

Rapport de stage

Développement d'un noyau de programmation synchrone

Jordan Ischard
3ème année de licence Informatique
Université d'Orléans

25 Mars 2019

Table des matières

1	Introduction	3
2	Journal de bord	4
2.1	Mars	4
2.1.1	Semaine 25 au 29 Mars	4
2.2	Avril	4
2.2.1	Semaine du 1 au 5 Avril	4
2.2.2	Semaine du 8 au 12	5
2.2.3	Semaine du 15 au 19	6
2.2.4	Semaine du 22 au 16	7
2.3	Mai	8
2.3.1	Semaine du 29 Avril au 3	8
2.3.2	Semaine du 6 au 10	8
2.3.3	Semaine du 13 au 17	8
2.3.4	Semaine du 20 au 24	8
3	Recherche d'Informations	9
3.1	Le Réactive ML	9
3.2	Les λ -calculs	10
3.2.1	Les règles de β -réduction	10
3.2.2	Les règles de réduction générale	10
3.2.3	Les règles de priorité	10
3.2.4	Comment savoir si on a une forme normal ?	10
3.3	ISWIM	11
3.4	Les différentes machines traitées	12
3.4.1	CC Machine	12
3.4.2	SCC Machine	12
3.4.3	CK Machine	13
3.4.4	CEK Machine	13
3.4.5	SECD Machine	14
4	Conclusion	15
5	Annexes	16
5.1	Les Exemples des machines étudiées	16
5.1.1	Exemple de fonctionnement de la machine CC	16
5.1.2	Exemple de fonctionnement de la machine SCC	16
5.1.3	Exemple de fonctionnement de la machine CK	17
5.1.4	Exemple de fonctionnement de la machine CEK	18
5.1.5	Exemple de fonctionnement de la machine SECD	19
5.2	Les différentes versions faite pour rendre la machine SECD concurrente	20
5.2.1	1ère version des règles de la machine SECD Concurrente	20
5.2.2	2ème version des règles de la machine SECD Concurrente	21
5.2.3	3ème version des règles de la machine SECD Concurrente	22
5.2.4	4ème version des règles de la machine SECD Concurrente	22
6	Bibliographie	23

1 Introduction

2 Journal de bord

2.1 Mars

2.1.1 Semaine 25 au 29 Mars

Afin de me remettre dans le contexte du stage et de comprendre ces problématiques, le lundi a été entièrement consacré à la relecture et lecture des deux articles ,proposés durant l'entretien, un sur le RéactiveML[1] et un sur le ZINC[2].

Toutes mes notes, qui sont un résumé de ce que j'ai compris, sont données dans la section **Recherche d'Informations**.

Le mardi matin a servie à clarifier mes questions en une petite liste présentée lors de la réunion. La réunion s'est déroulée l'après-midi et a permis d'apporter des réponses aux questions pré-écrite le matin et à fixer des objectifs. La réunion en a fixé 2 :

1. Implémenter les λ -calculs
2. Implémenter la machine SECD

2 semaines m'ont été accordé pour réaliser ces objectifs dans le langage de programmation *Ocaml*. J'ai eu, pour cela, un article traitant des λ -calculs[3] afin de m'aider dans ma démarche.

Il a été décidé que tous les programmes seront partagé sur un github qui a pour adresse :

<https://github.com/JordanIschard/StageL3>.

Le reste de la journée a été utilisé pour commencer ce dit article.

La compréhension des λ -calcul a été étalé sur le reste de la semaine. Cependant l'avancement a été perturbé le mercredi à cause d'un problème de compréhension des règles de priorité dans les λ -calculs. Ce temps perdu en compréhension a été compensé dans l'écriture de ce rapport.

M. Dabrowski m'a aidé à régler ce problème le jeudi matin ce qui m'a permis d'assimilé en presque totalité les λ -calculs et de commencer l'implémentation de ceux-ci en Ocaml.

2.2 Avril

2.2.1 Semaine du 1 au 5 Avril

L'implémentation des λ -calculs a presque été terminé à l'exception du parser, de l' α -réduction et d'un petit bug de renommage lié à la règle de β -réduction suivante :

$$(\lambda X_1.M_1)[X_2 \leftarrow M_2] = (\lambda X_3.M_1[X_1 \leftarrow X_3][X_2 \leftarrow M_2])$$

où $X_1 \neq X_2$, $X_3 \notin FV(M_2)$ et $X_3 \notin FV(M_1) \setminus X_1$

La réunion a changé les objectifs de base , en gardant le langage Ocaml, par les objectifs suivants :

1. Implémenter les λ -calculs
2. Implémenter la machine CC et par extension la machine SCC
3. Implémenter la machine CK et par extension la machine CEK
4. Implémenter la machine SECD

Une semaine supplémentaire m'a été accordé pour faire cela.

La compréhension des différentes machines ont été faite le lundi après la réunion, le mardi et le mercredi, toutes les informations récolté sont dans la partie traitant des différentes machines.

Cependant rien a été implémenté à cause d'un problème lié à Ocaml et Emacs. En effet, Ocaml propose un système de module pour pouvoir séparer des parties de codes dans différents fichiers.

Malheureusement après une après-midi complète de recherche sur ce sujet, Je n'ai pas réussi à trouver une solution. Un mail a donc été envoyé à M. Dabrowski et Mme Bousdira pour avoir une aide le mercredi matin.

La structure des méthodes et leurs contenus a commencé à être écrit sur papier jusqu'au jeudi midi. En effet, M. Dabrowski m'a aidé avec les modules en Ocaml. Il fallait laisser tomber emacs et son interaction dynamique et faire une compilation grâce à *ocamlbuild*.

L'implémentation du langage ISWIM a été fait le vendredi . La machine CC est presque terminé mais un bug lié aux opérations créer une erreur. L'implémentation de la machine SCC sera faite durant la semaine prochaine. Le problème de l'implémentation des λ -calculs a été fixé ainsi que le bug de la machine CC durant le week-end.

2.2.2 Semaine du 8 au 12

L'implémentation des machines CK,CEK et SECD ont été faite le lundi et le mardi. La réunion hebdomadaire ne s'est pas déroulé le lundi par manque de temps, donc les nouvelles consignes de mes professeurs n'ont pas été donné. En attendant le mardi après-midi a été utilisé pour lire la suite de l'article [3].

Une implémentation du systèmes d'erreur a été faite le mercredi matin. Une réunion a été placé le jeudi à 14h. Le temps entre mercredi matin et jeudi après-midi fut utilisé à la lecture du même article et plus précisément la mise en place de gestionnaire d'erreur et le système d'affectation.

La réunion a permis de mettre en place un objectif ainsi qu'un objectif secondaire :

1. Implémenter une concurrence dans la machine SECD avec les notions suivantes :

- (a) **spawn t** : lance un nouveau thread et le mets en attente
- (b) **present s in t1 t2** : si le signal s est présent prends t1 sinon attend le prochain temps logique et prends t2
- (c) **emit s** : émet le signal s
- (d) **signal s in t** : initialise le signal s pour t

2. Implémenter une gestion d'erreur dans la machine SECD

Ces 2 objectifs doivent être atteint en 3 semaines maximum.

Le reste de la semaine a été utilisé pour écrire les règles de la machine SECD avec les nouvelles notions. En effet, toute la difficulté du rajout de notion dans une machine est de ne pas "casser" les règles de base. Pour cela, un petit point théorique est obligatoire.

Tout d'abord, il faut distinguer deux types de thread différentes :

- ceux qui attendent
- ceux qui sont bloqués

Ce qui oblige de rajouter à notre machine SECD deux élément en plus, \widehat{W} les éléments qui attendent et \widehat{ST} les éléments qui sont bloqués.

La question suivante se pose : **Que doit contenir les éléments de ces deux listes ?**

La réponse est : le nécessaire au fonctionnement des threads indépendamment les uns des autres. Comme dit plus haut ils sont indépendant les uns des autres , leurs piles, leurs environnements, leurs chaînes de contrôle et leurs dépôts sont obligatoirement indépendants pour ne pas créer de conflits mais par contre la liste d'attente et la liste d'éléments bloqués sont eux communs à tous.

Donc \widehat{W} contient une liste de sauvegarde de la machine SECD c'est-à-dire une liste de \widehat{D} . \widehat{ST} contient une liste de couple contenant une sauvegarde et le signal attendu. On a pour l'instant :

$$\widehat{W} = \{\widehat{D}, \dots\}$$

$$\widehat{ST} = \{(s, \widehat{D}), \dots\} \text{ avec } s \text{ un signal}$$

Ensuite, il faut savoir quand un signal est émit et enfin le plus dur quand est-ce que le signal n'est pas émit. La liste de signaux émis doit-il être commun ou privé à chacun ? La question mérite d'être posé et après plusieurs essais le plus simple reste commun car pour vérifier la présence d'un signal émit par un autre thread avant il faut partager cette liste. Du coup on a \widehat{SI} qui est une liste de signaux émis.

On dit qu'un signal n'est pas émit si pendant tout un instant logique on a pas d'émission de ce signal. **Comment est représenté la fin d'un instant logique ?** Il est représenté par la \widehat{W} vide, \widehat{D} vide et \widehat{C} vide.

Du coup quand ces trois conditions sont réunis, on peut prendre tous les éléments de \widehat{ST} et les travailler en prenant en compte que le signal n'a pas été émit.

Une première version sans le **signal s in t** a commencé à être conçu.

2.2.3 Semaine du 15 au 19

La 1ère version des règles de la nouvelle machine SECD fût terminée le lundi et vous pouvez trouver ces dites règles en Annexe. Un ajout a été fait en plus dans la 1ère version, quand on tombe sur un élément comme **spawn** ou **emit** il faut continuer de faire fonctionner la machine. Donc une constante **Remp** a été ajoutée dans l'optique de ne pas casser le fonctionnement de la machine SECD de base.

Cette version est inutilisable car elle part dans l'optique que tous les signaux sont déjà initialisés.

Pour palier à ce problème une 2ème version de ces règles fût créée le lundi matin, vous pouvez les retrouver en Annexe comme pour la 1ère version.

L'implémentation de ces règles a été faite le lundi après-midi. Cette implémentation semble concluante mais il reste potentiellement des failles. Cette semaine aucune réunion sera faite donc je présenterai mes travaux à la réunion de la semaine prochaine pour avoir l'approbation de mes professeurs.

L'objectif principal étant atteint, je me suis penché sur l'objectif secondaire le mardi.

Tout d'abord, j'ai rajouté un système d'erreur dans la machine SECD qui arrête le fonctionnement de celle-ci et renvoie un message d'erreur et non qui s'arrête par la gestion d'erreur de Ocaml. Le système fonctionnant, j'ai changé les erreurs par des **Throw** en prévision de son utilisation futur. M'inspirant des règles créées pour la machine CCH présentée dans [3] j'ai aussi rajouté un autre élément à la machine SECD concurrente : le **Handler**. Le **Handler** doit contenir une sauvegarde de la machine secd au complet. Donc on a :

$$\widehat{H} = \text{Vide} \mid \langle \text{erreur}, \langle \widehat{S}, \widehat{E}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \rangle$$

Pour aller de pair avec le **Throw** j'ai aussi rajouté un **Catch**. Une 3ème version des règles de la machine SECD concurrente fût créée le mardi après-midi. Elle se trouve, elle aussi, en Annexe. L'implémentation a été faite le mercredi matin.

Ayant terminé le travail demandé, j'ai envoyé un mail le mercredi pour avoir plus ample information sur le travail à faire ensuite. Un mail contenant la suite m'a été envoyé le mercredi vers midi. Après avoir créé des threads et les avoir fait travailler indépendamment, le but est ici de donner la possibilité de partager des valeurs.

Pour cela il faut :

1. Un **identifiant unique** à chaque thread
2. une **liste de valeurs** liés à un signal et un thread précis
3. une opération **put s v** qui met une valeur v dans la liste de valeurs associée à s qui est lui-même associé au thread
4. une opération **get id s** qui récupère une valeur v dans la liste de valeurs associée à s qui est lui-même associé au thread id

Un temps de réflexion sur ces ajouts, et les questions qui vont avec, a été faite le mercredi après-midi. Les questions étaient liées à quand est-ce que l'on met à jour les valeurs partagées principalement mais aussi comment avec le get on récupère les informations.

Tout le jeudi et le vendredi ont été utilisé pour voir les ajouts à la machine. Voilà le fruit de mes réflexions. Reprenons dans l'ordre :

- Un **identifiant unique** à chaque thread : Cela implique deux choses.
 1. Il faut garder en mémoire l'id de chaque thread, en soit il faut rajouté un élément dans la machine car un thread en cours est une machine secd classique donc on rajoute \widehat{I} l'identifiant et il faut le garder aussi dans la sauvegarde faite pour la mise en attente ;
 2. Il faut que l'id soit unique, pour cela deux possibilités :
 - (a) un id d'un thread fini peut être réutilisé mais cela peut créer des conflit plus tard (De plus, le vendredi un mail de M. Dabrowski confirmera mes craintes et privilegiera la 2nd option dans un premier temps)
 - (b) un id d'un thread fini est bloqué jusqu'à la fin du fonctionnement de la machine. Pour cela, il faut garder un entier dans la machine qui va être pris pour chaque nouveau thread et qui va s'incrémenter. On a donc \widehat{AI} un entier.
- une **liste de valeurs** liés à un signal et un thread précis : Cela implique de modifier la liste des signaux pour ajouter une liste de valeur et un identifiant de thread liés.

De plus, dans la version précédente les signaux initialisés étaient dans l'environnement mais ici cela impliquera de garder deux listes de valeurs indépendante pour le même signal et le même thread. Ce serai un gâchis d'espace mémoire. Du coup, on va tout mettre dans \widehat{SI} avec la forme suivante :

$$\widehat{SI} = \{\langle id, \{\langle signal, \{valeur, \dots\}, init, emit \rangle\} \rangle, \dots\}$$

avec id = un identifiant de thread

avec init = un booléen indiquant si il est initialisé ou non

avec emit = un booléen indiquant si il est émit ou non

On peut légitimement se poser la question de pourquoi un booléen init, et je vais vous expliquer pourquoi.

En effet, quand on mets un signal dans cette liste c'est qu'il a été initialisé à un moment dans le fonctionnement de la machine. **A un moment** est la partie importante car en effet un signal peut ne plus être initialisé dans le thread en cours (par exemple en terminant un *signal s in t*) mais avoir été émit avant. Donc pour gagner de l'espace je préfère un booléen à la place d'une liste de valeurs en double.

- les opérateurs **put** et **get**, qui eux une fois les points précédemment expliqués seront implémentés, rajouteront deux règles mais de provoquera pas de gros changement.

Seul le **get** va être un peu plus difficile car on a la contrainte de ne pouvoir prendre qu'une fois une valeur dans une liste. On va donc devoir rajouter dans la liste des valeurs partagées que l'on va définir plus loin une liste d'identifiant à chaque élément de la liste de valeurs pour simuler un pointeur.

Une 4ème version a donc été écrit sans les put et get et peut être retrouvé dans l'Annexe. La définition des éléments de la machine y sont aussi.

2.2.4 Semaine du 22 au 16

2.3 Mai

2.3.1 Semaine du 29 Avril au 3

2.3.2 Semaine du 6 au 10

2.3.3 Semaine du 13 au 17

2.3.4 Semaine du 20 au 24

3 Recherche d'Informations

3.1 Le Réactive ML

Modèle de programmation \rightarrow concurrence coopérative

L'analyse est découpé en 2 sous analyse :

- **statique** : système de type et d'effet
- **réactive** : détecter les erreurs de concurrence

Ordonnancement coopératif* \rightarrow *chaque processus va régulièrement "laisser la main aux autres"*

Ordonnancement préemptif \rightarrow *le système va "donner un temps de parole" à chacun*

Points fort :

- *implémentation séquentielle efficace*
- *pas de problème de parallélisme*

Points faible :

- *responsabilité de la réactivité au dev*

Le modèle réactif synchrone définit une notion de temps logique qui est une succession d'instant.

Un programme est réactif si son exécution fait progresser les instants logique.

Exemple

1. let process clock timer s =
2. let time = ref(Unix.gettimeofday()) in
3. loop
4. let time' = Unix.gettimeofday() in
5. if time' -. !time >= timer
6. then(emit s(); time := time')
7. end

Le problème ici est que le contenu de la boucle peut s'effectuer instantanément or il faut attendre un instant logique pour. Du coup, on doit ajouter une **pause entre la ligne 6 et 7**.

Une **condition suffisante** pour q'un **processus récursif soit réactif** est qu'il ait **toujours** au moins **un instant logique** entre l'instanciation du processus et l'appel récursif.

Définition des termes $k := \bullet \mid \circ \mid \phi \mid k \parallel k \mid k + k \mid k ; k \mid \mu\phi.k \mid \text{run } k \mid k^{oo}$

3.2 Les λ -calculs

3.2.1 Les règles de β -réduction

- $X_1[X_1 \leftarrow M] = M$
- $X_2[X_1 \leftarrow M] = X_2$
où $X_1 \neq X_2$
- $(\lambda X_1.M_1)[X_1 \leftarrow M_2] = (\lambda X_1.M_1)$
- $(\lambda X_1.M_1)[X_2 \leftarrow M_2] = (\lambda X_3.M_1[X_1 \leftarrow X_3])[X_2 \leftarrow M_2]$
où $X_1 \neq X_2$, $X_3 \notin FV(M_2)$ et $X_3 \notin FV(M_1) \setminus X_1$
- $(M_1 M_2)[X \leftarrow M_3] = (M_1[X \leftarrow M_3] M_2[X \leftarrow M_3])$

3.2.2 Les règles de réduction générale

- $(\lambda X_1.M) \alpha (\lambda X_1.M[X_1 \leftarrow X_2])$ où $X_2 \notin FV(M)$
- $((\lambda X_1.M_1)M_2) \beta M_1[X \leftarrow M_2]$
- $(\lambda X.(M X)) \eta M$ où $X \notin FV(M)$

La réduction générale $\mathbf{n} = \alpha \cup \beta \cup \eta$.

3.2.3 Les règles de priorité

- Application associative à gauche : $M_1 M_2 M_3 = ((M_1 M_2)M_3)$
- Application prioritaire par rapport au abstraction : $\lambda X.M_1 M_2 = \lambda X.(M_1 M_2)$
- Les abstractions consécutives peuvent être regroupé : $\lambda XYZ.M = (\lambda X.(\lambda Y.(\lambda Z.M)))$

3.2.4 Comment savoir si on a une forme normal ?

Une expression est une forme normale si on ne peut pas réduire l'expression via une β ou η réduction.

Théorème de la forme normale : Si on peut réduire L tels que $L =_n M$ et $L =_n N$ et que N et M sont en forme normal alors $M = N$ à n renommage près.

Théorème de Church-Rosser (pour $=_n$) : Si on a $M =_n N$, alors il existe un L' tels que $M \rightarrow_n L'$ et $N \rightarrow_n L'$.

Certaines lambda calcul expression n'a pas de forme normal comme : $((\lambda x.x x) (\lambda x.x x))$.

D'autres en ont une mais on peut rentrer dans une boucle infini de réduction si on choisi la mauvaise réduction.

Le problème est quand on évalue un argument de la fonction qui n'est jamais utilisé. Pour palier à ça, on utilise la stratégie d'appliquer toujours les β et η réduction le plus à gauche. Ces règles sont les suivantes :

- $M \rightarrow_{\bar{n}} N$ if $M \beta N$
- $M \rightarrow_{\bar{n}} N$ if $M \eta N$
- $(\lambda X.M) \rightarrow_{\bar{n}} (\lambda X.N)$
- $(M N) \rightarrow_{\bar{n}} (M' N)$ if $M \rightarrow_{\bar{n}} M'$
et $\forall L, (M N) \beta L$ impossible et $(M N) \eta L$ impossible
- $(M N) \rightarrow_{\bar{n}} (M N')$ if $N \rightarrow_{\bar{n}} N'$
et M est une forme normale
et $\forall L, (M N) \beta L$ impossible et $(M N) \eta L$ impossible

Cette solution est sûr mais reste peut utilisé car elle est assez lente.

3.3 ISWIM

ISWIM à une grammaire étendue de la grammaire des λ -calcul.

$M, N, L, K =$

| X (les variables)

| $(\lambda X.M)$

| $(M M)$

| b (les constantes b)

| $(o^n M \dots M)$ avec o^n les fonctions primitives

$V, U, W =$

| b

| X

| $(\lambda X.M)$

Une application n'est jamais une valeur alors qu'une abstraction est toujours une valeur.

Les règles de β -réductions sont les mêmes que celle pour les λ -calcul avec 2 ajouts :

— $b[X \leftarrow M] = b$

— $(o^n M_1 \dots M_n)[X \leftarrow M] = (o^n M_1[X \leftarrow M] \dots M_n[X \leftarrow M])$

La réduction est la même quand lambda calcul mais on vérifie juste que la réduction est faite avec une valeur. $((\lambda X.M) V) \beta_v M[X \leftarrow V]$. Cette restriction permet une sorte d'ordre dans les calculs.

η et α réduction ne sont plus vue comme tel. L' η -réduction n'est pas utilisé et l' α -réduction sera utilisé que pour chercher une équivalence entre deux termes.

Cependant on ajoute une δ -réduction en plus qui va s'occuper de gérer les réductions avec opérations.

3.4 Les différentes machines traitées

3.4.1 CC Machine

CC vient des termes **Control string** et **Context** qui représente respectivement :

- la partie du λ -calcul que l'on traite
- la partie du λ -calcul que l'on met en attente

Elle utilise le langage ISWIM.

Les règles définies pour cette machine sont les suivantes :

1. $\langle (M N), E \rangle \mapsto_{cc} \langle M, E[(\square N)] \rangle$ si $M \notin V$
2. $\langle (V_1 N), E \rangle \mapsto_{cc} \langle M, E[(V_1 \square)] \rangle$ si $M \notin V$
3. $\langle (o^n V_1 \dots V_i M N \dots), E \rangle \mapsto_{cc} \langle M, E[(o^n V_1 \dots V_i \square N \dots)] \rangle$ si $M \notin V$
4. $\langle ((\lambda X.M)V), E \rangle \mapsto_{cc} \langle M[X \leftarrow V], E \rangle$
5. $\langle (o^n b_1 \dots b_n), E \rangle \mapsto_{cc} \langle V, E \rangle$ avec $V = \delta(o^n, b_1 \dots b_n)$
6. $\langle V, E[(U \square)] \rangle \mapsto_{cc} \langle (U V), E \rangle$
7. $\langle V, E[(\square N)] \rangle \mapsto_{cc} \langle (V N), E \rangle$
8. $\langle V, E[(o^n V_1 \dots V_i \square N \dots)] \rangle \mapsto_{cc} \langle (o^n V_1 \dots V_i V N \dots), E \rangle$

La machine peut s'arrêter dans 3 états différents :

- on a une **constante** b tels que $\langle M, \square \rangle \rightarrow_{cc} \langle b, \square \rangle$;
- on a une **abstraction function** tels que $\langle M, \square \rangle \rightarrow_{cc} \langle \lambda X.N, \square \rangle$;
- on a un **état inconnu** soit une **erreur**.

Un exemple de la machine CC est fait dans les Annexes.

3.4.2 SCC Machine

Le SCC est une simplification de règle du CC. En effet, le CC exploite uniquement les informations de la chaîne de contrôle (Control string). Du coup on combine certaines règles pour en faire qu'une.

Les règles qui définissent la machine SCC sont les suivantes :

1. $\langle (M N), E \rangle \mapsto_{scc} \langle M, E[(\square N)] \rangle$
2. $\langle (o^n M N \dots), E \rangle \mapsto_{scc} \langle M, E[(o^n \square N \dots)] \rangle$
3. $\langle V, E[(\lambda X.M \square)] \rangle \mapsto_{scc} \langle M[X \leftarrow V], E \rangle$
4. $\langle V, E[(\square N)] \rangle \mapsto_{scc} \langle N, E[(V \square)] \rangle$
5. $\langle b, E[(o^n, b_1, \dots, b_i, \square)] \rangle \mapsto_{scc} \langle V, E \rangle$ avec $\delta(o^n, b_1, \dots, b_i, b) = V$
6. $\langle V, E[(o^n, V_1, \dots, V_i, \square, N L)] \rangle \mapsto_{scc} \langle N, E[(o^n, V_1, \dots, V_i, V, \square, L)] \rangle$

De même que pour la machine CC, la machine SCC peut s'arrêter dans 3 états différents :

- on a une **constante** b tels que $\langle M, \square \rangle \rightarrow_{scc} \langle b, \square \rangle$;
- on a une **abstraction function** tels que $\langle M, \square \rangle \rightarrow_{scc} \langle \lambda X.N, \square \rangle$;
- on a un **état inconnu** soit une **erreur**.

Un exemple de la machine SCC est fait dans les Annexes.

3.4.3 CK Machine

Les machines CC et SCC fonctionnent en allant chercher le plus à l'intérieur, c'est-à-dire que si l'on a une application on va en créer une intermédiaire dans le contexte avec un trou et traité la partie gauche de cette application etc jusqu'à arriver à un état traitable pour pouvoir "reconstruire", en reprenant l'application intermédiaire. C'est le style **LIFO (Last In, First Out)**. Ce qui fait que les étapes de transition dépendent directement de la forme du 1ère élément et non de la structure générale.

Pour palier à ce problème, la machine CK ajoute un nouvelle élément le **registre de contexte d'évaluation**, nommé κ , qui garde la partie "le plus à l'intérieur" accessible facilement.

$$\begin{aligned} \kappa = & \text{mt} \\ & | \langle \text{fun}, V, \kappa \rangle \\ & | \langle \text{arg}, N, \kappa \rangle \\ & | \langle \text{opd}, \langle V, \dots, V, o^n \rangle, \langle N, \dots \rangle, \kappa \rangle \end{aligned}$$

Cette structure est nommé **la continuation**.

Les règles qui définisse la machine CK sont les suivantes :

1. $\langle (M \ N), \kappa \rangle \mapsto_{ck} \langle M, \langle \text{arg}, N, \kappa \rangle \rangle$
2. $\langle (o^n \ M \ N \dots), \kappa \rangle \mapsto_{ck} \langle M, \langle \text{opd}, \langle o^n \rangle, \langle N, \dots \rangle, \kappa \rangle \rangle$
3. $\langle V, \langle \text{fun}, (\lambda X.M), \kappa \rangle \rangle \mapsto_{ck} \langle M[X \leftarrow V], \kappa \rangle$
4. $\langle V, \langle \text{arg}, N, \kappa \rangle \rangle \mapsto_{ck} \langle N, \langle \text{fun}, V, \kappa \rangle \rangle$
5. $\langle b, \langle \text{opd}, \langle b_i, \dots, b_1, o^n \rangle, \langle \rangle, \kappa \rangle \rangle \mapsto_{ck} \langle V, \kappa \rangle$ avec $\delta(o^n, b_1, \dots, b_i, b) = V$
6. $\langle V, \langle \text{opd}, \langle V', \dots, o^n \rangle, \langle N, L, \dots \rangle, \kappa \rangle \rangle \mapsto_{ck} \langle N, \langle \text{opd}, \langle V, V', \dots, o^n \rangle, \langle L, \dots \rangle, \kappa \rangle \rangle$

la machine CK peut s'arrêter dans 3 états différents :

- on a une **constante b** tels que $\langle M, mt \rangle \rightarrow_{ck} \langle b, mt \rangle$;
- on a une **abstraction function** tels que $\langle M, mt \rangle \rightarrow_{ck} \langle \lambda X.N, mt \rangle$;
- on a un **état inconnu** soit une **erreur**.

Un exemple de la machine CK est fait dans les Annexes.

3.4.4 CEK Machine

Pour toutes les machines vues pour l'instant la β -réduction était appliquée immédiatement. Cela coûte cher surtout quand l'expression devient grande. De plus, si notre substitution n'est pas une variable elle est traité avant d'être appliqué.

Il est plus intéressant d'appliquer les substitutions quand on en a vraiment la nécessité. Pour cela, la machine CEK ajoute les clauses et un environnement ε qui va stocker les substitutions à faire.

On a alors :

$$\begin{aligned} \varepsilon = & \text{une fonction } \{\langle X, c \rangle, \dots\} \text{ c } = \{\langle M, \varepsilon \rangle \mid FV(M) \subset \text{dom}(\varepsilon)\} \text{ v } = \{\langle V, \varepsilon \rangle \mid \langle V, \varepsilon \rangle \in c\} \\ \varepsilon[X \leftarrow c] = & \{\langle X, c \rangle\} \cup \{\langle Y, c' \rangle \mid \langle Y, c' \rangle \in \varepsilon \text{ et } Y \neq X\} \end{aligned}$$

κ est renommé $\bar{\kappa}$ et est définit par :

$$\begin{aligned} \bar{\kappa} = & \text{mt} \\ & | \langle \text{fun}, v, \bar{\kappa} \rangle \\ & | \langle \text{arg}, c, \bar{\kappa} \rangle \\ & | \langle \text{opd}, \langle v, \dots, v, o^n \rangle, \langle c, \dots \rangle, \bar{\kappa} \rangle \end{aligned}$$

Les règles qui définisse la machine CEK sont les suivantes :

1. $\langle \langle (M \ N), \varepsilon \rangle, \bar{\kappa} \rangle \mapsto_{cek} \langle \langle M, \varepsilon \rangle, \langle \text{arg}, \langle N, \varepsilon \rangle, \bar{\kappa} \rangle \rangle$
2. $\langle \langle (o^n \ M \ N \dots), \varepsilon \rangle, \bar{\kappa} \rangle \mapsto_{cek} \langle \langle M, \varepsilon \rangle, \langle \text{opd}, \langle o^n \rangle, \langle \langle N, \varepsilon \rangle, \dots \rangle, \bar{\kappa} \rangle \rangle$
3. $\langle \langle V, \varepsilon \rangle, \langle \text{fun}, (\lambda X1.M), \varepsilon' \rangle, \bar{\kappa} \rangle \mapsto_{cek} \langle \langle M, \varepsilon'[X1 \leftarrow \langle V, \varepsilon \rangle] \rangle, \bar{\kappa} \rangle$ si $V \notin X$
4. $\langle \langle V, \varepsilon \rangle, \langle \text{arg}, \langle N, \varepsilon' \rangle, \kappa \rangle \rangle \mapsto_{cek} \langle \langle N, \varepsilon' \rangle, \langle \text{fun}, \langle V, \varepsilon \rangle, \bar{\kappa} \rangle \rangle$ si $V \notin X$
5. $\langle \langle b, \varepsilon \rangle, \langle \text{opd}, \langle \langle b_i, \varepsilon_i \rangle, \dots, \langle b_1, \varepsilon_1 \rangle, o^n \rangle, \langle \rangle, \bar{\kappa} \rangle \rangle \mapsto_{cek} \langle \langle V, \emptyset \rangle, \bar{\kappa} \rangle$ avec $\delta(o^n, b_1, \dots, b_i, b) = V$
6. $\langle \langle V, \varepsilon \rangle, \langle \text{opd}, \langle v', \dots, o^n \rangle, \langle \langle N, \varepsilon' \rangle, c, \dots \rangle, \bar{\kappa} \rangle \rangle \mapsto_{cek} \langle \langle N, \varepsilon' \rangle, \langle \text{opd}, \langle \langle V, \varepsilon \rangle, v', \dots, o^n \rangle, \langle c, \dots \rangle, \bar{\kappa} \rangle \rangle$ si $V \notin X$
7. $\langle \langle X, \varepsilon \rangle, \bar{\kappa} \rangle \mapsto_{cek} \langle c, \bar{\kappa} \rangle$ avec $\varepsilon(X) = c$

la machine CEK peut s'arrêter dans 3 états différents :

- on a une **constante b** tels que $\langle \langle M, \emptyset \rangle, mt \rangle \rightarrow_{cek} \langle \langle b, \varepsilon \rangle, mt \rangle$;
- on a une **abstraction function** tels que $\langle \langle M, \emptyset \rangle, mt \rangle \rightarrow_{cek} \langle \langle \lambda X.N, \varepsilon \rangle, mt \rangle$;
- on a un **état inconnu** soit une **erreur**.

Un exemple de la machine CEK est fait dans les Annexes.

3.4.5 SECD Machine

La différence entre la machine CEK et SECD est la façon dont le contexte est sauvegarder pendant que le sous-expressions sont évaluées.

En effet, dans la machine SECD le contexte est créer par un appel de fonction, quand toute est stocké dans \widehat{D} pour laisser un espace de travail. Par contre pour la machine CEK, le contexte est créé quant on évalue une application ou un argument indépendamment de la complexité de celui-ci.

Dans les langages tels que Java, Pascal ou encore C la façon de faire de la machine SECD est plus naturel. Par contre dans les langages λ -calculs, Scheme ou encore ML c'est la façon de faire de la machine CEK qui est la plus naturel.

La machine SECD est composé d'une pile (\widehat{S}), d'un environnement ($\widehat{\varepsilon}$), d'une chaîne de contrôle (\widehat{C}) et d'une sauvegarde (\widehat{D}). Les différentes définitions de ces élément sont les suivantes :

$$\begin{aligned}\widehat{S} &= \epsilon \mid \widehat{V} \widehat{S} \\ \widehat{\varepsilon} &= \text{une fonction } \{\langle X, \widehat{V} \rangle, \dots\} \\ \widehat{C} &= \epsilon \mid b \widehat{C} \mid X \widehat{C} \mid \text{ap } \widehat{C} \mid \text{prim}_{o^n} \widehat{C} \mid \langle X, \widehat{C} \rangle \widehat{C} \\ \widehat{D} &= \epsilon \mid \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D} \rangle \\ \widehat{V} &= b \mid \langle \langle X, \widehat{C} \rangle, \widehat{\varepsilon} \rangle \\ [b]_{secd} &= b \\ [X]_{secd} &= X \\ [(M_1 M_2)]_{secd} &= [M_1]_{secd} [M_2]_{secd} \text{ap} \\ [(o^n M_1 \dots M_n)]_{secd} &= [M_1]_{secd} \dots [M_n]_{secd} \text{prim}_{o^n} \\ [(\lambda X.M)]_{secd} &= \langle X, [M]_{secd} \rangle\end{aligned}$$

Les règles qui définisse la machine SECD sont les suivantes :

1. $\langle \widehat{S}, \widehat{\varepsilon}, b \widehat{C}, \widehat{D} \rangle \mapsto_{secd} \langle b \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D} \rangle$
2. $\langle \widehat{S}, \widehat{\varepsilon}, X \widehat{C}, \widehat{D} \rangle \mapsto_{secd} \langle \widehat{V} \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D} \rangle$ où $\widehat{V} = \varepsilon(X)$
3. $\langle b_1 \dots b_n \widehat{S}, \widehat{\varepsilon}, \text{prim}_{o^n} \widehat{C}, \widehat{D} \rangle \mapsto_{secd} \langle \widehat{V} \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D} \rangle$ où $\widehat{V} = \delta(o^n, b_1, \dots, b_n)$
4. $\langle \widehat{S}, \widehat{\varepsilon}, \langle X, C' \rangle \widehat{C}, \widehat{D} \rangle \mapsto_{secd} \langle \langle \langle X, C' \rangle, \varepsilon \rangle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D} \rangle$
5. $\langle \widehat{V} \langle \langle X, C' \rangle, \varepsilon' \rangle \widehat{S}, \widehat{\varepsilon}, \text{ap } \widehat{C}, \widehat{D} \rangle \mapsto_{secd} \langle \varepsilon, \varepsilon' [X \leftarrow \widehat{V}], C', \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D} \rangle \rangle$
6. $\langle \widehat{V} \widehat{S}, \widehat{\varepsilon}, \emptyset, \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C}', \widehat{D} \rangle \rangle \mapsto_{secd} \langle \widehat{V} \widehat{S}', \widehat{\varepsilon}', \widehat{C}', \widehat{D} \rangle$

la machine SECD peut s'arrêter dans 3 états différents :

- on a une **constante** b tels que $\langle \epsilon, \emptyset, [M]_{secd}, \epsilon \rangle \twoheadrightarrow_{secd} \langle b, \widehat{\varepsilon}, \epsilon, \epsilon \rangle$;
- on a une **abstraction function** tels que $\langle \epsilon, \emptyset, [M]_{secd}, \epsilon \rangle \twoheadrightarrow_{secd} \langle \langle \langle X, \widehat{C} \rangle, \widehat{\varepsilon}' \rangle, \widehat{\varepsilon}, \epsilon, \epsilon \rangle$;
- on a un **état inconnu** soit une **erreur**.

Un exemple de la machine SECD est fait dans les Annexes.

4 Conclusion

5 Annexes

5.1 Les Exemples des machines étudiées

5.1.1 Exemple de fonctionnement de la machine CC

Voici un exemple de fonctionnement de la machine CC :

CC machine : $\langle (((\lambda f.\lambda x.f\ x)\ \lambda y.(+ y\ y))\ \ulcorner 1^\urcorner), [] \rangle$
 $> \langle (M\ N), E \rangle \mapsto_{cc} \langle M, E[([]\ N)] \rangle$ si $M \notin V$
CC machine : $\langle ((\lambda f.\lambda x.f\ x)\ \lambda y.(+ y\ y)), [([]\ \ulcorner 1^\urcorner)] \rangle$
 $> \langle ((\lambda X.M)V), E \rangle \mapsto_{cc} \langle M[X \leftarrow V], E \rangle$
CC machine : $\langle (\lambda x.f\ x)[f \leftarrow \lambda y.(+ y\ y)], [([]\ \ulcorner 1^\urcorner)] \rangle$
CC machine : $\langle (\lambda x.(\lambda y.(+ y\ y))\ x), [([]\ \ulcorner 1^\urcorner)] \rangle$
 $> \langle V, E[([]\ N)] \rangle \mapsto_{cc} \langle (V\ N), E \rangle$
CC machine : $\langle ((\lambda x.(\lambda y.(+ y\ y))\ x)\ \ulcorner 1^\urcorner), [] \rangle$
 $> \langle ((\lambda X.M)V), E \rangle \mapsto_{cc} \langle M[X \leftarrow V], E \rangle$
CC machine : $\langle ((\lambda y.(+ y\ y))\ x)[x \leftarrow \ulcorner 1^\urcorner], [] \rangle$
CC machine : $\langle ((\lambda y.(+ y\ y))\ \ulcorner 1^\urcorner), [] \rangle$
 $> \langle ((\lambda X.M)V), E \rangle \mapsto_{cc} \langle M[X \leftarrow V], E \rangle$
CC machine : $\langle (+ y\ y)[y \leftarrow \ulcorner 1^\urcorner], [] \rangle$
CC machine : $\langle (+ \ulcorner 1^\urcorner\ \ulcorner 1^\urcorner), [] \rangle$
 $> \langle (o^n\ b_1...b_n), E \rangle \mapsto_{cc} \langle V, E \rangle$ avec $V = \delta(o^n, b_1...b_n)$
CC machine : $\langle \ulcorner 2^\urcorner, [] \rangle$

5.1.2 Exemple de fonctionnement de la machine SCC

Voici un exemple de fonctionnement de la machine SCC :

SCC machine : $\langle (((\lambda f.\lambda x.f\ x)\ \lambda y.(+ y\ y))\ \ulcorner 1^\urcorner), [] \rangle$
 $> \langle (M\ N), E \rangle \mapsto_{scc} \langle M, E[([]\ N)] \rangle$
SCC machine : $\langle ((\lambda f.\lambda x.f\ x)\ \lambda y.(+ y\ y)), [([]\ \ulcorner 1^\urcorner)] \rangle$
 $> \langle (M\ N), E \rangle \mapsto_{scc} \langle M, E[([]\ N)] \rangle$
SCC machine : $\langle (\lambda f.\lambda x.f\ x), [([]\ \ulcorner 1^\urcorner), ([]\ (\lambda y.(+ y\ y)))] \rangle$
 $> \langle V, E[([]\ N)] \rangle \mapsto_{scc} \langle N, E[(V\ [])] \rangle$
SCC machine : $\langle (\lambda y.(+ y\ y)), [([]\ \ulcorner 1^\urcorner), ((\lambda f.\lambda x.f\ x)\ [])] \rangle$
 $> \langle V, E[(\lambda X.M)\ []] \rangle \mapsto_{scc} \langle M[X \leftarrow V], E \rangle$
SCC machine : $\langle (\lambda x.f\ x)[f \leftarrow (\lambda y.(+ y\ y))], [([]\ \ulcorner 1^\urcorner)] \rangle$
SCC machine : $\langle (\lambda x.(\lambda y.(+ y\ y))\ x), [([]\ \ulcorner 1^\urcorner)] \rangle$
 $> \langle V, E[([]\ N)] \rangle \mapsto_{scc} \langle N, E[(V\ [])] \rangle$
SCC machine : $\langle \ulcorner 1^\urcorner, [(\lambda x.(\lambda y.(+ y\ y))\ x)\ []] \rangle$
 $> \langle V, E[(\lambda X.M)\ []] \rangle \mapsto_{scc} \langle M[X \leftarrow V], E \rangle$
SCC machine : $\langle ((\lambda y.(+ y\ y))\ x)[x \leftarrow \ulcorner 1^\urcorner], [] \rangle$
SCC machine : $\langle ((\lambda y.(+ y\ y))\ \ulcorner 1^\urcorner), [] \rangle$
 $> \langle (M\ N), E \rangle \mapsto_{scc} \langle M, E[([]\ N)] \rangle$
SCC machine : $\langle (\lambda y.(+ y\ y)), [([]\ \ulcorner 1^\urcorner)] \rangle$
 $> \langle V, E[([]\ N)] \rangle \mapsto_{scc} \langle N, E[(V\ [])] \rangle$
SCC machine : $\langle \ulcorner 1^\urcorner, [(\lambda y.(+ y\ y))\ []] \rangle$
 $> \langle V, E[(\lambda X.M)\ []] \rangle \mapsto_{scc} \langle M[X \leftarrow V], E \rangle$
SCC machine : $\langle (+ y\ y)[y \leftarrow \ulcorner 1^\urcorner], [] \rangle$
SCC machine : $\langle (+ \ulcorner 1^\urcorner\ \ulcorner 1^\urcorner), [] \rangle$
 $> \langle (o^n\ M\ N...), E \rangle \mapsto_{scc} \langle M, E[(o^n\ []\ N...)] \rangle$
SCC machine : $\langle \ulcorner 1^\urcorner, (+\ []\ \ulcorner 1^\urcorner) \rangle$
 $> \langle V, E[(o^n, V_1, ...V_i, [], N\ L)] \rangle \mapsto_{scc} \langle N, E[(o^n, V_1, ...V_i, V, [], L)] \rangle$
SCC machine : $\langle \ulcorner 1^\urcorner, (+\ \ulcorner 1^\urcorner\ []) \rangle$
 $\langle b, E[(o^n, b_1, ...b_i, [])] \rangle \mapsto_{scc} \langle V, E \rangle$ avec $\delta(o^n, b_1, ...b_i, b) = V$
SCC machine : $\langle \ulcorner 2^\urcorner, [] \rangle$

5.1.3 Exemple de fonctionnement de la machine CK

Voici un exemple de fonctionnement de la machine CK :

CK machine : $\langle ((\lambda f.\lambda x.f\ x)\ \lambda y.(+ y\ y))\ \lceil 1^\top \rceil, mt \rangle$
 $> \langle (M\ N), \kappa \rangle \mapsto_{ck} \langle M, \langle arg, N, \kappa \rangle \rangle$
 CK machine : $\langle ((\lambda f.\lambda x.f\ x)\ \lambda y.(+ y\ y)), \langle arg, \lceil 1^\top \rceil, mt \rangle \rangle$
 $> \langle (M\ N), \kappa \rangle \mapsto_{ck} \langle M, \langle arg, N, \kappa \rangle \rangle$
 CK machine : $\langle (\lambda f.\lambda x.f\ x), \langle arg, (\lambda y.(+ y\ y)), \langle arg, \lceil 1^\top \rceil, mt \rangle \rangle \rangle$
 $> \langle V, \langle arg, N, \kappa \rangle \rangle \mapsto_{ck} \langle N, \langle fun, V, \kappa \rangle \rangle$
 CK machine : $\langle (\lambda y.(+ y\ y)), \langle fun, (\lambda f.\lambda x.f\ x), \langle arg, \lceil 1^\top \rceil, mt \rangle \rangle \rangle$
 $> \langle V, \langle fun, (\lambda X.M), \kappa \rangle \rangle \mapsto_{ck} \langle M[X \leftarrow V], \kappa \rangle$
 CK machine : $\langle (\lambda x.f\ x)[f \leftarrow (\lambda y.(+ y\ y))], \langle arg, \lceil 1^\top \rceil, mt \rangle \rangle$
 CK machine : $\langle (\lambda x.(\lambda y.(+ y\ y))\ x), \langle arg, \lceil 1^\top \rceil, mt \rangle \rangle$
 $> \langle V, \langle arg, N, \kappa \rangle \rangle \mapsto_{ck} \langle N, \langle fun, V, \kappa \rangle \rangle$
 CK machine : $\langle \lceil 1^\top \rceil, \langle fun, (\lambda x.(\lambda y.(+ y\ y))\ x), mt \rangle \rangle$
 $> \langle V, \langle fun, (\lambda X.M), \kappa \rangle \rangle \mapsto_{ck} \langle M[X \leftarrow V], \kappa \rangle$
 CK machine : $\langle ((\lambda y.(+ y\ y))\ x)[x \leftarrow \lceil 1^\top \rceil], mt \rangle$
 CK machine : $\langle ((\lambda y.(+ y\ y))\ \lceil 1^\top \rceil), mt \rangle$
 $> \langle (M\ N), \kappa \rangle \mapsto_{ck} \langle M, \langle arg, N, \kappa \rangle \rangle$
 CK machine : $\langle (\lambda y.(+ y\ y)), \langle arg, \lceil 1^\top \rceil, mt \rangle \rangle$
 $> \langle V, \langle arg, N, \kappa \rangle \rangle \mapsto_{ck} \langle N, \langle fun, V, \kappa \rangle \rangle$
 CK machine : $\langle \lceil 1^\top \rceil, \langle fun, (\lambda y.(+ y\ y)), mt \rangle \rangle$
 $> \langle V, \langle fun, (\lambda X.M), \kappa \rangle \rangle \mapsto_{ck} \langle M[X \leftarrow V], \kappa \rangle$
 CK machine : $\langle (+ y\ y)[y \leftarrow \lceil 1^\top \rceil], mt \rangle$
 CK machine : $\langle (+ \lceil 1^\top \rceil \lceil 1^\top \rceil), mt \rangle$
 $> \langle (o^n\ M\ N\dots), \kappa \rangle \mapsto_{ck} \langle M, \langle opd, \langle o^n \rangle, \langle N, \dots \rangle, \kappa \rangle \rangle$
 CK machine : $\langle \lceil 1^\top \rceil, \langle opd, \langle + \rangle, \langle \lceil 1^\top \rceil \rangle, mt \rangle \rangle$
 $> \langle V, \langle opd, \langle V', \dots o^n \rangle, \langle N, L, \dots \rangle, \kappa \rangle \rangle \mapsto_{ck} \langle N, \langle opd, \langle V, V', \dots o^n \rangle, \langle L, \dots \rangle, \kappa \rangle \rangle$
 CK machine : $\langle \lceil 1^\top \rceil, \langle opd, \langle \lceil 1^\top \rceil, + \rangle, \langle \rangle, mt \rangle \rangle$
 $> \langle b, \langle opd, \langle b_i, \dots b_1, o^n \rangle, \langle \rangle, \kappa \rangle \rangle \mapsto_{ck} \langle V, \kappa \rangle$ avec $\delta(o^n, b_1, \dots b_i, b) = V$
 CK machine : $\langle \lceil 2^\top \rceil, mt \rangle$

5.1.4 Exemple de fonctionnement de la machine CEK

Voici un exemple de fonctionnement de la machine CEK :

[illegible]

5.1.5 Exemple de fonctionnement de la machine SECD

Voici un exemple de fonctionnement de la machine SECD :

Conversion : $[(((\lambda f.\lambda x.f\ x)\ \lambda y.(+ \ y\ y))\ \lceil 1 \rceil)]_{secd}$

Conversion : $[(\lambda f.\lambda x.f\ x)\ \lambda y.(+ \ y\ y)]_{secd} \ [\ulcorner 1 \urcorner]_{secd} \ ap$

Conversion : $[(\lambda f. \lambda x. f \ x)]_{secd} \ [\lambda y. (+ \ y \ y)]_{secd} \ ap \ \ulcorner 1 \urcorner \ ap$

Conversion : $\langle f, [\lambda x. (f \ x)]_{sec d} \rangle \langle y, y \ y \ prim_+ \rangle \text{ ap } \lceil 1 \rceil \text{ ap}$

Conversion : $\langle f, \langle x, [f \ x]_{secd} \rangle \rangle \langle y, y \ y \ prim_+ \rangle \text{ ap } \ulcorner 1 \urcorner \text{ ap}$

Conversion : $\langle f, \langle x, f \ x \ ap \rangle \rangle \ \langle y, y \ y \ prim_+ \rangle \ ap \ \lceil 1 \rceil \ ap$

SECD Machine : $\langle \epsilon, \emptyset, \langle f, \langle x, f \ x \ ap \rangle \rangle \ \langle y, y \ y \ prim_+ \rangle \ ap \ulcorner 1 \urcorner \ ap, \epsilon \rangle$

$$> \langle \widehat{S}, \widehat{\varepsilon}, \langle X, C' \rangle \widehat{C}, \widehat{D} \rangle \mapsto_{secd} \langle \langle X, C' \rangle, \varepsilon \rangle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D} \rangle$$

SECD Machine : $\langle\langle\langle f, \langle x, f \ x \ ap \rangle \rangle, \emptyset \rangle, \emptyset, \langle y, y \ y \ prim_+ \rangle \ ap \ulcorner 1 \urcorner \ ap, \epsilon \rangle$

$$> \langle \hat{S}, \hat{\varepsilon}, \langle X, C' \rangle \hat{C}, \hat{D} \rangle \mapsto_{secd} \langle \langle X, C' \rangle, \varepsilon \rangle \hat{S}, \hat{\varepsilon}, \hat{C}, \hat{D} \rangle$$

SECD Machine : $\langle \langle \langle y, y \text{ } y \text{ } prim_+ \rangle, \emptyset \rangle \langle \langle f, \langle x, f \text{ } x \text{ } ap \rangle \rangle, \emptyset \rangle, \emptyset, ap \text{ } \ulcorner 1 \urcorner \text{ } ap, \epsilon \rangle$

$$> \langle \hat{V} \langle \langle X, C' \rangle, \varepsilon' \rangle \hat{S}, \hat{\varepsilon}, ap \hat{C}, \hat{D} \rangle \mapsto_{secd} \langle \epsilon, \varepsilon' [X \leftarrow \hat{V}], C', \langle \hat{S}, \hat{\varepsilon}, \hat{C}, \hat{D} \rangle \rangle$$
$$\text{SECD Machine : } \langle \epsilon, \emptyset[f \leftarrow \langle \langle y, y \text{ } y \text{ } prim_+ \rangle, \emptyset \rangle], \langle x, f \text{ } x \text{ } ap \rangle, \langle \epsilon, \emptyset, \lceil 1 \rceil \text{ } ap, \epsilon \rangle \rangle$$

SECD Machine : $\langle \epsilon, \{ \langle f, \langle \langle y, y \text{ } y \text{ } prim_+ \rangle, \emptyset \rangle \rangle \}, \langle x, f \text{ } x \text{ } ap \rangle, \langle \epsilon, \emptyset, \lceil 1 \rceil \text{ } ap, \epsilon \rangle \rangle$

$$> \langle \hat{S}, \hat{\varepsilon}, \langle X, C' \rangle \hat{C}, \hat{D} \rangle \mapsto_{secd} \langle \langle X, C' \rangle, \varepsilon \rangle \hat{S}, \hat{\varepsilon}, \hat{C}, \hat{D} \rangle$$

SECD Machine : $\langle \langle \langle x, f \ x \ ap \rangle, \{ \langle f, \langle \langle y, y \ y \ prim_+ \rangle, \emptyset \rangle \} \rangle, \{ f, \langle \langle y, y \ y \ prim_+ \rangle, \emptyset \rangle \}, \emptyset, \langle \epsilon, \emptyset, \lceil 1 \rceil \ ap, \epsilon \rangle \rangle$

$$> \langle \widehat{V} \ \widehat{S}, \widehat{\varepsilon}, \emptyset, \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C}', \widehat{D} \rangle \rangle \longmapsto_{secd} \langle \widehat{V} \ \widehat{S}', \widehat{\varepsilon}', \widehat{C}', \widehat{D} \rangle$$

SECD Machine : $\langle \langle \langle x, f \ x \ ap \rangle, \{ \langle f, \langle y, y \ y \ prim_+ \rangle, \emptyset \} \rangle \rangle, \emptyset, \lceil 1 \rceil \ ap, \epsilon \rangle$

$$> \langle \hat{S}, \hat{\varepsilon}, b \hat{C}, \hat{D} \rangle \mapsto_{sec d} \langle b \hat{S}, \hat{\varepsilon}, \hat{C}, \hat{D} \rangle$$

SECD Machine : $\langle \lceil 1 \rceil \langle \langle x, f \ x \ ap \rangle, \{ \langle f, \langle \langle y, y \ y \ prim_+ \rangle, \emptyset \rangle \} \rangle, \emptyset, ap, \epsilon \rangle$

$$> \langle \hat{V} \langle \langle X, C' \rangle, \varepsilon' \rangle \hat{S}, \hat{\varepsilon}, ap \hat{C}, \hat{D} \rangle \mapsto_{\text{secd}} \langle \epsilon, \varepsilon' [X \leftarrow \hat{V}], C', \langle \hat{S}, \hat{\varepsilon}, \hat{C}, \hat{D} \rangle \rangle$$

SECD Machine : $\langle \epsilon, \{ \langle f, \langle \langle y, y \text{ } y \text{ } prim_+ \rangle, \emptyset \rangle \} [x \leftarrow \ulcorner 1 \urcorner], f \text{ } x \text{ } ap, \langle \epsilon, \emptyset, \epsilon, \epsilon \rangle \rangle$

SECD Machine : $\langle \epsilon, \{ \langle f, \langle \langle y, y \text{ } y \text{ } prim_+ \rangle, \emptyset \rangle \rangle, \langle x, \lceil 1 \rceil \rangle \}, f \text{ } x \text{ } ap, \langle \epsilon, \emptyset, \epsilon, \epsilon \rangle \rangle$

$$> \langle \hat{S}, \hat{\varepsilon}, X \hat{C}, \hat{D} \rangle \mapsto_{sec d} \langle \hat{V} \hat{S}, \hat{\varepsilon}, \hat{C}, \hat{D} \rangle \text{ où } \hat{V} = \varepsilon(X)$$

SECD Machine : $\langle \langle \langle y, y \text{ } y \text{ } prim_+ \rangle, \emptyset \rangle, \{ \langle f, \langle \langle y, y \text{ } y \text{ } prim_+ \rangle, \emptyset \rangle \rangle, \langle x, \lceil 1 \rceil \rangle \}, x \text{ } ap, \langle \epsilon, \emptyset, \epsilon, \epsilon \rangle \rangle$

$$> \langle \hat{S}, \hat{\varepsilon}, X \hat{C}, \hat{D} \rangle \mapsto_{secd} \langle \hat{V} \hat{S}, \hat{\varepsilon}, \hat{C}, \hat{D} \rangle \text{ où } \hat{V} = \varepsilon(X)$$

SECD Machine : $\langle \ulcorner 1 \urcorner \langle \langle y, y \ y \ prim_+ \rangle, \emptyset \rangle, \{ \langle f, \langle \langle y, y \ y \ prim_+ \rangle, \emptyset \rangle \rangle, \langle x, \ulcorner 1 \urcorner \rangle \}, ap, \langle \epsilon, \emptyset, \epsilon, \epsilon \rangle \rangle$

$$> \langle \hat{V} \langle \langle X, C' \rangle, \epsilon' \rangle \hat{S}, \hat{\epsilon}, ap \hat{C}, \hat{D} \rangle \mapsto_{secd} \langle \epsilon, \epsilon' [X \leftarrow \hat{V}], C', \langle \hat{S}, \hat{\epsilon}, \hat{C}, \hat{D} \rangle \rangle$$

SECD Machine : $\langle \epsilon, \emptyset[y \leftarrow \ulcorner 1 \urcorner], y \ y \ prim_+, \langle \epsilon, \{ \langle f, \langle \langle y, y \ y \ prim_+ \rangle, \emptyset \rangle \}, \langle x, \ulcorner 1 \urcorner \rangle \}, \emptyset, \langle \epsilon, \emptyset, \epsilon, \epsilon \rangle \rangle$

SECD Machine : $\langle \epsilon, \{\langle y, \lceil 1 \rceil\} \rangle, y \ y \ prim_+, \langle \epsilon, \{\langle f, \langle \langle y, y \ y \ prim_+ \rangle, \emptyset \rangle\}, \langle x, \lceil 1 \rceil \rangle\}, \emptyset, \langle \epsilon, \emptyset, \epsilon, \epsilon \rangle \rangle$

$$> \langle \hat{S}, \hat{\varepsilon}, X \hat{C}, \hat{D} \rangle \mapsto_{secd} \langle \hat{V} \hat{S}, \hat{\varepsilon}, \hat{C}, \hat{D} \rangle \text{ où } \hat{V} = \varepsilon(X)$$

SECD Machine : $\langle \lceil 1 \rceil, \{\langle y, \lceil 1 \rceil \rangle\}, y \text{ prim}_+, \langle \epsilon, \{\langle f, \langle \langle y, y \text{ prim}_+ \rangle, \emptyset \rangle\}, \langle x, \lceil 1 \rceil \rangle\}, \emptyset, \langle \epsilon, \emptyset, \epsilon, \epsilon \rangle \rangle$

$$> \langle \hat{S}, \hat{\varepsilon}, X \hat{C}, \hat{D} \rangle \mapsto_{\text{secd}} \langle \hat{V} \hat{S}, \hat{\varepsilon}, \hat{C}, \hat{D} \rangle \text{ où } \hat{V} = \varepsilon(X)$$

SECD Machine : $\langle \lceil 1 \rceil \lceil 1 \rceil, \{\langle y, \lceil 1 \rceil \rangle\}, prim_+, \langle \epsilon, \{\langle f, \langle \langle y, y \ y \ prim_+ \rangle, \emptyset \rangle\}, \langle x, \lceil 1 \rceil \rangle\}, \emptyset, \langle \epsilon, \emptyset, \epsilon, \epsilon \rangle \rangle$

$$> \langle b_1 \dots b_n \hat{S}, \hat{\varepsilon}, \text{prim}_{o^n} \hat{C}, \hat{D} \rangle \mapsto_{\text{secd}} \langle \hat{V} \hat{S}, \hat{\varepsilon}, \hat{C}, \hat{D} \rangle \text{ où } \hat{V} = \delta(o^n, b_1, \dots, b_n)$$
$$\text{SECD Machine : } \langle \lceil 2 \rceil, \{ \langle y, \lceil 1 \rceil \}, \emptyset, \langle \epsilon, \{ \langle f, \langle \langle y, y \text{ } y \text{ } prim_+ \rangle, \emptyset \rangle \}, \langle x, \lceil 1 \rceil \rangle \}, \emptyset, \langle \epsilon, \emptyset, \epsilon, \epsilon \rangle \rangle$$
$$> \langle \widehat{V} \ \widehat{S}, \widehat{\varepsilon}, \emptyset, \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C}', \widehat{D} \rangle \rangle \mapsto_{secd} \langle \widehat{V} \ \widehat{S}', \widehat{\varepsilon}', \widehat{C}', \widehat{D} \rangle$$

SECD Machine : $\langle \lceil 2 \rceil, \{ \langle f, \langle \langle y, y \text{ } y \text{ } prim_+ \rangle, \emptyset \rangle \rangle, \langle x, \lceil 1 \rceil \rangle \}, \emptyset, \langle \epsilon, \emptyset, \epsilon, \epsilon \rangle \rangle$

$$> \langle \widehat{V} \ \widehat{S}, \widehat{\varepsilon}, \emptyset, \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C}', \widehat{D} \rangle \rangle \longmapsto_{\text{secd}} \langle \widehat{V} \ \widehat{S}', \widehat{\varepsilon}', \widehat{C}', \widehat{D} \rangle$$
SECD Machine : $\langle \ulcorner 2 \urcorner, \emptyset, \epsilon, \epsilon \rangle$

5.2 Les différentes versions faite pour rendre la machine SECD concurrente

5.2.1 1ère version des règles de la machine SECD Concurrente

Soit $\langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle$ avec :

$$\widehat{S} = \dots \mid \langle C', C'' \rangle \widehat{S} \mid \text{Remp } \widehat{S}$$

$$\widehat{\varepsilon} = \dots$$

$$\widehat{C} = \dots \mid \text{bspawn } \widehat{C} \mid \text{espawn } \widehat{C} \mid \langle C', C'' \rangle \widehat{C} \mid \text{present}_s \widehat{C} \mid \text{emit}_s \widehat{C}$$

$$\widehat{W} = \{\widehat{D}, \dots\}$$

$$\widehat{ST} = \{\langle s, \widehat{D} \rangle, \dots\}$$

$$\widehat{SI} = \{s, \dots\}$$

$$\widehat{D} = \dots$$

Les nouvelles règles sont les suivantes :

1. $\langle \widehat{S}, \widehat{\varepsilon}, b \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{\text{secdv1}} \langle b \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle$
2. $\langle \widehat{S}, \widehat{\varepsilon}, X \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{\text{secdv1}} \langle \widehat{V} \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle$ où $\widehat{V} = \varepsilon(X)$
3. $\langle b_1 \dots b_n \widehat{S}, \widehat{\varepsilon}, \text{prim}_{o^n} \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{\text{secdv1}} \langle \widehat{V} \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle$ où $\widehat{V} = \delta(o^n, b_1, \dots, b_n)$
4. $\langle \widehat{S}, \widehat{\varepsilon}, \langle X, C' \rangle \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{\text{secdv1}} \langle \langle \langle X, C' \rangle, \varepsilon \rangle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle$
5. $\langle \widehat{V} \langle \langle X, C' \rangle, \varepsilon' \rangle \widehat{S}, \widehat{\varepsilon}, \text{ap } \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{\text{secdv1}} \langle \varepsilon, \varepsilon' [X \leftarrow \widehat{V}], C', \widehat{W}, \widehat{ST}, \widehat{SI}, \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D} \rangle \rangle$
6. $\langle \widehat{V} \widehat{S}, \widehat{\varepsilon}, \emptyset, \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C}', \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \rangle \mapsto_{\text{secdv1}} \langle \widehat{V} \widehat{S}', \widehat{\varepsilon}', \widehat{C}', \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle$
7. $\langle \widehat{V} \text{Remp } \widehat{S}, \widehat{\varepsilon}, \text{ap } \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{\text{secdv1}} \langle \widehat{V} \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle$
8. $\langle \text{Remp } \widehat{V} \widehat{S}, \widehat{\varepsilon}, \text{ap } \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{\text{secdv1}} \langle \widehat{V} \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle$
9. $\langle \text{Remp } \widehat{V} \widehat{S}, \widehat{\varepsilon}, \text{ap } \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{\text{secdv1}} \langle \widehat{V} \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle$
10. $\langle \widehat{S}, \widehat{\varepsilon}, \text{bspawn } \widehat{C}' \text{ espawn } \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{\text{secdv1}} \langle \text{Remp } \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W} \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}', \widehat{D} \rangle, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle$
11. $\langle \widehat{S}, \widehat{\varepsilon}, \langle C', C'' \rangle \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{\text{secdv1}} \langle \langle C', C'' \rangle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle$
12. $\langle \langle C', C'' \rangle \widehat{S}, \widehat{\varepsilon}, \text{present}_s \widehat{C}, \widehat{W}, \widehat{ST}, s \widehat{SI}, \widehat{D} \rangle \mapsto_{\text{secdv1}} \langle \widehat{S}, \widehat{\varepsilon}, C' \widehat{C}, \widehat{W}, \widehat{ST}, s \widehat{SI}, \widehat{D} \rangle$
13. $\langle \langle C', C'' \rangle \widehat{S}, \widehat{\varepsilon}, \text{present}_s \widehat{C}, \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C}', \widehat{D}' \rangle \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{\text{secdv1}} \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C}', \widehat{W}, \widehat{ST} \langle s, \langle \widehat{S}, \widehat{\varepsilon}, \text{present}_s \widehat{C}, \widehat{D} \rangle \rangle, \widehat{SI}, \widehat{D}' \rangle$
avec $s \notin \widehat{SI}$
14. $\langle \langle C', C'' \rangle \widehat{S}, \widehat{\varepsilon}, \text{present}_s \widehat{C}, \emptyset, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{\text{secdv1}} \langle \emptyset, \emptyset, \emptyset, \emptyset, \widehat{ST} \langle s, \langle \widehat{S}, \widehat{\varepsilon}, \text{present}_s \widehat{C}, \widehat{D} \rangle \rangle, \emptyset, \emptyset \rangle$ avec $s \notin \widehat{SI}$
15. $\langle \widehat{S}, \widehat{\varepsilon}, \emptyset, \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C}', \widehat{D}' \rangle \widehat{W}, \widehat{ST}, \widehat{SI}, \emptyset \rangle \mapsto_{\text{secdv1}} \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C}', \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}' \rangle$
16. $\langle \widehat{S}, \widehat{\varepsilon}, \emptyset, \emptyset, \widehat{ST}, \widehat{SI}, \emptyset \rangle \mapsto_{\text{secdv1}} \langle \widehat{S}, \widehat{\varepsilon}, \emptyset, \widehat{W}, \emptyset, \emptyset, \emptyset \rangle$ avec $\widehat{W} =$ tout les éléments de \widehat{ST} qui prennent leurs 2nd choix
17. $\langle \widehat{S}, \widehat{\varepsilon}, \text{emit}_s \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{\text{secdv1}} \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}', \widehat{ST}', \widehat{SI}, \widehat{D} \rangle$
avec $\widehat{W}' = \widehat{W} \cup$ tout les éléments de \widehat{ST} qui attendent l'émission de s et
avec $\widehat{ST}' = \widehat{ST} \setminus$ tout les éléments de \widehat{ST} qui attendent l'émission de s

la machine SECD version 1 peut s'arrêter dans 3 états différents :

- on a une **constante b** tels que $\langle \varepsilon, \emptyset, [M]_{\text{secdv1}}, \emptyset, \emptyset, \emptyset, \varepsilon \rangle \mapsto_{\text{secdv1}} \langle b, \widehat{\varepsilon}, \varepsilon, \emptyset, \emptyset, \widehat{SI}, \varepsilon \rangle$;
- on a une **abstraction function** tels que $\langle \varepsilon, \emptyset, [M]_{\text{secdv1}}, \emptyset, \emptyset, \emptyset, \varepsilon \rangle \mapsto_{\text{secdv1}} \langle \langle \langle X, \widehat{C} \rangle, \widehat{\varepsilon}' \rangle, \widehat{\varepsilon}, \varepsilon, \emptyset, \emptyset, \widehat{SI}, \varepsilon \rangle$;
- on a un **remplacement** tels que $\langle \varepsilon, \emptyset, [M]_{\text{secdv1}}, \emptyset, \emptyset, \emptyset, \varepsilon \rangle \mapsto_{\text{secdv1}} \langle \text{Remp}, \widehat{\varepsilon}, \varepsilon, \emptyset, \emptyset, \widehat{SI}, \varepsilon \rangle$;
- on a un **état inconnu** soit une **erreur**.

5.2.2 2ème version des règles de la machine SECD Concurrente

Soit $\langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle$ avec :

$$\widehat{S} = \dots \mid \langle C', C'' \rangle \widehat{S}$$

$$\widehat{\varepsilon} = \dots \mid s \text{ (un signal initialisé)}$$

$$\widehat{C} = \dots \mid [\langle C', C'' \rangle \widehat{C} \mid \text{present}_s \widehat{C}] \text{ changé en } \langle s, C', C'' \rangle \widehat{C} \mid \langle s, \widehat{C}' \rangle \widehat{C}$$

$$\widehat{W} = \dots$$

$$\widehat{ST} = \dots$$

$$\widehat{SI} = \dots$$

$$\widehat{D} = \dots$$

Les nouvelles règles sont les suivantes :

1. $\langle \widehat{S}, \widehat{\varepsilon}, \langle C', C'' \rangle \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{secdv1} \langle \langle C', C'' \rangle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle$
2. $\langle \langle C', C'' \rangle \widehat{S}, \widehat{\varepsilon}, \text{present}_s \widehat{C}, \widehat{W}, \widehat{ST}, s \widehat{SI}, \widehat{D} \rangle \mapsto_{secdv1} \langle \widehat{S}, \widehat{\varepsilon}, C' \widehat{C}, \widehat{W}, \widehat{ST}, s \widehat{SI}, \widehat{D} \rangle$
3. $\langle \langle C', C'' \rangle \widehat{S}, \widehat{\varepsilon}, \text{present}_s \widehat{C}, \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C}''', \widehat{D}' \rangle \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{secdv1} \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C}''', \widehat{W}, \widehat{ST} \langle s, \langle \widehat{S}, \widehat{\varepsilon}, \text{present}_s \widehat{C}, \widehat{D} \rangle \rangle, \widehat{SI}, \widehat{D}' \rangle$
avec $s \notin \widehat{SI}$
4. $\langle \langle C', C'' \rangle \widehat{S}, \widehat{\varepsilon}, \text{present}_s \widehat{C}, \emptyset, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{secdv1} \langle \emptyset, \emptyset, \emptyset, \emptyset, \widehat{ST} \langle s, \langle \widehat{S}, \widehat{\varepsilon}, \text{present}_s \widehat{C}, \widehat{D} \rangle \rangle, \emptyset, \emptyset \rangle$ avec $s \notin \widehat{SI}$
5. $\langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{secdv2} \langle \widehat{S}, \widehat{\varepsilon}, C' \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle$ si $s \in \widehat{SI}$ et $s \in \widehat{\varepsilon}$
6. $\langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle \widehat{C}, \langle \widehat{S}', \widehat{\varepsilon}', C''', \widehat{D}' \rangle \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{secdv2} \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C}''', \widehat{W}, \widehat{ST} \langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle \widehat{C}, \widehat{D} \rangle, \widehat{SI}, \widehat{D}' \rangle$ si $s \notin \widehat{SI}$ et $s \in \widehat{\varepsilon}$
7. $\langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle \widehat{C}, \emptyset, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{secdv2} \langle \epsilon, \emptyset, \emptyset, \emptyset, \widehat{ST} \langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle \widehat{C}, \widehat{D} \rangle, \widehat{SI}, \emptyset \rangle$ si $s \notin \widehat{SI}$ et $s \in \widehat{\varepsilon}$
8. $\langle \widehat{S}, \widehat{\varepsilon}, \langle s, C' \rangle \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{secdv2} \langle \epsilon, \widehat{\varepsilon}[\text{init} \leftarrow s], C', \widehat{W}, \widehat{ST}, \widehat{SI}, \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D} \rangle \rangle$

la machine SECD version 2 peut s'arrêter dans 3 états différents :

- on a une **constante** **b** tels que $\langle \epsilon, \emptyset, [M]_{secdv1}, \emptyset, \emptyset, \emptyset, \epsilon \rangle \twoheadrightarrow_{secdv1} \langle b, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon \rangle$;
- on a une **abstraction function** tels que $\langle \epsilon, \emptyset, [M]_{secdv1}, \emptyset, \emptyset, \emptyset, \epsilon \rangle \twoheadrightarrow_{secdv1} \langle \langle \langle X, \widehat{C} \rangle, \widehat{\varepsilon}' \rangle, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon \rangle$;
- on a un **remplacement** tels que $\langle \epsilon, \emptyset, [M]_{secdv1}, \emptyset, \emptyset, \emptyset, \epsilon \rangle \twoheadrightarrow_{secdv1} \langle \text{Remp}, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon \rangle$;
- on a un **état inconnu** soit une **erreur**.

5.2.3 3ème version des règles de la machine SECD Concurrente

5.2.4 4ème version des règles de la machine SECD Concurrente

6 Bibliographie

- [1] *Réactivité des systèmes coopératifs : le cas Réactive ML* de Louis Mandrel et Cédric Pasteur
- [2] *The ZINC experiment : an economical implementation of the ML language* de Xavier Leroy
- [3] *Programming Languages And Lambda Calculi* de Mathias Felleisen et Matthew Flatt