Rapport de stage
Développement d'un noyau de programmation synchrone

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

25 Mars 2019

Table des matières

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 $X_3 \notin FV(M_1) \backslash X_1$

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 intéraction dynamique et faire une compilation grâce à *ocamlbuild*.

L'implémentation du langage ISWIM a été fait le vendredi . La machinne 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 bloquées

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 :

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

$\widehat{ST} = \{\langle s, \widehat{D}\rangle, ...\}$ avec s 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 émits 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 émits.

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 réunit, 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 terminé le lundi et vous pouvez trouvez 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 ajouté 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 initialisé.

Pour palier à ce problème une 2ème version de ces règles fût créer 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 faites 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 créé pout la machine CCH présenté dans [3] j'ai aussi rajouté un autre élément à la machine SECD concurrente : le **Handler**.

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

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 $\longrightarrow$ 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* $\longrightarrow$ *chaque processus va régulièrement "laisser la main aux autres"*

Ordonnancement préemptif $\longrightarrow$ *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 is 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 instantannément 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 : := $\bullet$ | $\circ$ | ϕ | k||k | k+k | k ;k | $\mu\phi$.k | run k | 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 $X_3 \notin FV(M_1) \backslash X_1$

— $(M_1\ M_2)[X \leftarrow M_3] = (M_1[X \leftarrow M_3]\ M_2[X \leftarrow M_3])$

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 $\boldsymbol{n} = \alpha \cup \beta \cup \eta$.

3.2.3 Les règles de priorité

— Application associative à gauche : M1 M2 M3 = ((M1 M2)M3)

— Application prioritaire par rapport au abstraction : λX.M1 M2 = λX.(M1 M2)

— 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 rédution.

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 $M \twoheadrightarrow_{n_n} L'$ et $N \twoheadrightarrow_{n_n} L'$.

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

— $M \longrightarrow_{\bar{n}} N$ if $M\ \beta\ N$

— $M \longrightarrow_{\bar{n}} N$ if $M\ \eta\ N$

— $(\lambda X.M) \longrightarrow_{\bar{n}} (\lambda X.N)$

— $(M\ N) \longrightarrow_{\bar{n}} (M'\ N)$ if $M \longrightarrow_{\bar{n}} M'$
et $\forall L$, $(M\ N)\ \beta$ L impossible et $(M\ N)\ \eta$ L impossible

— $(M\ N) \longrightarrow_{\bar{n}} (M\ N')$ if $N \longrightarrow_{\bar{n}} N'$
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)

| (λX.M)

| (M M)

| b (les constantes b)

| (o^n M ... 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 $\longleftarrow$ M] = b

— (o^n M_1 ... M_n)[X $\longleftarrow$ M] = (o^n M_1[X $\longleftarrow$ M] ... M_n[X $\longleftarrow$ M])

La réduction est la même quand lambda calcul mais on vérifie juste que la réduction est faite avec une valeur. (λX.M) V) β_v M[X $\longleftarrow$ 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 language ISWIM.

Les règles définit pour cette machine sont les suivantes :

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

La machine peut s'arrêter dans 3 états différents :

$\longrightarrow$ on a une **constante b** tels que $\langle M, []\rangle \twoheadrightarrow_{cc} \langle b, []\rangle$;

$\longrightarrow$ on a une **abstraction function** tels que $\langle M, []\rangle \twoheadrightarrow_{cc} \langle \lambda X.N, []\rangle$;

$\longrightarrow$ 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 définisse la machine SCC sont les suivantes :

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

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

$\longrightarrow$ on a une **constante b** tels que $\langle M, []\rangle \twoheadrightarrow_{scc} \langle b, []\rangle$;

$\longrightarrow$ on a une **abstraction function** tels que $\langle M, []\rangle \twoheadrightarrow_{scc} \langle \lambda X.N, []\rangle$;

$\longrightarrow$ 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 context 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.

κ = mt
| $\langle fun, V, \kappa\rangle$
| $\langle arg, N, \kappa\rangle$
| $\langle opd, \langle V, ..., V, o^n\rangle, \langle N, ...\rangle, \kappa\rangle$

Cette structure est nommé **la continuation**.

Les règles qui définisse la machine CK sont les suivantes :

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

la machine CK peut s'arrêter dans 3 états différents :
$\longrightarrow$ on a une **constante b** tels que $\langle M, mt\rangle \twoheadrightarrow_{ck} \langle b, mt\rangle$;
$\longrightarrow$ on a une **abstraction function** tels que $\langle M, mt\rangle \twoheadrightarrow_{ck} \langle \lambda X.N, mt\rangle$;
$\longrightarrow$ 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 necessitée. Pour cela, la machine CEK ajoute les clauses et un environnement ε qui va stocker les substitutions à faire.

On a alors :
ε = une fonction $\{\langle X, c\rangle, ...\}$ c = $\{\langle M, \varepsilon\rangle \mid FV(M) \subset dom(\varepsilon)\}$ v = $\{\langle V, \varepsilon\rangle \mid \langle V, \varepsilon\rangle \in c\}$
$\varepsilon[X \leftarrow c] = \{\langle X, c\rangle\} \cup \{\langle Y, c'\rangle \mid \langle Y, c'\rangle \in \varepsilon \text{ et } Y \neq X\}$

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

$\overline{\kappa}$ = mt
| $\langle fun, v, \overline{\kappa}\rangle$
| $\langle arg, c, \overline{\kappa}\rangle$
| $\langle opd, \langle v, ..., v, o^n\rangle, \langle c, ...\rangle, \overline{\kappa}\rangle$

Les règles qui définisse la machine CEK sont les suivantes :

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

la machine CEK peut s'arrêter dans 3 états différents :
$\longrightarrow$ on a une **constante b** tels que $\langle\langle M, \emptyset\rangle, mt\rangle \twoheadrightarrow_{cek} \langle\langle b, \varepsilon\rangle, mt\rangle$;
$\longrightarrow$ on a une **abstraction function** tels que $\langle\langle M, \emptyset\rangle, mt\rangle \twoheadrightarrow_{cek} \langle\langle \lambda X.N, \varepsilon\rangle, mt\rangle$;
$\longrightarrow$ on a un **état inconnu** soit une **erreur**.

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

3.4.5 SECD

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 :

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

Les règles qui définisse la machine SECD sont les suivantes :

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

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

$\longrightarrow$ on a une **constante b** tels que $\langle \epsilon, \emptyset, [M]_{secd}, \epsilon\rangle \twoheadrightarrow_{cek} \langle b, \widehat{\varepsilon}, \epsilon, \epsilon\rangle$;

$\longrightarrow$ on a une **abstraction function** tels que $\langle \epsilon, \emptyset, [M]_{secd}, \epsilon\rangle \twoheadrightarrow_{cek} \langle\langle\langle X, \widehat{C}\rangle, \widehat{\varepsilon'}\rangle, \widehat{\varepsilon}, \epsilon, \epsilon\rangle$;

$\longrightarrow$ on a un **état inconnu** soit une **erreur**.

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

4 Conclusion

5 Annexe

5.1 Exemple de fonctionnement de la machine CC

Voici un exemple de fonctionnement de la machine CC :

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

5.2 Exemple de fonctionnement de la machine SCC

Voici un exemple de fonctionnement de la machine SCC :

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

5.3 Exemple de fonctionnement de la machine CK

Voici un exemple de fonctionnement de la machine CK :

CK machine : $\langle(((\lambda f.\lambda x.f\ x)\ \lambda y.(+\ y\ y))\ \ulcorner 1\urcorner), mt\rangle$

$> \langle(M\ N), \kappa\rangle \longmapsto_{ck} \langle M, \langle arg, N, \kappa\rangle\rangle$

CK machine : $\langle((\lambda f.\lambda x.f\ x)\ \lambda y.(+\ y\ y)), \langle arg, \ulcorner 1\urcorner, mt\rangle\rangle$

$> \langle(M\ N), \kappa\rangle \longmapsto_{ck} \langle M, \langle arg, N, \kappa\rangle\rangle$

CK machine : $\langle(\lambda f.\lambda x.f\ x), \langle arg, (\lambda y.(+\ y\ y)), \langle arg, \ulcorner 1\urcorner, mt\rangle\rangle\rangle$

$> \langle V, \langle arg, N, \kappa\rangle\rangle \longmapsto_{ck} \langle N, \langle fun, V, \kappa\rangle\rangle$

CK machine : $\langle(\lambda y.(+\ y\ y)), \langle fun, (\lambda f.\lambda x.f\ x), \langle arg, \ulcorner 1\urcorner, mt\rangle\rangle\rangle$

$> \langle V, \langle fun, (\lambda X.M), \kappa\rangle\rangle \longmapsto_{ck} \langle M[X \leftarrow V], \kappa\rangle$

CK machine : $\langle(\lambda x.f\ x)[f \leftarrow (\lambda y.(+\ y\ y))], \langle arg, \ulcorner 1\urcorner, mt\rangle\rangle$

CK machine : $\langle(\lambda x.(\lambda y.(+\ y\ y))\ x), \langle arg, \ulcorner 1\urcorner, mt\rangle\rangle$

$> \langle V, \langle arg, N, \kappa\rangle\rangle \longmapsto_{ck} \langle N, \langle fun, V, \kappa\rangle\rangle$

CK machine : $\langle \ulcorner 1\urcorner, \langle fun, (\lambda x.(\lambda y.(+\ y\ y))\ x), mt\rangle\rangle$

$> \langle V, \langle fun, (\lambda X.M), \kappa\rangle\rangle \longmapsto_{ck} \langle M[X \leftarrow V], \kappa\rangle$

CK machine : $\langle((\lambda y.(+\ y\ y))\ x)[x \leftarrow \ulcorner 1\urcorner], mt\rangle$

CK machine : $\langle((\lambda y.(+\ y\ y))\ \ulcorner 1\urcorner), mt\rangle$

$> \langle(M\ N), \kappa\rangle \longmapsto_{ck} \langle M, \langle arg, N, \kappa\rangle\rangle$

CK machine : $\langle(\lambda y.(+\ y\ y)), \langle arg, \ulcorner 1\urcorner, mt\rangle\rangle$

$> \langle V, \langle arg, N, \kappa\rangle\rangle \longmapsto_{ck} \langle N, \langle fun, V, \kappa\rangle\rangle$

CK machine : $\langle \ulcorner 1\urcorner, \langle fun, (\lambda y.(+\ y\ y)), mt\rangle\rangle$

$> \langle V, \langle fun, (\lambda X.M), \kappa\rangle\rangle \longmapsto_{ck} \langle M[X \leftarrow V], \kappa\rangle$

CK machine : $\langle(+\ y\ y)[y \leftarrow \ulcorner 1\urcorner], mt\rangle$

CK machine : $\langle(+\ \ulcorner 1\urcorner\ \ulcorner 1\urcorner), mt\rangle$

$> \langle(o^n\ M\ N...), \kappa\rangle \longmapsto_{ck} \langle M, \langle opd, \langle o^n\rangle, \langle N, ...\rangle, \kappa\rangle\rangle$

CK machine : $\langle \ulcorner 1\urcorner, \langle opd, \langle +\rangle, \langle \ulcorner 1\urcorner\rangle, mt\rangle\rangle$

$> \langle V, \langle opd, \langle V', ...o^n\rangle, \langle N, L, ...\rangle, \kappa\rangle\rangle \longmapsto_{ck} \langle N, \langle opd, \langle V, V', ...o^n\rangle, \langle L, ...\rangle, \kappa\rangle\rangle$

CK machine : $\langle \ulcorner 1\urcorner, \langle opd, \langle \ulcorner 1\urcorner, +\rangle, \langle\rangle, mt\rangle\rangle$

$> \langle b, \langle opd, \langle b_i, ...b_1, o^n\rangle, \langle\rangle, \kappa\rangle\rangle \longmapsto_{ck} \langle V, \kappa\rangle$ avec $\delta(o^n, b_1, ...b_i, b) = V$

CK machine : $\langle \ulcorner 2\urcorner, mt\rangle$

5.4 Exemple de fonctionnement de la machine CEK

Voici un exemple de fonctionnement de la machine CEK :

CEK machine : $\langle\langle(((\lambda f.\lambda x.f\ x)\ \lambda y.(+\ y\ y))\ \ulcorner 1\urcorner)\emptyset\rangle, mt\rangle$

$> \langle\langle(M\ N), \varepsilon\rangle, \overline{\kappa}\rangle \longmapsto_{cek} \langle\langle M, \varepsilon\rangle, \langle arg, \langle N, \varepsilon\rangle, \overline{\kappa}\rangle\rangle$

CEK machine : $\langle\langle((\lambda f.\lambda x.f\ x)\ \lambda y.(+\ y\ y))\emptyset\rangle, \langle arg, \langle\ulcorner 1\urcorner, \emptyset\rangle, mt\rangle\rangle$

$> \langle\langle(M\ N), \varepsilon\rangle, \overline{\kappa}\rangle \longmapsto_{cek} \langle\langle M, \varepsilon\rangle, \langle arg, \langle N, \varepsilon\rangle, \overline{\kappa}\rangle\rangle$

CEK machine : $\langle\langle(\lambda f.\lambda x.f\ x), \emptyset\rangle, \langle arg, \langle(\lambda y.(+\ y\ y)), \emptyset\rangle, \langle arg, \langle\ulcorner 1\urcorner, \emptyset\rangle, mt\rangle\rangle\rangle$

$> \langle\langle V, \varepsilon\rangle, \langle arg, \langle N, \varepsilon'\rangle, \kappa\rangle\rangle \longmapsto_{cek} \langle\langle N, \varepsilon'\rangle, \langle fun, \langle V, \varepsilon\rangle, \overline{\kappa}\rangle\rangle$ si $V \notin X$

CEK machine : $\langle\langle(\lambda y.(+\ y\ y)), \emptyset\rangle, \langle fun, \langle(\lambda f.\lambda x.f\ x), \emptyset\rangle, \langle arg, \langle\ulcorner 1\urcorner, \emptyset\rangle, mt\rangle\rangle\rangle$

$> \langle\langle V, \varepsilon\rangle, \langle fun, \langle(\lambda X1.M), \varepsilon'\rangle, \overline{\kappa}\rangle\rangle \longmapsto_{cek} \langle\langle M, \varepsilon'[X1 \leftarrow \langle V, \varepsilon\rangle]\rangle, \overline{\kappa}\rangle$ si $V \notin X$

CEK machine : $\langle\langle(\lambda x.f\ x), \emptyset[f \leftarrow \langle(\lambda y.(+\ y\ y)), \emptyset\rangle]\rangle, \langle arg, \langle\ulcorner 1\urcorner, \emptyset\rangle, mt\rangle\rangle$

CEK machine : $\langle\langle(\lambda x.f\ x), \{\langle f, \langle(\lambda y.(+\ y\ y)), \emptyset\rangle\rangle\}\rangle, \langle arg, \langle\ulcorner 1\urcorner, \emptyset\rangle, mt\rangle\rangle$

$> \langle\langle V, \varepsilon\rangle, \langle arg, \langle N, \varepsilon'\rangle, \kappa\rangle\rangle \longmapsto_{cek} \langle\langle N, \varepsilon'\rangle, \langle fun, \langle V, \varepsilon\rangle, \overline{\kappa}\rangle\rangle$ si $V \notin X$

CEK machine : $\langle\langle\ulcorner 1\urcorner, \emptyset\rangle, \langle fun, \langle(\lambda x.f\ x), \{\langle f, \langle(\lambda y.(+\ y\ y)), \emptyset\rangle\rangle\}\rangle, mt\rangle\rangle$

$> \langle\langle V, \varepsilon\rangle, \langle fun, \langle(\lambda X1.M), \varepsilon'\rangle, \overline{\kappa}\rangle\rangle \longmapsto_{cek} \langle\langle M, \varepsilon'[X1 \leftarrow \langle V, \varepsilon\rangle]\rangle, \overline{\kappa}\rangle$ si $V \notin X$

CEK machine : $\langle\langle(f\ x), \{\langle f, \langle(\lambda y.(+\ y\ y)), \emptyset\rangle\rangle\}[x \leftarrow \langle\ulcorner 1\urcorner, \emptyset\rangle]\rangle, mt\rangle$

CEK machine : $\langle\langle(f\ x), \{\langle f, \langle(\lambda y.(+\ y\ y)), \emptyset\rangle\rangle, \langle x, \langle\ulcorner 1\urcorner, \emptyset\rangle\rangle\}\rangle, mt\rangle$

$> \langle\langle(M\ N), \varepsilon\rangle, \overline{\kappa}\rangle \longmapsto_{cek} \langle\langle M, \varepsilon\rangle, \langle arg, \langle N, \varepsilon\rangle, \overline{\kappa}\rangle\rangle$

CEK machine : $\langle\langle f, \{\langle f, \langle(\lambda y.(+\ y\ y)), \emptyset\rangle\rangle, \langle x, \langle\ulcorner 1\urcorner, \emptyset\rangle\rangle\}\rangle, \langle arg, \langle x, \{\langle f, \langle(\lambda y.(+\ y\ y)), \emptyset\rangle\rangle, \langle x, \langle\ulcorner 1\urcorner, \emptyset\rangle\rangle\}\rangle, mt\rangle\rangle$

$> \langle\langle X, \varepsilon\rangle, \overline{\kappa}\rangle \longmapsto_{cek} \langle c, \overline{\kappa}\rangle$ avec $\varepsilon(X) = c$

CEK machine : $\langle\langle(\lambda y.(+\ y\ y)), \emptyset\rangle, \langle arg, \langle x, \{\langle f, \langle(\lambda y.(+\ y\ y)), \emptyset\rangle\rangle, \langle x, \langle\ulcorner 1\urcorner, \emptyset\rangle\rangle\}\rangle, mt\rangle\rangle$

$> \langle\langle V, \varepsilon\rangle, \langle arg, \langle N, \varepsilon'\rangle, \kappa\rangle\rangle \longmapsto_{cek} \langle\langle N, \varepsilon'\rangle, \langle fun, \langle V, \varepsilon\rangle, \overline{\kappa}\rangle\rangle$ si $V \notin X$

CEK machine : $\langle\langle x, \{\langle f, \langle(\lambda y.(+\ y\ y)), \emptyset\rangle\rangle, \langle x, \langle\ulcorner 1\urcorner, \emptyset\rangle\rangle\}\rangle, \langle fun, \langle(\lambda y.(+\ y\ y)), \emptyset\rangle, mt\rangle\rangle$

$> \langle\langle X, \varepsilon\rangle, \overline{\kappa}\rangle \longmapsto_{cek} \langle c, \overline{\kappa}\rangle$ avec $\varepsilon(X) = c$

CEK machine : $\langle\langle\ulcorner 1\urcorner, \emptyset\rangle, \langle fun, \langle(\lambda y.(+\ y\ y)), \emptyset\rangle, mt\rangle\rangle$

$> \langle\langle V, \varepsilon\rangle, \langle fun, \langle(\lambda X1.M), \varepsilon'\rangle, \overline{\kappa}\rangle\rangle \longmapsto_{cek} \langle\langle M, \varepsilon'[X1 \leftarrow \langle V, \varepsilon\rangle]\rangle, \overline{\kappa}\rangle$ si $V \notin X$

CEK machine : $\langle\langle(+\ y\ y), \emptyset[y \leftarrow \langle\ulcorner 1\urcorner, \emptyset\rangle]\rangle, mt\rangle$

CEK machine : $\langle\langle(+\ y\ y), \{\langle y, \langle\ulcorner 1\urcorner, \emptyset\rangle\rangle\}\rangle, mt\rangle$

$> \langle\langle(o^n\ M\ N...), \varepsilon\rangle, \overline{\kappa}\rangle \longmapsto_{cek} \langle\langle M, \varepsilon\rangle, \langle opd, \langle o^n\rangle, \langle\langle N, \varepsilon\rangle, ...\rangle, \overline{\kappa}\rangle\rangle$

CEK machine : $\langle\langle y, \{\langle y, \langle\ulcorner 1\urcorner, \emptyset\rangle\rangle\}\rangle, \langle opd, \langle +\rangle, \langle\langle y, \{\langle y, \langle\ulcorner 1\urcorner, \emptyset\rangle\rangle\}\rangle\rangle, mt\rangle\rangle$

$> \langle\langle X, \varepsilon\rangle, \overline{\kappa}\rangle \longmapsto_{cek} \langle c, \overline{\kappa}\rangle$ avec $\varepsilon(X) = c$

CEK machine : $\langle\langle\ulcorner 1\urcorner, \emptyset\rangle, \langle opd, \langle +\rangle, \langle\langle y, \{\langle y, \langle\ulcorner 1\urcorner, \emptyset\rangle\rangle\}\rangle\rangle, mt\rangle\rangle$

$> \langle\langle V, \varepsilon\rangle, \langle opd, \langle v', ...o^n\rangle, \langle\langle N, \varepsilon'\rangle, c, ...\rangle, \overline{\kappa}\rangle\rangle \longmapsto_{cek} \langle\langle N, \varepsilon'\rangle, \langle opd, \langle\langle V, \varepsilon\rangle, v', ...o^n\rangle, \langle c, ...\rangle, \overline{\kappa}\rangle\rangle$ si $V \notin X$

CEK machine : $\langle\langle y, \{\langle y, \langle\ulcorner 1\urcorner, \emptyset\rangle\rangle\}\rangle, \langle opd, \langle\ulcorner 1\urcorner +\rangle, \langle\rangle, mt\rangle\rangle$

$> \langle\langle X, \varepsilon\rangle, \overline{\kappa}\rangle \longmapsto_{cek} \langle c, \overline{\kappa}\rangle$ avec $\varepsilon(X) = c$

CEK machine : $\langle\langle\ulcorner 1\urcorner, \emptyset\rangle, \langle opd, \langle\ulcorner 1\urcorner +\rangle, \langle\rangle, mt\rangle\rangle$

$> \langle\langle b, \varepsilon\rangle, \langle opd, \langle\langle b_i, \varepsilon_i\rangle, ...\langle b_1, \varepsilon_1\rangle, o^n\rangle, \langle\rangle, \overline{\kappa}\rangle\rangle \longmapsto_{cek} \langle\langle V, \emptyset\rangle, \overline{\kappa}\rangle$ avec $\delta(o^n, b_1, ...b_i, b) = V$

CEK machine : $\langle\langle\ulcorner 2\urcorner, \emptyset\rangle, mt\rangle$

5.5 Exemple de fonctionnement de la machine SECD

Voici un exemple de fonctionnement de la machine SECD :

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

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

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

Conversion : $\langle f, \langle x, f\ x\ ap\rangle\rangle\ \langle y, y\ y\ prim_+\rangle\ ap\ \ulcorner 1\urcorner\ ap$

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

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

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

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

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

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

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

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

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

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

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

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

$> \langle \widehat{S}, \widehat{\varepsilon}, b\ \widehat{C}, \widehat{D}\rangle \longmapsto_{secd} \langle b\ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D}\rangle$

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

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

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

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

$> \langle \widehat{S}, \widehat{\varepsilon}, X\ \widehat{C}, \widehat{D}\rangle \longmapsto_{secd} \langle \widehat{V}\ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D}\rangle$ où $\widehat{V} = \varepsilon(X)$

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

$> \langle \widehat{S}, \widehat{\varepsilon}, X\ \widehat{C}, \widehat{D}\rangle \longmapsto_{secd} \langle \widehat{V}\ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D}\rangle$ où $\widehat{V} = \varepsilon(X)$

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

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

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

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

$> \langle \widehat{S}, \widehat{\varepsilon}, X\ \widehat{C}, \widehat{D}\rangle \longmapsto_{secd} \langle \widehat{V}\ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D}\rangle$ où $\widehat{V} = \varepsilon(X)$

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

$> \langle \widehat{S}, \widehat{\varepsilon}, X\ \widehat{C}, \widehat{D}\rangle \longmapsto_{secd} \langle \widehat{V}\ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D}\rangle$ où $\widehat{V} = \varepsilon(X)$

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

$> \langle b_1\ ...\ b_n \widehat{S}, \widehat{\varepsilon}, prim_{o^n}\ \widehat{C}, \widehat{D}\rangle \longmapsto_{secd} \langle \widehat{V}\ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D}\rangle$ où $\widehat{V} = \delta(o^n, b_1, ...b_n)$

SECD Machine : $\langle \ulcorner 2\urcorner, \{\langle y, \ulcorner 1\urcorner\rangle\}, \emptyset, \langle \epsilon, \{\langle f, \langle\langle y, y\ y\ prim_+\rangle, \emptyset\rangle\rangle, \langle x, \ulcorner 1\urcorner\rangle\}, \emptyset, \langle \epsilon, \emptyset, \epsilon, \epsilon\rangle\rangle\rangle$

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

SECD Machine : $\langle \ulcorner 2\urcorner, \{\langle f, \langle\langle y, y\ y\ prim_+\rangle, \emptyset\rangle\rangle\}, \langle x, \ulcorner 1\urcorner\rangle\}, \emptyset, \langle \epsilon, \emptyset, \epsilon, \epsilon\rangle\rangle$

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

SECD Machine : $\langle \ulcorner 2\urcorner, \emptyset, \epsilon, \epsilon\rangle$

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