(x,\ 1) \\ x := +(s,\ 1) \end{array}\right]$$

La valeur associée à un symbole doit être testée dans son environnement. L'exemple suivant n'est pas calculable, car u établit un cycle dans la définition de s :

$$\left[\begin{array}{l} s := \left[\begin{array}{l} u \\ 2 \end{array}\right] \\ u := s \end{array}\right]$$

Par contre, cet exemple est calculable, car il ne s'agit par de la même variable s :

$$\left[\begin{array}{l} s := 2 \\ \left[\ s := \left[\begin{array}{l} u \\ 2 \end{array}\right]\ \right] \\ u := s \end{array}\right]$$

Attention, les cycles sont autorisés dans le contrat des flots, comme dans :

$$\left[\ s := 1\ f_{by}\ +\ (s,\ 1)\ \right]$$

B.3.3 Stabilité

Un système n'est stable que si toutes ses composantes sont stables. Par exemple le système suivant est instable car le vecteur interne a sa première composante instable car elle fait indirectement référence à s :

$$\left[\begin{array}{l} s := \left[\begin{array}{l} u \\ 2 \end{array}\right] \\ u := s \end{array}\right]$$

Remarquons qu'il est aussi incalculable, du fait de son équation de point-fixe. Le système suivant est lui aussi instable, pour les mêmes raisons :

$$\left[\begin{array}{l} s := \left[\begin{array}{l} u \\ 2 \end{array}\right] \\ u := f(s) \end{array}\right]$$

Bien qu'étant calculable, ce système est instable car le contrat du flot est un vecteur dont la première composante est instable :

$$\left[\ s := \left[\begin{array}{l} 1 \\ 2 \end{array}\right]\ f_{by}\ \left[\begin{array}{l} s \\ 2 \end{array}\right]\ \right]$$

Ceci est aussi vrai pour ce système, bien que l'instabilité apparaisse dans une application:

$$\left[\ s := \left[\begin{array}{l} 1 \\ 2 \end{array}\right]\ f_{by}\ \left[\begin{array}{l} f(s) \\ 2 \end{array}\right]\ \right]$$

Là, nous avons un système incalculable en regard du lien sur s dans l'application, mais il est parfaitement stable, car les composantes de ses vecteurs sont stables :

$$\left[\ s := f(s)\ \right]$$

Le système suivant est calculable car le lien à lieu au travers d'un flot, et il ne contient pas de vecteur aux composantes instables :

$$\left[\ s := 1\ f_{by}\ f(s)\ \right]$$

Enfin, ce système n'est pas stable car les composantes du flots on des dimensions différentes :

$$\left[\begin{array}{l} a := 1 \\ b := \left[\begin{array}{l} 1 \\ 2 \end{array}\right] \\ f := a\ f_{by}\ b \end{array}\right]$$

B.4 Applications

B.4.1 Compresseur de données

Cette section proposes de réaliser un compresseur de données extrêmement simple. Ce compresseur reçoit un flot en entrée et compte le nombre d'occurrences d'une même valeur. Lorsque la valeur en entrée est différente de la précédente, le compresseur écrit le couple formé par le nombre d'occurrence et la valeur sur la sortie du filtre. Nous obtenons les flots suivants :

$$\begin{array}{lccccccc} e = & <1 & 5 & 5 & 5 & 7 & 7 & 1 & \ldots > \\ s = & <\bot & (1,1) & \bot & \bot & (5,3) & \bot & (7,2) & \ldots > \end{array}$$

L'intérêt de cet exemple n'est pas dans la méthode de compression, mais dans le sur échantillonnage de la sortie par rapport à l'entrée. En effet, si les valeurs d'entrées changent à chaque fois, le flot de sortie aura deux fois plus de valeurs.

Pour réaliser ce compresseur, voici les flots nécessaires :

$$\begin{array}{lccccccc} e = & <1 & 5 & 5 & 5 & 7 & 7 & 1 & \ldots > \\ e^{-1} = & <\bot & 1 & 5 & 5 & 5 & 7 & 7 & \ldots > \\ t = & <0 & 0 & 1 & 1 & 0 & 1 & 0 & \ldots > \\ c = & <\bot & 1 & 1 & 2 & 3 & 1 & 2 & \ldots > \\ s_1 = & <\bot & 1 & \bot & \bot & 5 & \bot & 7 & \ldots > \\ s_2 = & <\bot & 1 & \bot & \bot & 3 & \bot & 2 & \ldots > \end{array}$$

où :

- e est la valeur de l'entrée n° 0 ;
- e' est cette même entrée, retardée d'une unité de temps ;
- t est le résultat de la comparaison de e et e' ;
- c est un compteur réinitialisé à 1 chaque fois que t vaut 0 ;
- s réalisent l'envoi de e' et c sur les ports de sortie lorsque t vaut 0.

Il ne reste plus qu'à écrire la λ-matrice correspondante. Ici, le sur échantillonnage est réalisé en considérant les valeurs de sorties comme un vecteur à deux dimension :

$$\left[\begin{array}{l} e := in(0) \\ r := \bot\ f_{by}\ e \\ t := \stackrel{?}{=}(e,\ r) \\ c := \bot\ f_{by}\ \stackrel{?}{=}(t,\ 0) \rightarrow 1, +(c,\ 1) \\ s_1 := r \\ s_2 := c \end{array}\right]$$

En appliquant les fonctions de critère à ce système, on montre qu'il est complet, calculable, stable et borné.

Le sur-échantillonnage peut aussi être réalisé en modifiant les flots de manière à ce qu'ils produisent deux fois plus de valeurs, en répétant une fois sur deux la valeurs précédente, sauf pour la sortie.

C Preuves

C.1 Sur les applications de bases ...

Dans ce qui suit, on suppose x, y, $u_1, u_2, \ldots, u_i$, $v_1, v_2, \ldots, v_i$, $z_1, z_2, \ldots, z_i \in \mathcal{X}$, $\forall i \in \mathbb{N}^*$.

Preuve 1 *: **un acteur qui appartient à un autre est algébriquement inclus dans ce dernier, ce qui s'énonce*** $x \succeq y \Rightarrow x \subset y$.

On a $x = \{u_1, u_2, \ldots, u_n\}_s$ *et* $y = \{v_1, v_2, \ldots, v_m\}_t$.

Si $x \succeq y$, *il existe au moins un* $\alpha \in \mathbb{N}^*$ *tel que* $\delta(\alpha, y) = x$.

Donc il existe au moins un v_α *de* y *égal à* x. *Comme* $v_\alpha \subset y$, *on a* $x \subset y$.

C.Q.F.D.

Preuve 2 *: **un acteur qui est inclus au sens des λ-matrices dans un autre est algébriquement inclus dans ce dernier et réciproquement, ce qui s'énonce*** $x \succ y \Leftrightarrow x \subset y$.

Soit $z_1, z_2, \ldots, z_n, y$ *le chemin de* x *dans* y. *Par définition, on a* $x \succeq z_1$, $z_1 \succeq z_2$, ..., $z_n \succeq y$. *On en déduit que* $x \subset z_1$, $z_1 \subset z_2$, ..., $z_k \subset y$, *d'où* $x \subset y$.

C.Q.F.D.

Preuve 3 *: **un acteur ne s'appartient pas à lui-même, ce qui s'énonce*** $x \not\succeq x$.

Si $x \succeq x$, *on a* $x \subset x$, *ce qui est contraire à la définition des éléments de* $\mathcal{X}$ *(cf. définition 4.2).*

C.Q.F.D.

Preuve 4 *: **un acteur ne peut s'inclure lui-même, ce qui s'énonce*** $x \succ / x$.

Soit $z_1, z_2, \ldots, z_n, x$ *le chemin de* x *dans* x. *On en déduit que* $x \subset z_1$, $z_1 \subset z_2$, ..., $z_k \subset x$, *d'où* $x \subset x$, *ce qui est contraire à la définition des éléments de* $\mathcal{X}$.

C.Q.F.D.

Preuve 5 *: **si un acteur appartient à un autre, la réciproque est fausse, ce qui s'énonce*** $x \succeq y \Rightarrow y \not\succeq x$.

Comme $x \succeq y$, *on a* $x \subset y$.

Si $y \succeq x$, *on a* $y \subset x$, *ce qui impose* $x = y$, *et donc que* $x \not\succeq y$ *et* $y \not\succeq x$.

Donc, si $x \succeq y$, *on a* $y \not\succeq x$.

C.Q.F.D.

Preuve 6 *: **si un acteur est inclus dans un autre, la réciproque est toujours fausse, ce qui s'énonce*** $x \succ y \Rightarrow y \not\succ x$.

Preuve identique à $x \succeq y \Rightarrow y \not\succeq x$.

C.Q.F.D.

C.2 Sur les environnements ...

Preuve 7 : ***tout symbole a une valeur associée, éventuellement $\perp$, et donc la recherche de cette valeur se termine toujours, ce qui s'énonce*** $\rho(s,E) \in \mathcal{X}, \forall s \in \mathcal{I}, \forall E \in \Gamma$.

Rappelons que E est un environnement de la forme $E = E_n \star \ldots \star E_2 \star E_1$, avec $E_1, E_2, \ldots, E_n \in \mathcal{X}$. De plus, les environnements appartiennent à l'ensemble des acteurs énumérés et indexés dont la définition interdit les cycles.

Si la recherche ne se termine pas, c'est qu'un cycle peut exister dans au moins l'un des E_i.

Comme aucun cycle n'est présent dans l'environnement, la recherche se termine toujours.

C.Q.F.D.

Preuve 8 : ***la valeur associée à un symbole dans un environnement régénéré est le régénéré de cette valeur dans l'environnement original, et l'environnement associé à la valeur associée à un symbole dans un environnement régénéré est le régénéré de l'environnement associé à cette valeur dans l'environnement original, ce qui s'énonce :***

$$\forall E \in \Gamma,\ \left.\begin{array}{r} \mu(s,E) = E_v \neq \perp \\ \rho(s,E) = v \neq \perp \end{array}\right\} \Rightarrow \left\{\begin{array}{l} \mu(s,\overset{\triangledown}{E}) = \overset{\triangledown}{E}_v, \\ \rho(s,\overset{\triangledown}{E}) = \bigtriangledown_E v. \end{array}\right.$$

On pose $E = A \star E_v \star B$ tel que $\mu(s,B) = \perp$, et $\mu(s,E_v) = E_v$.

L'environnement régénéré de E est $\overset{\triangledown}{E} = \overset{\triangledown}{A} \star \overset{\triangledown}{E}_v \star \overset{\triangledown}{B}$, avec $\overset{\triangledown}{E}_v = \bigtriangledown_A E_v$, par définition.

Comme $\mu(s,E_v) = E_v$, on a $s := v \succ E_v$. On a donc $s := \bigtriangledown_{A\star E_v} v \succ \bigtriangledown_A E_v$.

Donc on a $\mu(s, \bigtriangledown_A E_v) = \bigtriangledown_A E_v = \overset{\triangledown}{E}_v$, et $\rho(s, \bigtriangledown_A E_v) = \bigtriangledown_{A\star E_v} v$.

Ceci nous conduit à $\mu(s, \overset{\triangledown}{A} \star \overset{\triangledown}{E}_v) = \overset{\triangledown}{A} \star \overset{\triangledown}{E}_v$ et à $\rho(s, \overset{\triangledown}{A} \star \overset{\triangledown}{E}_v) = \bigtriangledown_{A\star E_v} v$.

Pour que s soit définie dans l'environnement $\overset{\triangledown}{B}$, occultant ainsi la définition de E_v, il faut qu'elle soit définie dans B, ce qui n'est pas le cas.

Ainsi, nous obtenons $\mu(s,\overset{\triangledown}{E}) = \overset{\triangledown}{E}_v$ et $\rho(s,\overset{\triangledown}{E}) = \bigtriangledown_{E_v} v$.

C.Q.F.D.

C.3 Sur les opérateurs dynamiques ...

C.3.1 Évaluation

Preuve 9 : ***l'évaluation a pour codomaine l'ensemble des acteurs figés, ce qui s'énonce*** $\forall u \in \mathcal{X} \quad | \quad \bigtriangleup_E u \in \mathcal{X}, v \succ \bigtriangleup_E u \Rightarrow v \in \overline{\mathcal{X}}$.

Procédons par induction sur les termes des λ-matrices pris en compte par l'opérateur d'évaluation.

- *alternative : $u = c \rightarrow a, s$.*

On a $\triangle_E u$ *qui vaut soit* $\triangle_E a$, *soit* $\triangle_E s$.

Par hypothèse de récurrence, $\triangle_E a$ *et* $\triangle_E s$ *appartiennent à* $\overline{\mathcal{X}}$.

- *application :* $u = o(a_1, a_2, \ldots, a_n)$.

  On a $\triangle_E u = \triangleleft o(a_1, a_2, \ldots, a_n)$.

  Par définition $\triangleleft \alpha$ *appartient à* $\mathcal{D} + \{\bot\}$ *et donc à* $\overline{\mathcal{X}}$.

- *atome :* $u = x$.

  On a $\triangle_E u = x$, *donc* $x \in \overline{\mathcal{X}}$.

- *définition :* $u = s := v$.

  On a $\triangle_E u = \triangle_E v$, *qui par hypothèse de récurrence appartient à* $\overline{\mathcal{X}}$.

- *extraction :* $u = v \cdot i$.

  On a $\triangle_E u = \delta(\triangle_E i, \triangle_E v)$. *Par hypothèse de récurrence,* $\triangle_E v \in \overline{\mathcal{X}}$, *donc toutes ses composantes sont elles-aussi dans* $\overline{\mathcal{X}}$.

  Comme la valeur indexée est soit $\bot$ *qui fait partie de* $\overline{\mathcal{X}}$, *soit l'une des composantes de* $\triangle_E v$, *on en conclut que* $\triangle_E u \in \overline{\mathcal{X}}$.

- *flot :* $u = e\, f_{by}\, c$.

  On a $\triangle_E u = \triangle_E e \in \overline{\mathcal{X}}$, *par hypothèse de récurrence.*

- *symbole :* $u = s$.

  On a $\triangle_E u = \triangle_{E_v} v$, *avec* $v = \rho(s, E)$ *et* $E_v = \mu(s, E)$.

  Par hypothèse de récurrence, $\triangle_{E_v} v \in \overline{\mathcal{X}}$.

- *vecteur :* $u = \begin{bmatrix} c_1 \\ \ldots \\ c_k \end{bmatrix}$.

  On a $\triangle_E u = \begin{bmatrix} \triangle_{E'} c_1 \\ \ldots \\ \triangle_{E'} c_k \end{bmatrix}$, *avec* $E' = E \star u$.

  On a $\triangle_{E'} c_1, \triangle_{E'} c_2, \ldots, \triangle_{E'} c_k \in \overline{\mathcal{X}}$, *par hypothèse de récurrence, alors* $\begin{bmatrix} c'_1 \\ \ldots \\ c'_k \end{bmatrix} \in \overline{\mathcal{X}}$.

C.Q.F.D.

Preuve 10 *:* ***un acteur figé est identique à son*** $k^{\text{ième}}$ ***évalué, ce qui s'énonce*** $\forall u \in \overline{\mathcal{X}}, u = \triangle_\bot u$

Par induction sur les termes de $\overline{\mathcal{X}}$, *on obtient :*

- *atomes, sans les symboles :* $u = s$.

  On a $\triangle_E u = x = u$.

– *vecteur :* $u = \left[\begin{array}{c} c_1 \\ \ldots \\ c_k \end{array} \right]$.

De même, on a $\triangle_E u = \left[\begin{array}{c} \triangle_{E'} c_1 \\ \ldots \\ \triangle_{E'} c_k \end{array} \right]$, *avec* $E' = E \star u$.

Par hypothèse de récurrence, $\triangle_{E'} c_i = c_i$, *pour* $i \in [1, k]$.

Donc $\triangle_E u = u$.

C.Q.F.D.

C.3.2 Complétude

Preuve 11 *:* ***si un système est complet, à tout symbole autre que le nom d'une définition correspond une valeur dans son environnement courant, ce qui s'énonce*** $\forall u \in \mathcal{X}, \Lambda_l(u, E) = 1 \Rightarrow \forall s \in \mathcal{I} \mid s \succ u, s \notin \mathcal{F}, \rho(s, F) \neq \perp$**.**

Par définition, on a $\Lambda_l(u, E) = \prod_{i=1}^{n} \Lambda_l(s_i, E_i)$, *où* s_i *sont des symboles.*

Or, si $\Lambda_l(u, E) = 1$, *on a* $\Lambda_l(s_i, E_i) = 1$, *ce qui signifie que* $\rho(s_i, E_i) \neq \perp$.

C.Q.F.D.

Preuve 12 *:* ***si un système est complet et que son régénéré existe, il est complet, ce qui s'énonce :*** $\forall u \in \mathcal{X}, \Lambda_l(u, E) = 1 \Rightarrow \Lambda_l(\bigtriangledown_E u, \overset{\bigtriangledown}{E}) = 1$**.**

Nous utilisons un raisonnement récurrent par induction sur les termes des λ*-matrices pris en compte à la fois par la complétude et la régénération :*

– *atome :* $u = x$.

Comme le régénéré d'un atome est lui-même, $\bigtriangledown_E u = x$, *et qu'un atome (sauf un symbole, décrit plus bas) est toujours complet, on a* $\Lambda_l(u, E) = 1 \Rightarrow \Lambda_l(\bigtriangledown_E u, \overset{\bigtriangledown}{E}) = 1$.

– *définition :* $u = \ s{:=}v$.

Le régénéré d'une définition est $\bigtriangledown_E u = \ s{:=} \bigtriangledown_E v$.

Comme u est complet, on a $\Lambda_l(u, E) = 1 \Rightarrow \Lambda_l(v, E) = 1$, *ce qui donne par hypothèse de récurrence* $\Lambda_l(\bigtriangledown_E v, \overset{\bigtriangledown}{E}) = 1$ *qui vaut aussi* $\Lambda_l(\bigtriangledown_E u, \overset{\bigtriangledown}{E})$.

Donc on a $\Lambda_l(u, E) = 1 \Rightarrow \Lambda_l(\bigtriangledown_E u, \overset{\bigtriangledown}{E}) = 1$.

– *flot :* $u = e \, f_{by} \, c$.

Le régénéré d'un flot est $\bigtriangledown_E u = \triangle_E c \, f_{by} \ \bigtriangledown_E \ c$.

Comme u est complet, nous obtenons $\Lambda_l(u, E) = 1 \Rightarrow \Lambda_l(e, E).\Lambda_l(c, E) = 1$.

Par hypothèse de récurrence, nous avons $\Lambda_l(c, E) = 1 \Rightarrow \Lambda_l(\bigtriangledown_E c, \overset{\bigtriangledown}{E}) = 1$. *Nous avons aussi prouvé que toute évaluation est complète (cf. preuve 19), donc nous obtenons* $\Lambda_l(\triangle_E c, \overset{\bigtriangledown}{E}) = 1$.

Nous obtenons donc $\Lambda_l(\bigtriangledown_E u, \overset{\triangledown}{E}) = 1$.

- *objet :* $u = \{c_1, c_2, \ldots, c_n\}_t$.

  Le régénéré d'un objet et l'objet des régénérés de ses composantes, soit $\bigtriangledown_E u = \{\bigtriangledown_E c_1, \bigtriangledown_E c_2, ..., \bigtriangledown_E c_n\}_t$.

  Comme par hypothèse u est complet, on a $\Lambda_l(u, E) = 1 \Rightarrow \prod_{i=1}^{n} \Lambda_l(c_i, E) = 1$, *ce qui nous donne par hypothèse de récurrence* $\prod_{i=1}^{n} \Lambda_l(\bigtriangledown_E c_i, \overset{\triangledown}{E}) = 1$, *qui vaut aussi* $\Lambda_l(\bigtriangledown_E u, \overset{\triangledown}{E})$.

- *symbole :* $u = s$.

  Le régénéré d'un symbole est ce symbole, soit $\bigtriangledown_E u = s$.

  Comme u est complet, on a $\Lambda_l(u, E) = 1 \Rightarrow \rho(s, E) = v$, *avec* $v \neq \perp$.

  Or la recherche de s dans l'environnement régénéré de E donne $\rho(s, \overset{\triangledown}{E}) = \bigtriangledown_E v = v'$, *avec* $v' \neq \perp$.

  Donc nous avons $\Lambda_l(\bigtriangledown_E u, \overset{\triangledown}{E}) = 1$.

- *vecteur :* $u = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}$.

  Le régénéré d'un vecteur est $\bigtriangledown_E u = \begin{bmatrix} \bigtriangledown_{E'} c_1 \\ \cdots \\ \bigtriangledown_{E'} c_n \end{bmatrix}$, *avec* $E' = E \star u$.

  L'environnement régénéré de E' *est* $\overset{\triangledown}{E'} = \overset{\triangledown}{E} \star \overset{\triangledown}{u} = \overset{\triangledown}{E} \star \bigtriangledown_E u$.

  Par ailleurs, comme u est complet, on a $\Lambda_l(u, E) \Rightarrow \prod_{i=1}^{n} \Lambda_l(c_i, E') = 1$, *ce qui donne par hypothèse de récurrence* $\prod_{i=1}^{n} \Lambda_l(\bigtriangledown_{E'} c_i, \overset{\triangledown}{E'}) = 1$.

  Si le régénéré est complet, alors $\Lambda_l(\bigtriangledown_E u, \overset{\triangledown}{E}) = 1 \Rightarrow \prod_{i=1}^{n} \Lambda_l(\bigtriangledown_{E'} c_i, E'') = 1$, *avec* $E'' = \overset{\triangledown}{E} \star \bigtriangledown_E u = \overset{\triangledown}{E'}$.

  Donc nous avons $\Lambda_l(\bigtriangledown_E u, \overset{\triangledown}{E}) = 1$.

C.Q.F.D.

C.3.3 Détecteur de cycle

Preuve 13 *:* ***aucun évalué s'il existe ne crée de cycle, ou, autrement dit, les acteurs figés sont acycliques, ce qui s'énonce*** $\forall s \in \mathcal{I}, \forall u \in \overline{\mathcal{X}}, \forall E \in \Gamma, \gamma(s, u, E) = 1$**.**

Comme seul un symbole est capable d'annuler le détecteur de cycle (cf. définition 6.6), et que les symboles et tous les acteurs en contenant ne font pas partie de l'ensemble des acteurs figés, alors aucun évalué s'il existe n'annule le détecteur de cycle.

C.Q.F.D.

Preuve 14 : ***si aucun cycle n'existe sur un symbole dans un acteur d'un environnement, alors aucun cycle n'existe non plus sur ce symbole dans l'acteur régénéré de l'environnement régénéré, ce qui s'énonce*** $\gamma(s,u,E)=1 \Rightarrow \gamma(s,\ \bigtriangledown_E u, \overset{\bigtriangledown}{E}), \forall u \in \mathcal{X}, \forall E \in \Gamma$**.**

Par induction sur les termes des λ-matrices concernés par le détecteur de cycle et l'opérateur de régénération, on obtient :

- *atome (sauf symbole) :* $u = x$.

  Le régénéré d'un atome et lui-même, $\bigtriangledown_E u = x$. *Or aucun atome ne créé de cycle,* $\gamma(s,x,E)=1$, *donc nous avons* $\gamma(s,\ \bigtriangledown_E u, \overset{\bigtriangledown}{E})=1$.

- *définition :* $u = \ s:=v$.

  Le régénéré d'une définition est $\bigtriangledown_E u = \ s:= \bigtriangledown_E v$.

  Comme u est acyclique, nous avons $\gamma(s,u,E)=1 \Rightarrow \gamma(s,v,E)=1$. *Par hypothèse de récurrence, nous avons* $\gamma(s,\ \bigtriangledown_E v, \overset{\bigtriangledown}{E})=1$, *et donc* $\gamma(s,\ \bigtriangledown_E u, \overset{\bigtriangledown}{E})=1$.

- *flot :* $u = e\ f_{by}\ c$.

  Le régénéré d'un flot est $\bigtriangledown_E u = \bigtriangleup_E c\ f_{by}\ \bigtriangledown_E\ c$.

  Si le régénéré du flot est acyclique, on a $\gamma(s,\ \bigtriangleup_E c, \overset{\bigtriangledown}{E})=1$, *ce qui est vrai car tout évalué est acyclique. Donc dans ce cas aussi on a* $\gamma(s,\ \bigtriangledown_E u, \overset{\bigtriangledown}{E})=1$.

- *objet :* $u = \{c_1, c_2, \ldots, c_n\}_t$.

  Par définition, le régénéré d'un objet est l'objet de ses régénérés, on a $\bigtriangledown_E u = \{\ \bigtriangledown_E c_1,\ \bigtriangledown_E c_2, ...,\ \bigtriangledown_E c_n\}_t$.

  Gomme l'objet est acyclique, on a $\gamma(s,u,E)=1 \Rightarrow \prod_{i=1}^{n} \gamma(s,c_i,E)=1$. *Par hypothèse de récurrence, nous avons* $\prod_{i=1}^{n} \gamma(s,\ \bigtriangledown_E c_i, \overset{\bigtriangledown}{E})=1$. *Nous en concluons que* $\gamma(s,\ \bigtriangledown_E u, \overset{\bigtriangledown}{E})=1$.

- *symbole :* $u = s'$

  Comme u est acyclique, $\gamma(s,u,E)=1$, *par définition nous avons*

$$\begin{cases} s \neq s', \\ \gamma(s, \rho(s,E), \mu(s',E)) = 1 \end{cases} \tag{40}$$

  .

  Si le régénéré est acyclique, on a $\gamma(s,\ \bigtriangledown_E u, \overset{\bigtriangledown}{E})=1$, *donc* $\gamma(s,s',\overset{\bigtriangledown}{E})=1$, *ce qui nous amène, comme* $s \neq s'$ *à* $\gamma(s,\rho(s',\overset{\bigtriangledown}{E}),\mu(s',\overset{\bigtriangledown}{E}))=1$.

  D'une part, on a montré que si $\mu(s',E)=E_v$ *alors* $\mu(s',\overset{\bigtriangledown}{E}) = \overset{\bigtriangledown}{E_v}$. *Il est évident que si* $E \neq E_v$ *alors* $\overset{\bigtriangledown}{E} \neq \overset{\bigtriangledown}{E_v}$.

D'autre part, on a montré que si $\rho(s', E) = v$ *alors* $\rho(s', \overset{\triangledown}{E}) = \bigtriangledown_{E_v} v$. *Par hypothèse de récurrence, on a* $\gamma(s, v, E_v) \Rightarrow \gamma(s, \bigtriangledown_E v, \overset{\triangledown}{E_v})$.

Ici aussi, on a $\gamma(s, \bigtriangledown_E u, \overset{\triangledown}{E}) = 1$.

– *vecteur :* $u = \left[\begin{array}{c} c_1 \\ \vdots \\ c_n \end{array} \right]$.

Si s est définie dans u, alors $\rho(s, u) \neq \perp$, *est donc u est acyclique sur s. On a donc* $\rho(s, \overset{\triangledown}{u}) \neq \perp$, *et donc le régénéré de u est aussi acyclique.*

Sinon, on a $\rho(s, \overset{\triangledown}{u}) = \perp$, *donc comme u est acyclique,* $\prod_{i=1}^{n} \gamma(s, c_i, E') = 1$, *avec* $E' = E \star u$. *Par hypothèse de récurrence* $\prod_{i=1}^{n} \gamma(s, \bigtriangledown_{E'} c_i, \overset{\triangledown}{E'}) = 1$, *donc* $\gamma(s, \bigtriangledown_E u, \overset{\triangledown}{E}) = 1$.

Nous avons donc $\gamma(s, \bigtriangledown_E u, \overset{\triangledown}{E}) = 1$.

C.Q.F.D.

C.3.4 Calculabilité

Preuve 15 *:* ***l'évalué d'un acteur calculable existe, ce qui s'énonce*** $\forall u \in \mathcal{X}, \Lambda_c(u, E) = 1 \Rightarrow \exists!\ \triangle_E\ u \in \mathcal{X}$**.**

Un acteur n'est pas évaluable lorsque l'une de ses composantes ne l'est pas.

L'évaluation d'un acteur n'existe pas lorsqu'elle ne se termine pas. Et elle ne se termine pas lorsqu'elle conduit à ré-évaluer cet acteur, formant ainsi un cycle.

Comme un acteur ne peut se contenir lui-même (cf. preuve 4), un cycle dans un évaluation ne peut se produire qu'en la présence d'un cycle symbolique.

Montrons donc d'un acteur calculable ne peut contenir de tels cycles.

Soit une définition dont la valeur utilise le nom de manière à former un cycle symbolique. Posons :

$$u = \ s{:=}v \ \ | \ \ \left\{ \begin{array}{l} s \succ v, \\ e\ f_{by}\ c \text{ avec } s \succ c \not\succ v, \\ s{:=}v' \not\succ v, \end{array} \right.$$

La calculabilité de u est par définition :

$$\Lambda_c(u, E) = \left\{ \begin{array}{l} \Lambda_c(v, E), \text{ si } \gamma(s, v, E) = 1, \\ 0, \text{ sinon}, \end{array} \right.$$

Il vient $\gamma(s, v, E) = \prod_{i=1}^{n} \gamma(s, u_i, E_i)$.

Et pour au moins une réalisation de i, on a $\gamma(s, s, E_\beta)$ *qui vaut* 0. *Donc* $\Lambda_c(u, E)$ *vaut aussi* 0.

C.Q.F.D.

Preuve 16 *:* ***si un système est calculable, son régénéré l'est aussi, ce qui s'énonce :*** $\forall u \in \mathcal{X}, \Lambda_c(u, E) = 1 \Rightarrow \Lambda_c(\bigtriangledown_E u, \overset{\triangledown}{E}) = 1$**.**

Nous utilisons un raisonnement récurrent par induction sur les termes des λ-matrices pris en compte à la fois par la calculabilité, la régénération et le détecteur de cycle :

- *atome :* $u = x$.

  Comme le régénéré d'un atome est lui-même, $\bigtriangledown_E u = x$, *et qu'un atome (sauf un symbole, décrit plus bas) est toujours calculable, on a* $\Lambda_c(u, E) = 1 \Rightarrow \Lambda_c(\bigtriangledown_E u, \overset{\bigtriangledown}{E}) = 1$.

- *définition :* $u = \ s := v$.

  Le régénéré d'une définition est $\bigtriangledown_E u = \ s := \bigtriangledown_E v$.

  Comme par hypothèse u est calculable, nous obtenons $\begin{cases} \gamma(s, v, E) = 1, \\ \Lambda_c(v, E) = 1. \end{cases}$

  Par hypothèse de récurrence nous avons $\Lambda_c(\bigtriangledown_E v, \overset{\bigtriangledown}{E}) = 1$. *Nous avons de plus montré que* $\gamma(s, v, E) = 1 \Rightarrow \gamma(s, \bigtriangledown_E v, \overset{\bigtriangledown}{E}) = 1$. *Donc* $\Lambda_c(\bigtriangledown_E u, \overset{\bigtriangledown}{E}) = 1$.

- *flot :* $u = e\ f_{by}\ c$.

  Le régénéré d'un flot est $\bigtriangledown_E u = \bigtriangleup_E c\ f_{by}\ \bigtriangledown_E\ c$.

  Comme u est calculable, nous obtenons $\Lambda_c(u, E) = 1 \Rightarrow \Lambda_c(e, E).\Lambda_c(c, E) = 1$.

  Par hypothèse de récurrence, nous avons $\Lambda_c(c, E) = 1 \Rightarrow \Lambda_c(\bigtriangledown_E c, \overset{\bigtriangledown}{E}) = 1$. *Nous avons aussi montré que toute évaluation est calculable (cf. preuve 19), donc nous obtenons* $\Lambda_c(\bigtriangleup_E c, \overset{\bigtriangledown}{E}) = 1$.

  Nous en concluons que $\Lambda_c(\bigtriangledown_E u, \overset{\bigtriangledown}{E}) = 1$.

- *objet :* $u = \{c_1, c_2, \ldots, c_n\}_t$.

  Le régénéré d'un objet et l'objet des régénérés de ses composantes, soit $\bigtriangledown_E u = \{\bigtriangledown_E c_1, \bigtriangledown_E c_2, ..., \bigtriangledown_E c_n\}_t$.

  Comme par hypothèse u est calculable, $\Lambda_c(u, E) = 1 \Rightarrow \prod_{i=1}^{n} \Lambda_c(c_i, E) = 1$, *ce qui nous donne par hypothèse de récurrence* $\prod_{i=1}^{n} \Lambda_c(\bigtriangledown_E c_i, \overset{\bigtriangledown}{E}) = 1$. *Donc* $\Lambda_c(\bigtriangledown_E u, \overset{\bigtriangledown}{E}) = 1$.

- *symbole :* $u = s$.

  Le régénéré d'un symbole est ce symbole, soit $\bigtriangledown_E u = s$.

  Or un symbole est par définition toujours calculable, donc $\Lambda_c(\bigtriangledown_E u, \overset{\bigtriangledown}{E}) = 1$.

– *vecteur :* $u = \left[\begin{array}{c} c_1 \\ \cdots \\ c_n \end{array} \right]$.

Le régénéré d'un vecteur est $\bigtriangledown_E u = \left[\begin{array}{c} \bigtriangledown_{E'} c_1 \\ \cdots \\ \bigtriangledown_{E'} c_n \end{array} \right]$, *avec* $E' = E \star u$.

L'environnement régénéré de E' *est* $\overset{\bigtriangledown}{E'} = \overset{\bigtriangledown}{E} \star \overset{\bigtriangledown}{u} = \overset{\bigtriangledown}{E} \star \bigtriangledown_E u$.

Par ailleurs, comme u *est complet, on a* $\Lambda_c(u, E) \Rightarrow \prod_{i=1}^{n} \Lambda_c(c_i, E') = 1$, *ce qui donne par hypothèse de récurrence* $\prod_{i=1}^{n} \Lambda_c(\bigtriangledown_{E'} c_i, \overset{\bigtriangledown}{E'}) = 1$.

Si le régénéré est calculable, alors $\Lambda_c(\bigtriangledown_E u, \overset{\bigtriangledown}{E}) = 1 \Rightarrow \prod_{i=1} n\Lambda_c(\bigtriangledown_{E'} c_i, E'') = 1$, *avec* $E'' = \overset{\bigtriangledown}{E} \star \bigtriangledown_E u = \overset{\bigtriangledown}{E'}$.

Donc nous avons $\Lambda_c(\bigtriangledown_E u, \overset{\bigtriangledown}{E}) = 1$.

C.Q.F.D.

C.3.5 Norme

Preuve 17 *:* ***la norme de tout acteur calculable dans un environnement est supérieure ou égale à celle de son évalué sans environnement, ce qui s'énonce :***

$\forall u \in \mathcal{X}, \forall E \in \Gamma, \Lambda_c(u, E) = 1 \Rightarrow \sigma(u, E) \geq \sigma(\bigtriangleup_E u, \bot,)$**.**

Par induction sur les termes des λ*-matrices concernés par la norme, nous obtenons :*

– *alternative :* $u = c \rightarrow a, s$.

L'évaluation d'une alternative donne $\bigtriangleup_E u = \bigtriangleup_E a$ *ou* $\bigtriangleup_E s$, *selon la valeur de* $\bigtriangleup_E c$.

Par définition, la norme d'une alternative est

$$\sigma(u, E) = sup(\sigma(a, E), \sigma(s, E)) \tag{41}$$

.

Donc $\sigma(\bigtriangleup_E u, \bot) = \sigma(\bigtriangleup_E a, \bot)$ *ou* $\sigma(\bigtriangleup_E s, \bot)$.

Nous obtenons donc :

	$\sigma(u, E)$	
$\sigma(\bigtriangleup_E u, \bot)$	$\sigma(a, E)$	$\sigma(s, E)$
$\sigma(\bigtriangleup_E a, \bot)$	$\sigma(a, E)$ $\geq \sigma(\bigtriangleup_E a, \bot)$	$\sigma(s, E)$ $> \sigma(a, E)$ $\geq \sigma(\bigtriangleup_E a, \bot)$
$\sigma(\bigtriangleup_E s, \bot)$	$\sigma(a, E)$ $> \sigma(s, E)$ $\geq \sigma(\bigtriangleup_E s, \bot)$	$\sigma(s, E)$ $\geq \sigma(\bigtriangleup_E s, \bot)$

Pour conclure, nous avons $\sigma(u, E) \geq \sigma(\triangle_E u, \bot,)$.

- *application :* $u = o(a_1, a_2, \ldots, a_n)$.

  Le résultat de l'évaluation d'une application est soit une donnée de $\mathcal{D}$ *soit* $\bot$. *Comme la norme d'une application est par définition* 1 *et que la norme d'une donnée ou de* $\bot$ *est aussi* 1, *on en conclut que* $\sigma(u, E) \geq \sigma(\triangle_E u, \bot)$.

- *atome :* $u = x$.

  On a $\triangle_E u = x$, *donc* $\sigma(u, E) = \sigma(\triangle_E u, \bot) = 1$.

- *définition :* $u = s{:=}v$.

  L'évaluation d'une définition est $\triangle_E u = \triangle_E v$.

  Or la norme d'une définition est $\sigma(u, E) = \sigma(v, E)$, *et par hypothèse de récurrence* $\sigma(v, E) \geq \sigma(\triangle_E v, \bot)$ *qui n'est autre que* $\sigma(\triangle_E u, \bot)$.

  Là encore, on a $\sigma(u, E) \geq \sigma(\triangle_E u, \bot)$.

- *extraction :* $u = v \cdot i$.

  L'évaluation de u vaut soit $\bot$ *soit* $\triangle_{E'} c_\alpha$, *avec* $E' = E \star v$. *Le premier cas est trivial.*

  Dans le second cas, on pose $v = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}$. *On a* $\sigma(u, E) = \sup_{i=1}^{n} \sigma(c_i, E')$.

  Posons $\triangle_{E'} u = \sigma(c_\beta, E') \geq \sigma(c_\alpha, E')$.

  Par hypothèse de récurrence, nous avons $\sigma(c_\alpha, E') \geq \sigma(\triangle_{E'} c_\alpha, \bot)$.

  Donc, dans les deux cas, $\sigma(u, E) \geq \sigma(\triangle_E u, \bot)$.

- *flot :* $u = e\ f_{by}\ c$.

  L'évaluation d'un flot est $\triangle_E u = \triangle_E e$.

  Or la norme d'un flot est $\sigma(u, E) = \sigma(e, E)$, *ce qui donne par hypothèse de récurrence* $\sigma(e, E) \geq \sigma(\triangle_E e, \bot)$ *qui n'est autre que* $\sigma(\triangle_E u, \bot)$.

  Nous avons donc $\sigma(u, E) \geq \sigma(\triangle_E u, \bot)$.

- *symbole :* $u = s$.

  L'évaluation d'un symbole dans un environnement est l'évaluation de la valeur associé dans l'environnement associé à ce symbole, soit $\triangle_E u = \triangle_{E_v} v$, *avec* $v = \rho(s, E)$ *et* $E_v = \mu(s, E)$.

  Or la norme d'un symbole dans un environnement est la norme de la valeur associée dans l'environnement associé, ce qui donne $\sigma(s, E) = \sigma(v, E_v)$. *Par hypothèse de récurrence, nous avons* $\sigma(v, E_v) \geq \sigma(\triangle_{E_v} v, \bot)$, *qui n'est autre que* $\sigma(\triangle_E u, \bot)$.

  Ici encore, nous avons bien $\sigma(u, E) \geq \sigma(\triangle_E u, \bot)$.

– *vecteur :* $u = \left[\begin{array}{c} c_1 \\ \vdots \\ c_n \end{array} \right]$.

L'évaluation d'un vecteur donne $\triangle_E\, u = \left[\begin{array}{c} \triangle_{E'}\, c_1 \\ \vdots \\ \triangle_{E'}\, c_n \end{array} \right]$, *avec* $E' = E \star u$.

La norme d'un vecteur est la somme des normes de ses composantes, soit $\sigma(u, E) = \sum_{i=1}^{n} \sigma(c_i, E')$. *Par hypothèse de récurrence, on a* $\sigma(u, E) \geq \sum_{i=1}^{n} \sigma(\triangle_{E'}\, c_i, \perp)$.

Nous savons que, la norme de l'évalué d'un vecteur est

$$\sigma(\triangle_E\, u, \perp) = \sum_{i=1}^{n} \sigma(\triangle_{E'}\, c_i, E''), \text{ avec } E'' = \triangle_E\, u \qquad (42)$$

.

On montre facilement que si $\sigma(\triangle_{E'}\, c_i, \perp) = \alpha$, *alors* $\sigma(\triangle_{E'}\, c_i, E'') = \alpha$, *car* E'' *ne contient aucun symbole (cf. preuve 9).*

Donc on a $\sigma(u, E) \geq \sigma(\triangle_E\, u, \perp)$.

Nous obtenons bien pour tous les cas $\sigma(u, E) \geq \sigma(\triangle_E\, u, \perp)$.
C.Q.F.D.

Preuve 18 *:* ***la norme d'un acteur calculable dans un environnement est supérieure ou égale à la dimension de son évalué, ce qui s'énonce*** $\forall u \in \mathcal{X}, \forall E \in \Gamma,$
$\Lambda_c(u, E) = 1 \Rightarrow \sigma(u, E) \geq |\triangle_E\, u|$**.**

Nous avons montré que l'évalué d'un acteur calculable appartient à l'ensemble des acteurs figés (cf. preuve 9).

Donc, $|\triangle_E\, u| = \sigma(\triangle_E\, u, \perp)$ *(cf. preuve 20).*

Or $\sigma(\triangle_E\, u, \perp) \geq \sigma(u, E)$ *(cf. preuve 17).*

Donc $\sigma(u, E) \geq |\triangle_E\, u|$.

C.Q.F.D.

C.3.6 Acteur neutre et figé

Les preuves suivantes ressemblent dans leur formes aux précédentes. Nous en donnerons donc une écriture simplifiée.

Preuve 19 *:* ***tout figé vérifie tous les critère :*** $\forall M \in \mathcal{X} \quad | \quad \exists\, \triangle_\perp\, u \in \mathcal{X}, \Lambda_l(\triangle_\perp\, M, \perp) = 1$**.**

Les fonctions de critère sont des fonctions récursives. Elles sont annulées dans certaines conditions. Nous allons montré simplement que ces conditions ne peuvent être atteintes avec les acteurs figés.

Rappelons que l'ensemble des acteurs figés est constitué des vecteurs $\mathcal{V}$ *et des atomes* $\mathcal{A}$ *sans les symboles* $\mathcal{I}$.

– *complétude : annulée pour un symbole qui ne fait pas partie des acteurs figés ;*

- *calculabilité : annulée pour une définition qui ne fait par partie des acteurs figés ;*
- *stabilité : annulée pour une alternative ou pour pour une extraction, qui ne font pas partie des acteurs figés ;*
- *mesurabilité : raison identique à celles de la stabilité.*

C.Q.F.D.

Preuve 20 *:* ***la norme de tout acteur figé est égale à sa dimension, ce qui s'énonce :*** $\forall u \in \overline{\mathcal{X}}, \sigma(u, \perp) = |u|$**.**

Par induction sur les termes des acteurs figés, nous obtenons :

- *atome :* $u = x$.

  $|\triangle_E u| = |x| = 1 = \sigma(u, E)$.

- *vecteur :* $u = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}$.

  $|u| = \sum_{i=1}^{n} |c_i|$.

  Par hypothèse de récurrence
  $\sum_{i=1}^{n} |c_i| = \sum_{i=1}^{n} \sigma(c_i, E') = \sigma(u, E)$, *avec* $E' = E \star u$.

Preuve 21 *:* ***la norme de tout acteur neutre est égale à la dimension de son évalué, ce qui s'énonce :*** $\forall u \in \mathbb{X}_n, \sigma(u, \perp) = |\triangle_E u|$**.**

Par induction sur les termes des acteurs neutres, nous obtenons :

- *atome :* $u = x$.

  $|\triangle_E u| = |x| = 1 = \sigma(u, E)$.

- *application :* $u = o(a_1, a_2, \ldots, a_n)$.

  $|\triangle_E u| = |\triangle_E u| = |x| = 1 = \sigma(u, E)$.

- *vecteur :* $u = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}$.

  $|u| = \sum_{i=1}^{n} |c_i|$.

  Par hypothèse de récurrence
  $\sum_{i=1}^{n} |c_i| = \sum_{i=1}^{n} \sigma(c_i, E') = \sigma(c_i, E') = \sigma(u, E)$, *avec* $E' = E \star u$.

Preuve 22 *:* ***le régénéré de tout acteur neutre est également neutre, ce qui s'énonce :*** $\forall u \in \mathbb{X}_n, \bigtriangledown_E u \in \mathbb{X}_n$**.**

Par induction sur les termes de acteurs figé, nous obtenons :

- *atome :* $u = x$.

  $\bigtriangledown_E u = u \in \mathcal{A} \subset \mathbb{X}_n$

- *application :* $u = o(a_1, a_2, \ldots, a_n)$ *avec* $a_1, a_2, \ldots, a_n \in \mathbb{X}_n$.

  $\bigtriangledown_E u = o(\bigtriangledown_E a_1, \bigtriangledown_E a_2, \ldots, \bigtriangledown_E a_n) \in \mathcal{P}$.

  Par hypothèse de récurrence $\bigtriangledown_E a_1, \ldots \bigtriangledown_E a_n \in \mathbb{X}_n$, *donc* $\bigtriangledown_E u \in \mathbb{X}_n$.

- *vecteur :* $u = \left[\begin{array}{c} c_1 \\ \vdots \\ c_n \end{array} \right]$, *avec* $c_1, c_2, \ldots, c_n \in \mathbb{X}_n$.

  $\bigtriangledown_E u = \left[\begin{array}{c} \bigtriangledown_{E'} c_1 \\ \cdots \\ \bigtriangledown_{E'} c_n \end{array} \right] \in \mathcal{V}$, *avec* $E' = E \star u$.

  Par hypothèse de récurrence on a $\bigtriangledown_{E'} c_1, \bigtriangledown_{E'} c_n \in \mathbb{X}_n$, *donc* $\bigtriangledown_E u \in \mathbb{X}_n$.

Preuve 23 *:* ***la norme de tout acteur neutre est égale à la norme de son régénéré, ce qui s'énonce :*** $\forall u \in \mathbb{X}_n, \sigma(u, \perp) = \sigma(\bigtriangledown_E u, \overset{\bigtriangledown}{E})$**.**

Par induction sur les termes des acteurs neutres, nous obtenons :

- *atome :* $u = x$.

  $\sigma(\bigtriangledown_E u, \overset{\bigtriangledown}{E}) = \sigma(x, \overset{\bigtriangledown}{E}) = 1 = \sigma(u, E)$.

- *application :* $u = o(a_1, a_2, \ldots, a_n)$.

  $\sigma(\bigtriangledown_E u, \overset{\bigtriangledown}{E}) = 1 = \sigma(u, E)$.

- *vecteur :* $u = \left[\begin{array}{c} c_1 \\ \vdots \\ c_n \end{array} \right]$.

  $\sigma(\bigtriangledown_E u, E) = \sum_{i=1}^{n} \sigma(\bigtriangledown_{E'} c_i, \overset{\bigtriangledown}{E'})$, *avec* $E' = E \star u$.

  Par hypothèse de récurrence on a $\sum_{i=1}^{n} \sigma(\bigtriangledown_{E'} c_i, \overset{\bigtriangledown}{E'}) = \sum_{i=1}^{n} \sigma(c_i, \overset{\bigtriangledown}{E'}) = \sigma(u, E)$.

C.3.7 Stabilité

Preuve 24 *:* ***la dimension d'un acteur stable dans un environnement est supérieure ou égale à la dimension de son régénéré dans l'environnement régénéré, ce qui s'énonce*** $\forall u \in \mathcal{X}, \forall E \in \Gamma$,
$\Lambda_s(u, E) = 1 \Rightarrow |u| \geq |\bigtriangledown_E u|$**.**

Par induction sur les termes des λ*-matrices concernés par la dimension et la régénération, nous obtenons :*

- *alternative :* $u = c \rightarrow a, s$.

  $|u| = |c| + |a| + |s|$.

  $|\bigtriangledown_E u| = |\bigtriangledown_E c| + |\bigtriangledown_E a| + |\bigtriangledown_E s|$.

  Par hypothèse de récurrence : $|c| \geq |\bigtriangledown_E c|$, $|c| \geq |\bigtriangledown_E a|$, $|s| \geq |\bigtriangledown_E s|$.

  Donc $|u| \geq |\bigtriangledown_E u|$.

- *atome :* $u = x$.

  $|u| = |x| = 1 = |\bigtriangledown_E u|$.

- *flot :* $u = e\ f_{by}\ c$.

  $|u| = |e| + |c|$.

  $|\bigtriangledown_E u| = |\bigtriangleup_E c| + |\bigtriangledown_E c|$.

  Par hypothèse de récurrence $|c| \geq |\bigtriangledown_E c|$.

  Comme $e \in \overline{\mathcal{X}}$, *on a* $\sigma(e, \bot) = |e|$ *(cf. preuve 20).*

  Nous avons montré que $|\bigtriangleup_E c| \leq \sigma(c, E)$ *(cf. preuve 18).*

  Comme $\sigma(e, E) \geq \sigma(c, E)$ *on a* $|e| \geq |\bigtriangleup_E c|$.

  Donc $|u| \geq |\bigtriangledown_E u|$.

- *objet :* $u = \{c_1, c_2, \ldots, c_n\}_t$.

  $|u| = \sum_{i=1}^{n} |c_i|$.

  $|\bigtriangledown_E u| = \sum_{i=1}^{n} |\bigtriangledown_E c_i|$.

  Par hypothèse de récurrence $|c_i| \geq |\bigtriangledown_E c_i|$.

  Donc $|u| \geq |\bigtriangledown_E u|$.

- *vecteur :* $u = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}$.

  $|u| = \sum_{i=1}^{n} |c_i|$.

  $|\bigtriangledown_E u| = \sum_{i=1}^{n} \left|\bigtriangledown_{E'} c_i\right|$, *avec* $E' = E \star u$.

  Par hypothèse de récurrence $|c_i| \geq \left|\bigtriangledown_{E'} c_i\right|$.

  Donc $|u| \geq |\bigtriangledown_E u|$.

Nous obtenons bien pour tous les cas $|u| \geq |\bigtriangledown_E u|$.
C.Q.F.D.

Preuve 25 *:* ***la norme du régénéré d'un acteur stable est inférieure ou égale à la dimension de l'évalué de cet acteur, ce qui s'énonce :*** $\forall u \in \mathcal{X}, \forall E \in \Gamma, \Lambda_s(u, E) = 1 \Rightarrow \sigma(\bigtriangledown_E u, \overset{\triangledown}{E}) \leq |\bigtriangleup_E u|$**.**

Par induction sur les termes des λ*-matrices concernés par la norme, nous obtenons :*

- *alternative :* $u = c \rightarrow a, s$.

  $a, s \in \overline{\mathcal{X}} \Rightarrow \sigma(a, E) = |\bigtriangleup_E a|$ *(cf. preuve 21) et* $\sigma(s, E) = |\bigtriangleup_E s|$, *avec* $\sigma(a, E) = \sigma(s, E)$ *(cf. preuve 20).*

  De plus $\sigma(a, E) = \sigma(\bigtriangledown_E a, \overset{\triangledown}{E})$ *et* $\sigma(s, E) = \sigma(\bigtriangledown_E s, \overset{\triangledown}{E})$ *(cf. preuve 23).*

Enfin, on a $\bigtriangledown_E a, \bigtriangledown_E s \in \mathbb{X}_n$ *(cf. preuve 22).*

Donc $\sigma(\bigtriangledown_E u, \overset{\triangledown}{E}) \leq |\bigtriangleup_E u|$.

- *atome :* $u = x$.

  $\sigma(\bigtriangledown_E u, \overset{\triangledown}{E}) = \sigma(x, \overset{\triangledown}{E}) = 1 = |x| = |\bigtriangleup_E u|$.

- *application :* $u = o(a_1, a_2, \ldots, a_n)$.

  $\sigma(\bigtriangledown_E u, \overset{\triangledown}{E}) = 1 = |x| = |\bigtriangleup_E u|$, *avec* $x \in \mathcal{D}$.

- *définition :* $u = s := v$.

  $\sigma(\bigtriangledown_E u, \overset{\triangledown}{E}) = \sigma(\bigtriangledown_E v, \overset{\triangledown}{E})$.

  Par hypothèse de récurrence $\sigma(\bigtriangledown_E v, \overset{\triangledown}{E}) \leq |\bigtriangleup_E v| = |\bigtriangleup_E u|$.

- *extraction :* $u = u \cdot i$.

  Comme u est stable, i est un entier appartenant à $\mathcal{N}$. *Donc son régénéré est lui-même,* $\bigtriangledown_E i = i$. *Ce qui nous conduit à* $\sigma(\bigtriangledown_E u, \overset{\triangledown}{E}) = \sigma(\bigtriangledown_E v \cdot i, \overset{\triangledown}{E})$ *et donc nous avons* $\sigma(\delta(i, \bigtriangledown_E v), \overset{\triangledown}{E})$.

  Si $\delta(i, v) = \perp$, *alors* $\delta(i, \bigtriangleup_E v) = \delta(i, \bigtriangledown_E v) = \perp$. *Donc* $\sigma(\perp, \overset{\triangledown}{E}) = |\perp|$.
  Si $\delta(i, v) = c_\alpha$, *avec* $c_\alpha \succ v$, *alors* $\delta(i, \bigtriangleup_E v) = \bigtriangleup_{E'} c_\alpha$ *et* $\delta(i, \bigtriangledown_E v) = \bigtriangledown_{E'} c_\alpha$. *Donc on a* $\sigma(\delta(i, \bigtriangledown_E v), \overset{\triangledown}{E'}) = \sigma(\bigtriangledown_{E'} c_\alpha, \overset{\triangledown}{E'})$. *Par hypothèse de récurrence* $\sigma(\bigtriangledown_{E'} c_\alpha, \overset{\triangledown}{E'}) \geq \left|\bigtriangleup_{E'} c_\alpha\right| = |\delta(i, \bigtriangleup_E v)| = |\bigtriangleup_E u|$.

  Donc, dans les deux cas, $\sigma(\bigtriangledown_E u, \overset{\triangledown}{E}) \geq |\bigtriangleup_E u|$.

- *flot :* $u = e\, f_{by}\, c$.

  $\sigma(\bigtriangledown_E u, \overset{\triangledown}{E}) = \sigma(\bigtriangleup_E c, \overset{\triangledown}{E}) = \sigma(\bigtriangleup_E c, \perp)$.
  On a montré que $\sigma(\bigtriangleup_E c, \perp) = |\bigtriangleup_E c|$ *(cf. preuve 20), et que* $|\bigtriangleup_E c| \leq \sigma(c, E)$ *(cf. preuve 18).*
  On sait que $\sigma(c, E) \leq \sigma(e, E)$ *car u est stable.*
  On a montré que $\sigma(e, E) = |e|$ *(cf. preuve 20) et que* $|e| = |\bigtriangleup_E e|$ *(cf. preuve 9).*

  Comme $|\bigtriangleup_E e| = |\bigtriangleup_E u|$ *on a* $\sigma(\bigtriangledown_E u, \overset{\triangledown}{E}) \leq |\bigtriangleup_E u|$.

- *symbole :* $u = s$.

  $\sigma(\bigtriangledown_E u, \overset{\triangledown}{E}, =)\sigma(s, \overset{\triangledown}{E}) = 1 \leq \left|\bigtriangleup_{E_v} v\right| = |\bigtriangleup_E u|$, *avec* $v = \rho(s, E)$ *et* $E_v = \mu(s, E)$.

– *vecteur :* $u = \left[\begin{array}{c} c_1 \\ \vdots \\ c_n \end{array} \right]$.

$\sigma(\bigtriangledown_E u, \overset{\triangledown}{E}) = \sum_{i=1}^{n} \sigma(\bigtriangledown_{E'} c_i, \overset{\triangledown}{E'})$.

Par hypothèse de récurrence on a $\sum_{i=1}^{n} \sigma(\bigtriangledown_{E'} c_i, \overset{\triangledown}{E'}) \leq \sum_{i=1}^{n} \left| \bigtriangleup_{E'} c_i \right| = |\bigtriangleup_E u|$.

Nous obtenons bien pour tous les cas $\sigma(\bigtriangledown_E u, \overset{\triangledown}{E}) \leq |\bigtriangleup_E u|$
C.Q.F.D.

Preuve 26 *: **le régénéré d'un acteur stable dans un environnement est stable dans l'environnement régénéré, ce qui s'énonce*** $\forall u \in \mathcal{X}, \forall E \in \Gamma, \Lambda_s(u, E) = 1 \Rightarrow \Lambda_s(\bigtriangledown_E u, \overset{\triangledown}{E}) = 1$**.**

– *atome :* $u = x$.

$\Lambda_s(\bigtriangledown_E u, \overset{\triangledown}{E}) = \Lambda_s(x, \overset{\triangledown}{E}) = 1$.

– *alternative :* $u = c \rightarrow a, s$.

$\bigtriangledown_E u = \bigtriangledown_E c \rightarrow \bigtriangledown_E a, \bigtriangledown_E s$.

Par hypothèse de récurrence, $\bigtriangledown_E c$ *est stable.*

$a, s \in \overline{\mathcal{X}} \Rightarrow \left\{ \begin{array}{l} \sigma(a, E) = \sigma(\bigtriangledown_E a, \overset{\triangledown}{E}), \\ \sigma(s, E) = \sigma(\bigtriangledown_E s, \overset{\triangledown}{E}). \end{array} \right.$

Comme u est stable, on a $\sigma(a, E) = \sigma(s, E)$.

Donc $\sigma(\bigtriangledown_E a, \overset{\triangledown}{E}) = \sigma(\bigtriangledown_E s, \overset{\triangledown}{E})$.

Donc $\Lambda_s(\bigtriangledown_E u, \overset{\triangledown}{E}) = 1$.

– *extraction :* $u = v \cdot i$.

Par hypothèse de récurrence, $\bigtriangledown_E v$ *est stable.*

Comme u est stable, i est un entier appartenant à $\mathcal{N}$.

On a $\bigtriangledown_E u = \bigtriangledown_E v \cdot \bigtriangledown_E i = \bigtriangledown_E v \cdot i$, *car* $\bigtriangledown_E i = i$.

Donc $\Lambda_s(\bigtriangledown_E u, \overset{\triangledown}{E}) = 1$.

– *flot :* $u = e\, f_{by}\, c$.

$\bigtriangledown_E u = \bigtriangleup_E c\, f_{by}\ \bigtriangledown_E c$.

Par hypothèse de récurrence $\bigtriangledown_E c$ *est stable.*

On a montré que $\bigtriangleup_E c \in \overline{\mathcal{X}}$ *(cf. preuve 9).*

D'autre part $\sigma(\bigtriangledown_E c, \overset{\triangledown}{E}) \leq |\bigtriangleup_E c|$ *(cf. preuve 25) et* $|\bigtriangleup_E c| = \sigma(\bigtriangleup_E c, \overset{\triangledown}{E})$ *(cf. preuve 20).*

On a donc $\Lambda_s(\bigtriangledown_E u, \overset{\triangledown}{E}) = 1$.

– *objet :* $u = \{c_1, c_2, \ldots, c_n\}_t$.

$\Lambda_s(\bigtriangledown_E u, \overset{\triangledown}{E}) = \prod_{i=1}^{n} \Lambda_s(\bigtriangledown_E c_i, \overset{\triangledown}{E})$ *qui vaut* 1 *par hypothèse de récurrence.*

– *vecteur :* $u = \left[\begin{array}{c} c_1 \\ \vdots \\ c_n \end{array} \right]$.

$\Lambda_s(\bigtriangledown_E u, \overset{\triangledown}{E}) = \prod_{i=1}^{n} \Lambda_s(\bigtriangledown_{E'} c_i, \overset{\triangledown}{E})$ *qui vaut* 1 *par hypothèse de récurrence, avec* $E' = E \star u$.

C.Q.F.D.

D Lexique

Les ensembles classiques	
$\mathbb{D}$	données
$\mathbb{N}$	entiers naturels
$\mathbb{O}$	opérateurs
$\mathbb{O}_D$	opérateurs sur les données
$\mathbb{O}_S$	opérateurs spéciaux
$\mathbb{I}$	symboles

Les ensembles des λ-matrices	
$\mathcal{T}$	alternatives
$\mathcal{P}$	applications
$\mathcal{F}$	définitions
$\mathcal{D}$	données
$\mathcal{N}$	entiers
$\mathcal{E}$	extraction
$\mathcal{S}$	flots
$\mathcal{O}_D$	opérateurs sur données
$\mathcal{O}_S$	opérateurs spéciaux
$\mathcal{O}$	opérateurs
$\mathcal{I}$	symboles
$\mathcal{V}$	vecteurs

Les ensembles de construction	
$\mathcal{X}$	acteurs
$\overline{\mathcal{X}}$	acteurs figés
$\mathbb{X}_n$	acteurs neutres
$\mathcal{A}$	atomes
$\mathbb{G}_N$	acteurs énumérés
$\mathbb{G}_{NI}$	acteurs énumérés et indexés
$\mathbb{G}_{NIT}$	acteurs énumérés indexés et typés
$\mathbb{C}$	constructeurs
$\mathbb{T}$	types
$\mathbb{I}$	index
Γ	environnements

Éléments de l'ensemble des types

$\bar{a}$	alternative
$\bar{p}$	application
$\bar{d}$	définition
$\bar{t}$	donnée
$\bar{n}$	entier
$\bar{e}$	extraction
$\bar{s}$	flot
$\bot$	indéfini
$\bar{o}$	opérateur
$\top$	surdéfini
$\bar{i}$	symbole
$\bar{v}$	vecteur

Opérateurs spéciaux

$\triangle$	évaluation
$\triangledown$	régénération
$\triangleleft$	réduction
$\triangleright$	résolution

Opérateurs de construction

$\gamma(,)$	acyclique
$\mu(,)$	environnement associé
$\delta(,)$	index
$\sigma(,)$	norme
$\tau()$	type
$\rho(,)$	valeur associée

Objet, forme syntaxique

$c \rightarrow a, s$	alternative
$o(a_1, a_2, \ldots, a_n)$	application
$s := v$	définition
$u \cdot i$	extraction
$e\, f_{by}\, c$	flot
$\begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}$	vecteur

Atomes et objet, forme matricielle

$\{c, a, s\}_{\bar{a}}$	alternative
$\{o, , a_1, a_2, \ldots, a_n\}_{\bar{p}}$	application
$\{s, v\}_{\bar{d}}$	définition
$\{u, i\}_{\bar{e}}$	extraction
$\{d\}_{\bar{t}}$	donnée
$\{e\}_{\bar{n}}$	entier
$\{e, c\}_{\bar{s}}$	flot
$\{o\}_{\bar{o}}$	opérateur
$\{s\}_{\bar{i}}$	symbole
$\{c_1, \ldots, c_n\}_{\bar{v}}$	vecteur
$\{c_1, c_2, \ldots, c_n\}_t$	acteur, en général

Critères

$\Lambda_c(,)$	calculabilité
$\Lambda_l(,)$	complétude
$\Lambda(,)$	critère, en général
$\Lambda_s(,)$	stabilité

Constructions syntaxiques

$f(a_1, a_2, \ldots, a_k)$	fonction de f sur a
$f_{i,E}u$	fonction $f(u, i, E)$
$x \succeq y$	appartenance matricielle
$Card(u)$	cardinal
$\|u\|$	dimension
$E \star F$	chaînage d'environnement
$\overset{\triangledown}{E}$	environnement régénéré
$x \succ y$	inclusion matricielle

Flèches

$\Leftrightarrow$	équivalent
$\Rightarrow$	implique

Index