

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	Remerciement	3
2	Introduction	3
3	Programmation réactive	4
3.1	Recherches préliminaires	4
3.1.1	Comment créer un langage de programmation ?	4
3.1.2	Les λ -calculs	4
3.2	ISWIM	5
3.3	Les différentes machines traitées	6
3.3.1	CC Machine	6
3.3.2	SCC Machine	6
3.3.3	CK Machine	7
3.3.4	CEK Machine	7
3.3.5	SECD Machine	8
4	Programmation fonctionnelle	9
4.1	Le Réactive ML	9
5	Langage fonctionnel réactif	10
6	Conclusion	14
7	Annexes	15
7.1	Les Exemples des machines étudiées	15
7.1.1	Exemple de fonctionnement de la machine CC	15
7.1.2	Exemple de fonctionnement de la machine SCC	15
7.1.3	Exemple de fonctionnement de la machine CK	16
7.1.4	Exemple de fonctionnement de la machine CEK	17
7.1.5	Exemple de fonctionnement de la machine SECD	18
7.2	Les différentes versions faite pour rendre la machine SECD concurrente	19
7.2.1	1ère version des règles de la machine SECD Concurrente	19
7.2.2	2ème version des règles de la machine SECD Concurrente	21
7.2.3	3ème version des règles de la machine SECD Concurrente	22
7.2.4	4 ème version de la machine SECD concurrente	23
8	Bibliographie	29
9	Journal de bord	29
9.1	Mars	29
9.1.1	Semaine 25 au 29 Mars	29
9.2	Avril	29
9.2.1	Semaine du 1 au 5 Avril	29
9.2.2	Semaine du 8 au 12	31
9.2.3	Semaine du 15 au 19	32
9.2.4	Semaine du 22 au 26	33
9.3	Mai	34
9.3.1	Semaine du 29 Avril au 3	34
9.3.2	Semaine du 6 au 10	34
9.3.3	Semaine du 13 au 17	34
9.3.4	Semaine du 20 au 24	34

1 Remerciement

Avant tout développement sur mon sujet de stage, j'aimerais remercier mes 2 professeurs qui m'ont encadrés pour m'avoir permis de faire ce stage de recherche et de m'avoir aidé tout le long de celui-ci. J'ai beaucoup appris grâce à eux. Je remercie donc Madame Bousdira et Monsieur Dabrowski pour tout.

2 Introduction

Le Laboratoire d'Informatique Fondamentale d'Orléans (LIFO) est un laboratoire de l'Université d'Orléans et de l'INSA Centre-Val de Loire.

Les recherches menées au LIFO concernent la science informatique et les STIC. Elles vont de l'algorithmique au traitement des langues naturelles, de l'apprentissage au parallélisme massif, de la vérification et la certification à la sécurité des systèmes, du Big Data aux systèmes embarqués. Le laboratoire est structuré en cinq équipes :

- Contraintes et Apprentissage (CA)
- Graphes, Algorithmes et Modèles de Calcul (GAMoC)
- Langages, Modèles et Vérification (LMV)
- PaMDA
- Sécurité des Données et des Systèmes (SDS)

Afin d'offrir une autre approche du laboratoire et de promouvoir la coopération entre équipes, les thématiques transversales suivantes ont été définies :

- Masse de données et calcul haute performance
- Modélisation et algorithmique
- Sécurité et sûreté

J'ai eu l'occasion de travailler une partie de l'équipe LMV, dans l'optique d'un stage rémunéré de 3 mois ; voici une la description de l'équipe LMV.

L'objectif de l'équipe LMV est de contribuer à l'amélioration de la compréhension des problèmes de sûreté et de sécurité des systèmes informatiques. Des logiques «ordres partiels» aux langages de programmation usuels, les membres de l'équipes travaillent sur ces questions à différents niveaux d'abstraction et selon différents points de vue tout en cherchant à comprendre les relations fondamentales entre ces différentes approches. L'équipe est structurée autour de deux axes : la correction de programmes et la vérification de systèmes.

- Le premier axe s'intéresse au développement de techniques liées aussi bien à la vérification de propriétés spécifiques qu'à la satisfaction de propriété fonctionnelle quelconques. Dans les deux cas les propriétés peuvent être assurées par construction ou a posteriori (vérification déductive).
- Le second axe repose d'une part sur l'étude de techniques à base de systèmes de réécriture comme par exemple les problèmes d'accessibilités dans les systèmes de réécriture et d'autre part sur l'étude des logiques dites «ordres partiels» et leur utilisation dans le cadre du développement d'outils de vérification efficaces.

Le stage a pour intitulé *Programmation réactive synchrone, implantation d'une machine virtuelle*. Il se place dans la thématique *Sémantique des systèmes concurrents*. L'objectif de ce stage est de réaliser l'implantation d'une machine virtuelle (type JVM) destinée à exécuter un langage réactif synchrone purement fonctionnel encadré par deux maîtres de conférence : Mme Bousdira et Mr Dabrowski.

Pour cela une recherche préliminaire sur des machines existantes sera faite en premier lieu pour ce mettre à niveau. Ensuite une recherche sur la programmation réactive servira à comprendre les enjeux du sujet et à cerner les différentes contraintes que la réactivité apporte. Les recherches étant faite on pourra s'atteler à la création de notre propre machine concurrente. Pour finir, on pourra conclure par un résumer le travail abattu, des choix qui pourrait être améliorés, des voies qui n'ont pas été choisie et de ce qu'il reste à faire.

3 Programmation réactive

Ayant un niveau de 3ème année de licence informatique, beaucoup de lacune par rapport au travail demandé, il m'a fallu me mettre à niveau. Pour cela, trois articles ont été étudié :

- [1] expliquant le fonctionnement du ReactiveML un langage de programmation réactif
- [2] développant toute la réflexion que l'on doit avoir pour créer un langage de programmation
- [3] expliquant le fonctionnement des machines avec les λ -calculs

Pour mieux structurer ma démarche, je vais diviser mes recherches en deux sous parties. La première sera liée à mes recherches "préliminaires" qui ne sont pas liées directement à la programmation réactive. Elle regroupera l'article [2] et [3]. La seconde sera dédiée à l'article [1] qui est lui complètement axé sur la programmation réactive.

3.1 Recherches préliminaires

3.1.1 Comment créer un langage de programmation ?

3.1.2 Les λ -calculs

Les bases : Le λ -calcul est un système formel inventé par Alonzo Church dans les années 1930, qui fonde les concepts de fonction et d'application. On y manipule des expressions appelées λ -expressions, où la lettre grecque λ est utilisée pour lier une variable. Il y a trois termes qui composent les λ -expressions :

1. les variables : x, y, \dots sont des λ -termes
2. les applications : $u v$ est un λ -terme si u et v sont des λ -termes, on peut voir l'application comme ceci : si u est une fonction et v un argument, alors $u v$ est le résultat de l'application à v de la fonction u ;
3. les abstractions : $\lambda x.v$ est un λ -terme si x est une variable et v un λ -terme, on peut voir l'abstraction comme ceci : $\lambda x.v$ peut être interprétée comme la fonction qui, à x , associe v , où v contient en général des occurrences de x .

La réduction : Maintenant que nous connaissons les termes qui composent les λ -expressions, il faut savoir comment les faire interagir entre eux. Les λ -calculs fonctionnent par réductions des termes grâce à l'application. Il existe trois règles de **réduction générale** :

- $(\lambda X_1.M) \alpha (\lambda X_1.M[X_1 \leftarrow X_2])$ où $X_2 \notin FV(M)$: elle sert à renommer les variables
- $((\lambda X_1.M_1) M_2) \beta M_1[X \leftarrow M_2]$: elle substitue une variable par un λ -terme
- $(\lambda X.(M X)) \eta M$ où $X \notin FV(M)$: elle représente le cas si on a g une fonction qui à x associe $f(x)$ avec f une fonction alors autant utiliser directement la fonction f .

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

La β -réduction est plus complexe et à lui-même ses propres règles :

- $X_1[X_1 \leftarrow M] = M$
- $X_2[X_1 \leftarrow M] = X_2$ 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 $X_3 \notin FV(M_1) \setminus X_1$
- $(M_1 M_2)[X \leftarrow M_3] = (M_1[X \leftarrow M_3] M_2[X \leftarrow M_3])$

Simplification : Afin d'alléger l'écriture en enlèvement des parenthèses, il y a des règles de priorités qui sont les suivantes :

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

Version écriture lourde

$$\begin{aligned} & ((\lambda x.((\lambda z.z) x)) (\lambda x.x)) \\ \rightarrow_n^\alpha & ((\lambda x.((\lambda z.z) x)) (\lambda y.y)) \\ \rightarrow_n^\eta & ((\lambda z.z) (\lambda y.y)) \\ \rightarrow_n^\beta & (\lambda y.y) \end{aligned}$$

Version écriture légère

$$\begin{aligned} & \lambda x.(\lambda z.z) x \ \lambda x.x \\ \rightarrow_n^\alpha & \lambda x.(\lambda z.z) x \ \lambda y.y \\ \rightarrow_n^\eta & (\lambda z.z) \ \lambda y.y \\ \rightarrow_n^\beta & \lambda y.y \end{aligned}$$

Forme normale 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 normale 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 $M \rightarrow_{n_n} L'$ et $N \rightarrow_{n_n} L'$.

Certaines lambda calcul expression n'a pas de forme normal comme : $((\lambda x.xx)(\lambda x.xx))$. 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 réductions le plus à gauche. Ces règles sont les suivantes :

- $M \rightarrow_{\bar{n}} N$ si $M \beta N$
- $M \rightarrow_{\bar{n}} N$ si $M \eta N$
- $(\lambda X.M) \rightarrow_{\bar{n}} (\lambda X.N)$
- $(M N) \rightarrow_{\bar{n}} (M' N)$ si $M \rightarrow_{\bar{n}} M'$
et $\forall L, (M N) \beta L$ impossible et $(M N) \eta L$ impossible
- $(M N) \rightarrow_{\bar{n}} (M N')$ si $N \rightarrow_{\bar{n}} N'$
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.

La preuve de la compréhension des λ -calculs a été faite à travers de l'implantation de celle-ci en OCaml. Cependant cette implantation reste simplifiée et une version plus complète sera présenté après.

3.2 ISWIM

ISWIM est un langage impératif à noyau fonctionnel; en fait c'est une syntaxe lisible du λ -calcul à laquelle sont ajoutés des variables mutables et des définitions. Grâce au λ -calcul, ISWIM comporte des fonctions d'ordre supérieur et une portée lexicale des variables. Le but est de décrire des concepts en fonction d'autres concepts. Ce langage a fortement influencé les autres langages qui l'ont suivi, principalement dans la programmation fonctionnelle.

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

$M, N, L, K =$

les termes des λ -calculs :

- | X
- | $(\lambda X.M)$
- | $(M N)$

les nouveaux termes :

- | b : une constante
- | $(o^n M \dots N)$ avec o^n les fonctions primitives

On definit une valeur tels que :

$V, U, W =$

- | b
- | X
- | $(\lambda X.M)$

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

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

La β -réduction est la même quand λ -calcul mais à la condition que la réduction soit faite avec une valeur V .

- $((\lambda X.M) V) \beta_v M[X \leftarrow V]$

Cette restriction permet une sorte d'ordre dans les calculs.

Cependant l' η et l' α réduction ne sont plus vue comme tel. En effet l' η -réduction n'est pas utilisé car plus très utile et contraignante à programmer. L' α -réduction sera utilisé pour rechercher une équivalence entre deux termes, on le renommara d'ailleurs l'équivalence en α -équivalence.

Une réduction a été rajouté pour gérer les opérateurs : c'est la δ -réduction. Ce qui nous donne une nouvelle **n**-réduction tels que **n**-réduction = $\beta_v \cup \delta$

Un exemple d'implantation a été codé en Ocaml.

3.3 Les différentes machines traitées

3.3.1 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.

3.3.2 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.

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

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

3.3.4 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. $\langle \langle V, \varepsilon \rangle, \langle \text{fun}, \langle (\lambda X1.M), \varepsilon' \rangle, \bar{\kappa} \rangle \rangle \mapsto_{cek} \langle \langle M, \varepsilon'[X1 \leftarrow \langle V, \varepsilon \rangle] \rangle, \bar{\kappa} \rangle$ si $V \notin X$
4. $\langle \langle V, \varepsilon \rangle, \langle \text{arg}, \langle N, \varepsilon' \rangle, \bar{\kappa} \rangle \rangle \mapsto_{cek} \langle \langle N, \varepsilon' \rangle, \langle \text{fun}, \langle V, \varepsilon \rangle, \bar{\kappa} \rangle \rangle$ 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 $\langle \langle M, \emptyset \rangle, \text{mt} \rangle \mapsto_{cek} \langle \langle b, \varepsilon \rangle, \text{mt} \rangle$;
- on a une **abstraction function** tels que $\langle \langle M, \emptyset \rangle, \text{mt} \rangle \mapsto_{cek} \langle \langle \lambda X.N, \varepsilon \rangle, \text{mt} \rangle$;
- on a un **état inconnu** soit une **erreur**.

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

3.3.5 SECD Machine

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

En effet, dans la machine SECD le contexte est créer par un appel de fonction, quand toute est stocké dans \hat{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 (\hat{S}), d'un environnement ($\hat{\varepsilon}$), d'une chaîne de contrôle (\hat{C}) et d'une sauvegarde (\hat{D}). Les différentes définitions de ces élément sont les suivantes :

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

Les règles qui définisse la machine SECD sont les suivantes :

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

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

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

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

4 Programmation fonctionnelle

4.1 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 qu'un **processus récursif soit réactif** est qu'il ait **toujours** au moins **un instant logique** entre l'instanciation du processus et l'appel récursif.

Le reste de l'article rentre beaucoup plus dans les détails avec l'analyse syntaxique qui prends en compte la réactivité du langage.

5 Langage fonctionnel réactif

La machine ayant beaucoup évoluée depuis le début du travail, je vais redéfinir chaque élément de la machine pour pouvoir mieux comprendre les nouvelles règles.

Une suite de fonctions ont été écrite pour simplifier la lecture des règles. Les voici :

$\iota(SI)$ une fonction qui prends l'identifiant du dernier signal créer l'incréméte pour en créer un nouveau et retourner l'identifiant du signal créée avec la liste mise à jour.

Exemple : $\iota(\{..., \langle s, \langle emit, CS, SSI \rangle \rangle\}) = (s', \{..., \langle s, \langle emit, CS, SSI \rangle \rangle, \langle s', \langle false, \{\}, \{\} \rangle \rangle\})$ avec $s' = s + 1$

$SI(s)$ une fonction qui retourne le 2nd élément du couple correspondant à s.

Exemple : $SI(s) = \langle emit, CS, SSI, TL \rangle$

$\tau(SI)$ une fonction qui prends tous les éléments bloqués et les retourne en prenant en compte que le signal n'est pas émit, mets les liste de valeurs courantes en liste partagés si il est émit

Exemple : $\tau(TL, SI) = \forall si \in SI :$

- $\langle true, CS, SSI, \{\} \rangle \rightarrow \langle false, \{\}, CS, \{\} \rangle$

- $\langle false, CS, SSI, ST \rangle \rightarrow \langle false, \{\}, \{\}, \{\} \rangle$ et

$\forall st \in ST : \langle I, \langle \langle X', C'' \rangle, E \rangle \langle \langle X, C' \rangle, E \rangle s S, E, present C, D \rangle \rightarrow \langle I, S, E, C'' C, D \rangle$ et on l'ajoute à TL

$[(s, i) \leftarrow b]$ est une fonction qui mets dans la liste de valeurs ,de s pour le thread i, b et mets à vrai le booléen *emit*.

$SSI(s, i)$ une fonction qui retourne le triplet lié à un signal et un thread dans les signaux partagés.

Exemple : $SSI(s, i) = \langle i, CI, IL \rangle$

$CI(i)$ une fonction qui retourne la constante et décale l'itérateur

Soit $\langle T, TL, SI, IP \rangle$ **avec** :

TL = **une file de thread tels que** : $\forall tl \in TL \mid tl = T$ avec :

$T = \langle I, S, E, C, D \rangle$ **le thread courant avec** :

$V = b$

| $\langle \langle X, C' \rangle E \rangle$

| *signal*

I = un entier représentant l'identifiant du thread

$S = \emptyset$

| $V S$

$E = \{ \dots, \langle X, V \rangle, \dots \}$

$C = \epsilon$

| $b C$

(une constante)

| $X C$

(une variable)

| *signal* C

(un signal)

| $\langle X, C' \rangle C$

(une abstraction)

| *ap* C

(une application)

| *prim_{on}* C

(un opérateur)

| *spawn* C

(créateur d'un nouveau thread)

| *present* C

(le test de présence d'un signal)

| *init* C

(initialise un signal pour une chaîne de contrôle donné)

| *put* C

(insère une valeur dans la liste de valeurs d'un signal)

| *get* C

(prends une valeurs dans la liste de valeurs d'un signal)

$D = \emptyset$

| $\langle S, E, C, D \rangle$

(une sauvegarde liée à une abstraction)

SI = **une liste de signaux tels que** : $\forall si \in SI \mid si = \langle signal, \langle emit, CS, SSI, TL \rangle \rangle$ avec :

- un booléen représentant l'émission du signal : *emit*

- un identifiant de thread : I

- une liste des signaux courant tels que : $\forall cs \in CS \mid cs = \langle I, CL \rangle$ avec

- une liste de constante tels que : $\forall cl \in CL \mid cl = b$

- la liste des signaux partagés tels que : $\forall ssi \in SSI \mid ssi = \langle I, CI, IL \rangle$ avec

- une liste d'identifiant de threads : $\forall il \in IL \mid il = I$

- une liste de constante avec itérateur tels que : $\forall ci \in CI \mid ci = \langle b, IL \rangle$

IP = un entier servant à attribuer l'identifiant à un nouveau thread

On va définir une règle pour simplifier les règles futurs :

Dans tous les cas :

$$\frac{\langle S, E, C, D \rangle \rightarrow \langle S', E', C', D' \rangle}{\langle \langle I, S, E, C, D \rangle, TL, SI, IP \rangle \rightarrow \langle \langle I, S', E', C', D' \rangle, TL, SI, IP \rangle}$$

Si la règle utilisée n'est ni **Thread bloqué non remplacé** ni **Création thread** :

$$\frac{\langle \langle S, E, C, D \rangle, TL, SI \rangle \rightarrow \langle \langle S', E', C', D' \rangle, TL', SI' \rangle}{\langle \langle I, S, E, C, D \rangle, TL, SI, IP \rangle \rightarrow \langle \langle I, S', E', C', D' \rangle, TL', SI', IP \rangle}$$

Les éléments étant expliqués, voici les nouvelles règles de la machine :

Partie de base de la machine SECD

Constante : On a une constante, on la déplace dans la pile.

$$\langle S, E, n \ C, D \rangle \longrightarrow_{secdv5} \langle n \ S, E, C, D \rangle \text{ avec } n = b \text{ ou } signal$$

Substitution : On a une abstraction, on créer une fermeture avec celle-ci et l'environnement courant et on la place dans la pile.

$$\langle S, E, X \ C, D \rangle \longrightarrow_{secdv5} \langle V \ S, E, C, D \rangle \text{ avec } E(X) = V$$

Opération : On a un opérateur et le nombre de constante nécessaire dans la pile, via la fonction δ on retourne le résultat dans la pile.

$$\langle b_n, \dots, b_1 \ S, E, prim_{o^n} \ C, D \rangle \longrightarrow_{secdv5} \langle V \ S, E, C, D \rangle \text{ avec } \delta(o^n \ b_1 \dots b_n) = V$$

Abstraction : On a une abstraction, on créer une fermeture comportant l'abstraction et l'environnement courant et on mets la fermeture dans la pile.

$$\langle S, E, \langle X, C' \rangle \ C, D \rangle \longrightarrow_{secdv5} \langle \langle \langle X, C' \rangle, E \rangle \ S, E, C, D \rangle$$

Application : On a une application, donc on sauvegarde dans le dépôt, on ajoute une substitution et on remplace la chaîne de contrôle et l'environnement par ceux présent dans la fermeture.

$$\langle V \ \langle \langle X, C' \rangle, E' \rangle \ S, E, ap \ C, D \rangle \longrightarrow_{secdv5} \langle \emptyset, E'[X \leftarrow V], C', \langle S, E, C, D \rangle \rangle$$

Récupération de sauvegarde : On a rien mais le dépôt comporte une sauvegarde donc on prends celle-ci.

$$\langle V \ S, E, \epsilon, \langle S', E', C, D \rangle \rangle \longrightarrow_{secdv5} \langle V \ S', E', C, D \rangle$$

Partie pour la concurrence

Création thread : On veut créer un nouveau thread.

$$\langle \langle I, \langle \langle X, C' \rangle, E \rangle \ S, E, spawn \ C, D \rangle, TL, SI, IP \rangle \longrightarrow_{secdv5} \langle \langle I, S, E, C, D \rangle, TL \ \langle IP, S, E, C', D \rangle, SI, IP + 1 \rangle$$

Ajouter dans un signal : On ajoute une constante dans une liste de valeurs d'un signal et mets à vraie le booléen *emit*

$$\langle \langle I, s \ b \ S, E, put \ C, D \rangle, TL, SI \rangle \longrightarrow_{secdv5} \langle \langle I, S, E, C, D \rangle, TL, SI \ [(s, I) \leftarrow b] \rangle$$

à faire **Prendre une valeur partagée (possible) :** On prends dans la liste de valeurs d'un signal partagé lié à un thread et on décale l'itérateur.

$$\langle \langle I, s \ b \ n \ \langle \langle X, C' \rangle, E' \rangle \ S, E, get \ C, D \rangle, TL, SI \rangle \longrightarrow_{secdv5} \langle \langle I, \emptyset, E'[X \leftarrow V], C', \langle S, E, C, D \rangle \rangle, TL, SI \rangle$$

si $SSI(s, b) = \langle b, CI, IL \rangle$ avec $I \notin IL$ alors $CI(I) = V$

à faire **Prendre une valeur partagée (impossible) :** On prends dans la liste de valeurs d'un signal partagé lié à un identifiant une constante via la fonction γ mais on a déjà tout pris donc on lève une erreur.

$$\langle I, s \ b \ n \ \langle \langle X, C' \rangle, E' \rangle \ S, E, get \ C, TL, \langle CS, SSI \rangle, D, H, IP \rangle \longrightarrow_{secdv5} \langle I, n \ S, E, C, TL, \langle CS, SSI' \rangle, D, H, IP \rangle$$

avec $\gamma(SSI, s, I, b) = n$ si il reste aucune valeur à prendre et s partagé

Initialisation signal : On initialise le signal via la fonction ι .

$$\langle \langle I, S, E, init \ C, D \rangle, TL, SI \rangle \longrightarrow_{secdv4} \langle \langle I, signal \ S, E, C, D \rangle, TL, SI' \rangle \text{ avec } \iota(SI) = (signal, SI')$$

Présence du signal : On teste la présence d'un signal, via la fonction β on sait qu'il est émit donc on prends le 1er choix.

$$\langle \langle I, \langle \langle X', C'' \rangle, E \rangle \ \langle \langle X, C' \rangle, E \rangle \ s \ S, E, present \ C, D \rangle, TL, SI \rangle \longrightarrow_{secdv5} \langle \langle I, S, E, C' \ C, D \rangle, TL, SI \rangle$$

avec $SI(s) = \langle vraie, CS, SSI, TL \rangle$

à vérifier **Thread bloqué remplacé** : On teste la présence d'un signal, via la fonction β on sait qu'il n'est pas émit et il y a un thread dans la file d'attente donc on mets ce thread dans la liste de threads bloqués et on prends le thread en tête de la file.

$$\begin{aligned} & \langle \langle I, \langle \langle X', C'' \rangle, E \rangle \langle \langle X, C' \rangle, E \rangle s S, E, present C, D \rangle, \langle I', S', E', C''', D' \rangle TL, SI \rangle \\ & \longrightarrow_{secdv5} \langle \langle I', S', E', C''', D' \rangle, TL, SI' \rangle \\ & \text{avec } SI(s) = \langle faux, CS, SSI, TL' \rangle \\ & \text{et } SI'(s) = \langle faux, CS, SSI, TL' \langle I, \langle \langle X', C'' \rangle, E \rangle \langle \langle X, C' \rangle, E \rangle s S, E, present C, D \rangle \rangle \end{aligned}$$

à vérifier **Thread bloqué non remplacé** : On teste la présence d'un signal, via la fonction β on sait qu'il n'est pas émit donc on mets ce thread dans la liste de threads bloqués.

$$\begin{aligned} & \langle \langle I, \langle \langle X', C'' \rangle, E \rangle \langle \langle X, C' \rangle, E \rangle s S, E, C, D \rangle, \emptyset, SI, IP \rangle \longrightarrow_{secdv5} \langle \langle IP, \emptyset, \epsilon, \emptyset, \emptyset \rangle, \emptyset, SI', IP + 1 \rangle \\ & \text{avec } SI(s) = \langle faux, CS, SSI, TL' \rangle \\ & \text{et } SI'(s) = \langle faux, CS, SSI, TL' \langle I, \langle \langle X', C'' \rangle, E \rangle \langle \langle X, C' \rangle, E \rangle s S, E, present C, D \rangle \rangle \end{aligned}$$

Récupération dans la file d'attente : On a plus rien à traité et on a aucune sauvegarde, du coup on change le thread courant par le thread en tête de la file d'attente.

$$\langle \langle I, V S, E, \epsilon, \emptyset \rangle, \langle I', S', E', C, D \rangle TL, SI \rangle \longrightarrow_{secdv5} \langle \langle I', V S', E', C, D \rangle, TL, SI \rangle$$

Fin d'instant logique : On a plus rien à traiter, on a aucune sauvegarde et on a plus rien dans la file d'attente, c'est la fin d'un instant logique.

$$\begin{aligned} & \langle \langle I, V S, E, \epsilon, \emptyset \rangle, \emptyset, SI \rangle \longrightarrow_{secdv5} \langle \langle I, V S, E, \epsilon, \emptyset \rangle, TL, SI' \rangle \\ & \text{avec } \tau(SI) = (SI', TL) \text{ si il n'y a plus de thread bloqué} \end{aligned}$$

Partie commune

Application neutre : On a une application sur rien, cela revient juste à rien faire.

$$\langle S, E, ap C, D \rangle \longrightarrow_{secdv5} \langle S, E, C, D \rangle$$

la machine SECD version 4 peut s'arrêter dans 3 états différents :

on a une **constante b** tels que $\langle \langle 0, \emptyset, \emptyset, [M]_{secdv5}, \emptyset \rangle, \emptyset, \emptyset, 1 \rangle \twoheadrightarrow_{secdv5} \langle \langle I, b S, E, \epsilon, \emptyset \rangle, \emptyset, SI, IP \rangle ;$

on a une **abstraction function** tels que $\langle \langle 0, \emptyset, \emptyset, [M]_{secdv5}, \emptyset \rangle, \emptyset, \emptyset, 1 \rangle \twoheadrightarrow_{secdv5} \langle \langle I, \langle \langle X, C \rangle, E \rangle S, E, \epsilon, \emptyset \rangle, \emptyset, SI, IP \rangle ;$

on a une **erreur**

6 Conclusion

7 Annexes

7.1 Les Exemples des machines étudiées

7.1.1 Exemple de fonctionnement de la machine CC

Voici un exemple de fonctionnement de la machine CC :

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

7.1.2 Exemple de fonctionnement de la machine SCC

Voici un exemple de fonctionnement de la machine SCC :

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

7.1.3 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 : $\langle \lceil 2^\top \rceil, mt \rangle$

7.1.4 Exemple de fonctionnement de la machine CEK

Voici un exemple de fonctionnement de la machine CEK :

[illegible]

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

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

Conversion : $\langle f, \langle x, [f \ x]_{\text{secd}} \rangle \rangle \langle y, y \ y \ \text{prim}_+ \rangle \text{ ap } \lceil 1 \rceil \text{ ap}$

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

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

$$> \langle \widehat{S}, \widehat{\varepsilon}, \langle X, C' \rangle \widehat{C}, \widehat{D} \rangle \mapsto_{secd} \langle \langle X, C' \rangle, \varepsilon \rangle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D} \rangle$$
$$\text{SECD Machine : } \langle \langle \langle f, \langle x, f \ x \ ap \rangle \rangle, \emptyset \rangle, \emptyset, \langle y, y \ y \ prim_+ \rangle \ ap \top 1 \top ap, \epsilon \rangle$$
$$> \langle \hat{S}, \hat{\varepsilon}, \langle X, C' \rangle \hat{C}, \hat{D} \rangle \mapsto_{secd} \langle \langle X, C' \rangle, \varepsilon \rangle \hat{S}, \hat{\varepsilon}, \hat{C}, \hat{D} \rangle$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$> \langle b_1 \dots b_n \hat{S}, \hat{\varepsilon}, \text{prim}_{o^n} \hat{C}, \hat{D} \rangle \mapsto_{\text{secd}} \langle \hat{V} \hat{S}, \hat{\varepsilon}, \hat{C}, \hat{D} \rangle \text{ où } \hat{V} = \delta(o^n, b_1, \dots, b_n)$$

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

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

7.2 Les différentes versions faite pour rendre la machine SECD concurrente

7.2.1 1ère version des règles de la machine SECD Concurrente

Cette version ajoute les prémisses de la concurrence dans la machine SECD avec la possibilité de créer des threads le present et l'émission mais sans l'initialisation de la machine se qui la rend inutilisable. Ce point est rectifié dans la version 2.

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

$$\begin{aligned} \widehat{S} &= \dots \\ &| \langle C', C'' \rangle \widehat{S} \\ &| \text{Remp } \widehat{S} \\ \widehat{\varepsilon} &= \dots \\ \widehat{C} &= \dots \\ &| \text{bspawn } \widehat{C} \\ &| \text{espawn } \widehat{C} \\ &| \langle C', C'' \rangle \widehat{C} \\ &| \text{present}_s \widehat{C} \\ &| \text{emit}_s \widehat{C} \\ \widehat{W} &= \{\widehat{D}, \dots\} \\ \widehat{ST} &= \{\dots, \langle s, \widehat{D} \rangle, \dots\} \\ \widehat{SI} &= \{s, \dots\} \\ \widehat{D} &= \dots \end{aligned}$$

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. $\langle \widehat{V} \widehat{S}, \widehat{\varepsilon}, \emptyset, \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C}' \rangle, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{\text{secdv1}} \langle \widehat{V} \widehat{S}', \widehat{\varepsilon}', \widehat{C}', \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle$
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. $\langle \langle C', C'' \rangle \widehat{S}, \widehat{\varepsilon}, present_s \widehat{C}, \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C'''}, \widehat{D'} \rangle \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{secdv1} \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C'''}, \widehat{W}, \widehat{ST} \langle s, \langle \widehat{S}, \widehat{\varepsilon}, present_s \widehat{C}, \widehat{D} \rangle \rangle, \widehat{SI}, \widehat{D'} \rangle$
avec $s \notin \widehat{SI}$
14. $\langle \langle C', C'' \rangle \widehat{S}, \widehat{\varepsilon}, present_s \widehat{C}, \emptyset, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{secdv1} \langle \emptyset, \emptyset, \emptyset, \emptyset, \widehat{ST} \langle s, \langle \widehat{S}, \widehat{\varepsilon}, present_s \widehat{C}, \widehat{D} \rangle \rangle, \emptyset, \emptyset \rangle$
avec $s \notin \widehat{SI}$
15. $\langle \widehat{S}, \widehat{\varepsilon}, \emptyset, \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C'}, \widehat{D'} \rangle \widehat{W}, \widehat{ST}, \widehat{SI}, \emptyset \rangle \mapsto_{secdv1} \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C'}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D'} \rangle$
16. $\langle \widehat{S}, \widehat{\varepsilon}, \emptyset, \emptyset, \widehat{ST}, \widehat{SI}, \emptyset \rangle \mapsto_{secdv1} \langle \widehat{S}, \widehat{\varepsilon}, \emptyset, \widehat{W}, \emptyset, \emptyset, \emptyset \rangle$
avec $\widehat{W} =$ tout les éléments de \widehat{ST} qui prennent leurs 2nd choix
17. $\langle \widehat{S}, \widehat{\varepsilon}, emit_s \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \mapsto_{secdv1} \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W'}, \widehat{ST'}, \widehat{SI}, \widehat{D} \rangle$
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 4 états différents :

→ on a une **constante b** tels que $\langle \epsilon, \emptyset, [M]_{secdv1}, \emptyset, \emptyset, \emptyset, \epsilon \rangle \rightarrow_{secdv1} \langle b, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon \rangle ;$

→ on a une **abstraction function** tels que $\langle \epsilon, \emptyset, [M]_{secdv1}, \emptyset, \emptyset, \emptyset, \epsilon \rangle \rightarrow_{secdv1} \langle \langle \langle X, \widehat{C} \rangle, \widehat{\varepsilon}' \rangle, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon \rangle ;$

→ on a un **remplacement** tels que $\langle \epsilon, \emptyset, [M]_{secdv1}, \emptyset, \emptyset, \emptyset, \epsilon \rangle \rightarrow_{secdv1} \langle Remp, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon \rangle ;$

→ on a un **état inconnu** soit une **erreur**.

7.2.2 2ème version des règles de la machine SECD Concurrente

Cette version ajoute l'initialisation du signal promis dans la version 1 tout en modifiant un peu les éléments contenus dans la chaîne de contrôle.

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

$$\begin{aligned} \widehat{S} &= \dots \\ &| \langle C', C'' \rangle \widehat{S} \\ \widehat{\varepsilon} &= \dots \\ &| s \text{ (un signal initialisé)} \\ \widehat{C} &= \dots \\ &| [\langle C', C'' \rangle \widehat{C} \text{ present}_s \widehat{C}] \text{ changé en } \langle s, C', C'' \rangle \widehat{C} \\ &| \langle s, \widehat{C}' \rangle \widehat{C} \\ \widehat{W} &= \dots \\ \widehat{ST} &= \dots \\ \widehat{SI} &= \dots \\ \widehat{D} &= \dots \end{aligned}$$

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 4 états différents :

- on a une **constante** \mathbf{b} tels que $\langle \epsilon, \emptyset, [M]_{secdv2}, \emptyset, \emptyset, \emptyset, \epsilon \rangle \mapsto_{secdv2} \langle \mathbf{b}, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon \rangle$;
- on a une **abstraction function** tels que $\langle \epsilon, \emptyset, [M]_{secdv2}, \emptyset, \emptyset, \emptyset, \epsilon \rangle \mapsto_{secdv2} \langle \langle \langle X, \widehat{C} \rangle, \widehat{\varepsilon}' \rangle, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon \rangle$;
- on a un **remplacement** tels que $\langle \epsilon, \emptyset, [M]_{secdv2}, \emptyset, \emptyset, \emptyset, \epsilon \rangle \mapsto_{secdv2} \langle \text{Remp}, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon \rangle$;
- on a un **état inconnu** soit une **erreur**.

7.2.3 3ème version des règles de la machine SECD Concurrente

Cette version ajoute le contrôle des erreurs sans propagation via l'ajout d'un gestionnaire d'erreur dans la machine.

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

$$\begin{aligned}
 \widehat{S} &= \dots \\
 \widehat{\varepsilon} &= \dots \\
 \widehat{C} &= \dots \\
 &\quad | \text{throw}_e \widehat{C} \\
 &\quad | \langle e, \langle \widehat{C}', \langle X, \widehat{C}'' \rangle \rangle \rangle \widehat{C} \\
 \widehat{W} &= \dots \\
 \widehat{ST} &= \dots \\
 \widehat{SI} &= \dots \\
 \widehat{D} &= \dots \\
 \widehat{H} &= \epsilon \\
 &\quad | \langle e, \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \rangle \\
 \widehat{V} &= \dots \\
 &\quad | \text{erreur}_e
 \end{aligned}$$

Les nouvelles règles sont les suivantes :

1. $\langle \widehat{S}, \widehat{\varepsilon}, b \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \mapsto_{\text{secdv3}} \langle b \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$
2. $\langle \widehat{S}, \widehat{\varepsilon}, X \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \mapsto_{\text{secdv3}} \langle \widehat{V} \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$
où $\widehat{V} = \varepsilon(X)$
3. $\langle b_1 \dots b_n \widehat{S}, \widehat{\varepsilon}, \text{prim}_{o^n} \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \mapsto_{\text{secdv3}} \langle \widehat{V} \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$
où $\widehat{V} = \delta(o^n, b_1, \dots b_n)$
4. $\langle \widehat{S}, \widehat{\varepsilon}, \text{throw}_e \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \mapsto_{\text{secdv3}} \langle \text{erreur}_e \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$
5. $\langle \widehat{S}, \widehat{\varepsilon}, \langle X, C' \rangle \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \mapsto_{\text{secdv3}} \langle \langle \langle X, C' \rangle, \varepsilon \rangle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$
6. $\langle \widehat{V} \langle \langle X, C' \rangle, \varepsilon' \rangle \widehat{S}, \widehat{\varepsilon}, \text{ap} \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \mapsto_{\text{secdv3}} \langle \epsilon, \varepsilon' [X \leftarrow \widehat{V}], C', \widehat{W}, \widehat{ST}, \widehat{SI}, \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D} \rangle, \widehat{H} \rangle$
7. $\langle \widehat{V} \widehat{S}, \widehat{\varepsilon}, \emptyset, \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C}' \rangle, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \mapsto_{\text{secdv3}} \langle \widehat{V} \widehat{S}', \widehat{\varepsilon}', \widehat{C}', \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$
8. $\langle \widehat{V} \text{Remp} \widehat{S}, \widehat{\varepsilon}, \text{ap} \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \mapsto_{\text{secdv3}} \langle \widehat{V} \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$
9. $\langle \text{Remp} \widehat{V} \widehat{S}, \widehat{\varepsilon}, \text{ap} \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \mapsto_{\text{secdv3}} \langle \widehat{V} \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$
10. $\langle \text{Remp} \widehat{V} \widehat{S}, \widehat{\varepsilon}, \text{ap} \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \mapsto_{\text{secdv3}} \langle \widehat{V} \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$
11. $\langle \widehat{S}, \widehat{\varepsilon}, \text{bspawn} \widehat{C}' \text{ spawn} \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \mapsto_{\text{secdv3}} \langle \text{Remp} \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W} \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}', \widehat{D} \rangle, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$
12. $\langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \mapsto_{\text{secdv3}} \langle \widehat{S}, \widehat{\varepsilon}, C' \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$
si $s \in \widehat{SI}$ et $s \in \widehat{\varepsilon}$
13. $\langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle \widehat{C}, \langle \widehat{S}', \widehat{\varepsilon}', C''', \widehat{D}' \rangle \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \mapsto_{\text{secdv3}} \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C}''', \widehat{W}, \widehat{ST} \langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle \widehat{C}, \widehat{D} \rangle, \widehat{SI}, \widehat{D}', \widehat{H} \rangle$
si $s \notin \widehat{SI}$ et $s \in \widehat{\varepsilon}$

14. $\langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle \widehat{C}, \emptyset, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \mapsto_{secdv3} \langle \epsilon, \emptyset, \emptyset, \emptyset, \widehat{ST} \langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle \widehat{C}, \widehat{D} \rangle, \widehat{SI}, \emptyset, \widehat{H} \rangle$
si $s \notin \widehat{SI}$ et $s \in \widehat{\varepsilon}$
15. $\langle \widehat{S}, \widehat{\varepsilon}, \langle s, C' \rangle \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \mapsto_{secdv3} \langle \epsilon, \widehat{\varepsilon}[\text{init} \leftarrow s], C', \widehat{W}, \widehat{ST}, \widehat{SI}, \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D} \rangle, \widehat{H} \rangle$
16. $\langle \widehat{S}, \widehat{\varepsilon}, \emptyset, \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C}', \widehat{D}' \rangle \widehat{W}, \widehat{ST}, \widehat{SI}, \emptyset, \widehat{H} \rangle \mapsto_{secdv3} \langle \widehat{S}', \widehat{\varepsilon}', \widehat{C}', \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}', \widehat{H} \rangle$
17. $\langle \widehat{S}, \widehat{\varepsilon}, \emptyset, \emptyset, \widehat{ST}, \widehat{SI}, \emptyset, \widehat{H} \rangle \mapsto_{secdv3} \langle \widehat{S}, \widehat{\varepsilon}, \emptyset, \widehat{W}, \emptyset, \emptyset, \emptyset, \widehat{H} \rangle$
avec $\widehat{W} =$ tout les éléments de \widehat{ST} qui prennent leurs 2nd choix
18. $\langle \widehat{S}, \widehat{\varepsilon}, \text{emit}_s \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \mapsto_{secdv3} \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}', \widehat{ST}', \widehat{SI}, \widehat{D}, \widehat{H} \rangle$
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
19. $\langle \widehat{S}, \widehat{\varepsilon}, \langle e, \langle \widehat{C}', \langle X, \widehat{C}'' \rangle \rangle \rangle \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \mapsto_{secdv3} \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}' \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \langle e, \langle \widehat{S}, \widehat{\varepsilon}, \langle X, \widehat{C}'' \rangle \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \rangle \rangle$
20. $\langle \widehat{S}, \widehat{\varepsilon}, \text{throw}_e \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \langle e, \langle \widehat{S}', \widehat{\varepsilon}', \langle X, \widehat{C}'' \rangle \widehat{C}', \widehat{W}', \widehat{ST}', \widehat{SI}', \widehat{D}', \widehat{H} \rangle \rangle \rangle$
 $\mapsto_{secdv3} \langle \widehat{S}', \widehat{\varepsilon}', \langle X, \widehat{C}'' \rangle \text{throw}_e \widehat{C}, \widehat{W}', \widehat{ST}', \widehat{SI}', \widehat{D}', \widehat{H} \rangle$
21. $\langle \widehat{S}, \widehat{\varepsilon}, \text{throw}_e \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \langle e', \langle \widehat{S}', \widehat{\varepsilon}', \langle X, \widehat{C}'' \rangle \widehat{C}', \widehat{W}', \widehat{ST}', \widehat{SI}', \widehat{D}', \widehat{H} \rangle \rangle \rangle$
 \mapsto_{secdv3} on regarde dans \widehat{H} récursivement si il y a un gestionnaire pour l'erreur e
22. $\langle \widehat{S}, \widehat{\varepsilon}, \text{throw}_e \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \emptyset \rangle \mapsto_{secdv3} \langle \widehat{S}, \widehat{\varepsilon}, \text{throw}_e, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$

la machine SECD version 3 peut s'arrêter dans 4 états différents :

- on a une **constante** b tels que $\langle \epsilon, \emptyset, [M]_{secdv3}, \emptyset, \emptyset, \emptyset, \epsilon, \emptyset \rangle \mapsto_{secdv3} \langle b, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon, \widehat{H} \rangle$;
- on a une **abstraction function** tels que $\langle \epsilon, \emptyset, [M]_{secdv3}, \emptyset, \emptyset, \emptyset, \epsilon, \emptyset \rangle \mapsto_{secdv3} \langle \langle \langle X, \widehat{C} \rangle, \widehat{\varepsilon}' \rangle, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon, \widehat{H} \rangle$;
- on a un **remplacement** tels que $\langle \epsilon, \emptyset, [M]_{secdv3}, \emptyset, \emptyset, \emptyset, \epsilon, \emptyset \rangle \mapsto_{secdv3} \langle \text{Remp}, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon, \widehat{H} \rangle$;
- on a une **erreur** e tels que $\langle \epsilon, \emptyset, [M]_{secdv3}, \emptyset, \emptyset, \emptyset, \epsilon, \emptyset \rangle \mapsto_{secdv3} \langle \text{erreur}_e, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon, \widehat{H} \rangle$;

7.2.4 4 ème version de la machine SECD concurrente

La machine ayant beaucoup évoluée depuis le début du travail, je vais redéfinir chaque élément de la machine pour pouvoir mieux comprendre les nouvelles règles.

Une suite de fonctions ont été écrite pour simplifier la lecture des règles. Les voici :

$\rho(l, v, s, i) =$ la fonction qui pour une liste des signaux l , une valeur v , un signal s et un identifiant du thread courant donnés, renvoie la liste l' avec v ajouté à la liste des valeurs du signal s pour le thread i .

Exemple : $\rho(\{..., \langle s, \{..., \langle id, \text{valeur} \rangle, ... \rangle, \text{emit} \rangle, ... \}, v, s, id) = \{..., \langle s, \{..., \langle id, \text{valeur } v \rangle, ... \rangle, \text{emit} \rangle, ... \}$

$\gamma(l, s, i, i') =$ la fonction qui pour une liste de valeurs partagées classés par signaux et par thread l , un signal s , l'identifiant du thread courant et l'identifiant du thread auxquels on veut accéder, renvoie soit un couple la liste avec l'itérateur déplacé et la valeur ou une exception si on ne peut plus donner de nouvelles valeurs

Exemple :

$\gamma(\{..., \langle s, \{..., \langle id, \{..., \langle b, \{..., id', ... \rangle, \langle n, \{..., id' \rangle, ... \rangle, \{..., id' \rangle, ... \rangle, ... \rangle, ... \rangle, s, id, id' \rangle = \langle b, \{..., \langle s, \{..., \langle id, \{..., \langle b, \{..., id' \rangle, \langle n, \{..., id' \rangle, ... \rangle, \{..., id' \rangle, ... \rangle, ... \rangle, ... \rangle \rangle \rangle$

$\gamma(\{..., \langle s, \{..., \langle id, \{..., \langle b, \{..., id', ... \rangle, \{..., id' \rangle, ... \rangle, ... \rangle, ... \rangle, s, id, id' \rangle = \langle b, \{..., \langle s, \{..., \langle id, \{..., \langle b, \{..., id' \rangle, \{..., id' \rangle, ... \rangle, \{..., id' \rangle, ... \rangle, ... \rangle \rangle \rangle$

$\gamma(\{..., \langle s, \{..., \langle id, valeurs, \{..., id', ... \}, ... \rangle, ... \}, ..., s, id, id') = throw\ erreur_e$

$\iota(l, s, i)$ = la fonction qui pour une liste de signaux courant l, un signal s, renvoie une liste des signaux courant avec le signal s initialisé.

Exemple : $\iota(\{...\}, s) = \{..., \langle s, \{\}, false \rangle\}$

$\beta(l, s)$ = la fonction qui pour une liste de signal courant l et un signal s donnés, renvoie le booléen émit.

Exemple :

$\beta(\{..., \langle s, \{...\}, vraie \rangle, ... \}, s) = vraie$

$\beta(\{..., \langle s, \{...\}, faux \rangle, ... \}, s) = faux$

$\varepsilon(l, s)$ = la fonction qui pour une liste de signaux courant l et un signal s donnés, renvoie la liste avec le booléen représentant l'émission du signal à vraie.

Exemple :

$\varepsilon(\{..., \langle s, \{...\}, faux \rangle, ... \}, s) = \{..., \langle s, \{...\}, vraie \rangle, ... \}$

$\alpha(\langle CS, SSI \rangle)$ = la fonction qui pour la liste des signaux courant et la liste des signaux partagées données, renvoie la liste des signaux courant vidées de ses listes de valeurs et le booléen représentant l'émission mis à nulle ainsi que la liste des signaux partagées remplacer par les listes de valeurs de la liste des signaux courants qui sont émit.

Exemple :

$\alpha(\langle CS, SSI \rangle) =$

SSI vidé

$\forall x \in CS$ tels que $\langle s, \{..., \langle id, \{..., b, ... \}, ... \}, true \rangle$, on ajoute x dans SSI

$\forall x \in CS$ tels que $\langle s, \{..., \langle id, \{..., b, ... \}, ... \}, emit \rangle$ on le remplace par $\langle s, \{..., \langle id, \{\}, ... \}, faux \rangle$

Soit $\langle I, S, E, C, TL, SI, D, H, IP \rangle$ **avec** :

$V = b$

| $\langle \langle X, C' \rangle E \rangle$

| $erreur_e$

$I =$ un entier représentant l'identifiant du thread

$S = \emptyset$

| VS

| $signal\ S$

| $throw\ S$

$E = \{..., \langle X, V \rangle, ...\}$

$C = \epsilon$

| $b\ C$ (une constante)

| $X\ C$ (une variable)

| $signal\ C$ (un signal)

| $\langle X, C' \rangle\ C$ (une abstraction)

| $ap\ C$ (une application)

| $prim_{o^n}\ C$ (un opérateur)

| $bspawn\ C$ (début d'un nouveau thread)

| $espawn\ C$ (fin d'un nouveau thread)

| $\langle C', C'' \rangle\ C$ (le test de présence d'un signal)

| $emit\ C$ (émet un signal)

| $init\ C$ (initialise un signal pour une chaîne de contrôle donné)

| $put\ C$ (insère une valeur dans la liste de valeurs d'un signal)

| $get\ C$ (prends une valeurs dans la liste de valeurs d'un signal)

| $erreur_e\ C$ (une erreur)

| $throw\ C$ (lève une erreur)

| $\langle C', \langle X, C'' \rangle \rangle\ C$ (un gestionnaire d'erreur)

$TL = \langle W, ST \rangle$

$W = \{..., \langle I, S, E, C, D \rangle, ...\}$ (liste des threads en attente)

$ST = \{..., \langle s, \langle I, S, E, C, D \rangle \rangle, ...\}$ (liste des threads en attente d'un signal)

$SI = \langle CS, SSI \rangle$

$CS = \{..., \langle s, \{..., \langle id, \{..., b, ... \} \rangle, ... \rangle, emit \rangle, ...\}$ (liste des signaux courants)

on va découper cette élément pour mieux en comprendre le sens :

- $\{..., *, ... \}$ Une liste.

- $\langle s, \{..., **, ... \} \rangle, emit \rangle$

Une liste composée de trinôme comportant le identifiant du signal, une sous-liste et un booléen exprimant l'émission de ce signal.

- $\langle id, \{..., b, ... \} \rangle$

Une sous-liste composée d'un trinôme comportant l'identifiant du thread et une liste de valeur.

$SSI = \{..., \langle s, \{..., \langle id, \{..., \langle b, \{..., id', ... \} \rangle, ... \rangle, \{..., id'', ... \} \rangle, ... \rangle, ... \rangle, ... \}$ (liste des signaux partagés)

comme pour CS on va découper cette élément pour pouvoir le comprendre :

- $\{..., *, ... \}$ Une liste.

- $\langle s, \{..., **, ... \} \rangle$

Une liste composée d'un couple comportant un identifiant de signal et d'une sous-liste

- $\langle id, \{..., *, *, ..., ... \} \rangle, \{..., id'', ... \}$

Une sous-liste composée d'un trinôme comportant un identifiant d'un thread, d'un liste et d'une sous-sous-liste d'identifiant de thread représentant la liste des threads ayant fini leurs parcours de la sous-sous-liste.

- $\langle b, \{..., id', ... \} \rangle$

Une sous-sous-liste composée d'un couple comportant une valeur et une liste d'identifiant de threads qui représente un pointeur

$D = \emptyset$

| $\langle S, E, C, D \rangle$ (une sauvegarde liée à une abstraction)

$H = \emptyset$

| $\langle e \langle I, S, E, \langle X, C' \rangle C, TL, SI, D, H, IP \rangle \rangle$

$IP =$ un entier servant à attribuer l'identifiant à un nouveau thread

Les éléments étant expliqués, voici les nouvelles règles de la machine :

Partie de base de la machine SECD

Constante : On a une constante, on la déplace dans la pile.

$$\langle I, S, E, b \ C, TL, SI, D, H, IP \rangle \longrightarrow_{secdv4} \langle I, b \ S, E, C, TL, SI, D, H, IP \rangle$$

Substitution : On a une abstraction, on créer une fermeture avec celle-ci et l'environnement courant et on la place dans la pile.

$$\langle I, S, E, X \ C, TL, SI, D, H, IP \rangle \longrightarrow_{secdv4} \langle I, V \ S, E, C, TL, SI, D, H, IP \rangle$$

avec $E(X) = V$

Opération : On a un opérateur et le nombre de constante nécessaire dans la pile, via la fonction δ on retourne le résultat dans la pile.

$$\langle I, b_n, \dots, b_1 \ S, E, prim_{o^n} \ C, TL, SI, D, H, IP \rangle \longrightarrow_{secdv4} \langle I, V \ S, E, C, TL, SI, D, H, IP \rangle$$

avec $\delta(o^n \ b_1 \dots b_n) = V$

Abstraction : On a une abstraction, on créer une fermeture comportant l'abstraction et l'environnement courant et on mets la fermeture dans la pile.

$$\langle I, S, E, \langle X, C' \rangle \ C, TL, SI, D, H, IP \rangle \longrightarrow_{secdv4} \langle I, \langle \langle X, C' \rangle, E \rangle \ S, E, C, TL, SI, D, H, IP \rangle$$

Application : On a une application, donc on sauvegarde dans le dépôt, on ajoute une substitution et on remplace la chaîne de contrôle et l'environnement par ceux présent dans la fermeture.

$$\langle I, V \ \langle \langle X, C' \rangle, E' \rangle \ S, E, ap \ C, TL, SI, D, H, IP \rangle \longrightarrow_{secdv4} \langle I, \emptyset, E'[X \leftarrow V], C', TL, SI, \langle S, E, C, D \rangle, H, IP \rangle$$

Récupération de sauvegarde : On a rien mais le dépôt comporte une sauvegarde donc on prends celle-ci.

$$\langle I, V \ S, E, \epsilon, TL, SI, \langle S', E', C, D \rangle, H, IP \rangle \longrightarrow_{secdv4} \langle I, V \ S', E', C, TL, SI, D, H, IP \rangle$$

Partie pour les erreurs

Erreur : On a une erreur, on la déplace en tête de la pile.

$$\langle I, S, E, erreur_e \ C, TL, SI, D, H, IP \rangle \longrightarrow_{secdv4} \langle I, erreur_e \ S, E, C, TL, SI, D, H, IP \rangle$$

Lever erreur : On a un throw, on le déplace en tête de la pile.

$$\langle I, S, E, throw \ C, TL, SI, D, H, IP \rangle \longrightarrow_{secdv4} \langle I, throw \ S, E, C, TL, SI, D, H, IP \rangle$$

Opération sur erreur : On a l'opérateur qui traite cette erreur donc on mets le résultat de la fonction δ dans la pile.

$$\langle I, throw \ erreur_e \ S, E, prim_{o^{1e}} \ C, TL, SI, D, H, IP \rangle \longrightarrow_{secdv4} \langle I, V \ S, E, C, TL, SI, D, H, IP \rangle$$

avec $\delta(o^{1e} \ erreur_e) = V$

Propagation : On a un un élément excepté l'opérateur qui traite cette erreur donc on propage l'erreur.

$$\langle I, throw \ erreur_e \ S, E, M \ C, TL, SI, D, H, IP \rangle \longrightarrow_{secdv4} \langle I, throw \ erreur_e \ S, E, C, TL, SI, D, H, IP \rangle$$

avec $M = \text{un élément de } C \setminus prim_{o^{1e}}$

Traitée erreur via gestionnaire d'erreur : On a plus rien mais on a une erreur levé dans la pile du coup on regarde si le gestionnaire d'erreur gère celle-ci ; oui du coup prend la sauvegarde.

$$\langle I, throw \ erreur_e \ S, E, \epsilon, TL, SI, D, \langle e, \langle I', S', E', \langle X, C'' \rangle C', TL', SI', D', H, IP' \rangle \rangle, IP \rangle \longrightarrow_{secdv4} \langle I', \emptyset, E'[X \leftarrow erreur_e], C'', TL', SI', \langle S', E', C', D' \rangle, H, IP' \rangle$$

Traitement erreur récursif : On a plus rien mais on a une erreur levé dans la pile du coup on regarde si le gestionnaire d'erreur gère celle-ci mais non du coup on regarde pour le gestionnaire sauvegardé.

$$\langle I, throw \ erreur_e \ S, E, \epsilon, TL, SI, D, \langle e', \langle I', S', E', \langle X, C'' \rangle C', TL', SI', D', H, IP' \rangle \rangle, IP \rangle \longrightarrow_{secdv4} \langle I, throw \ erreur_e \ S, E, \epsilon, TL, SI, D, H, IP \rangle$$

Création d'un gestionnaire d'erreur : On a un try...catch donc on test avec la chaîne de contrôle du try et on sauvegarde catch dans le gestionnaire d'erreur.

$$\langle I, erreur_e S, E, \langle C', \langle X, C'' \rangle \rangle C, TL, SI, D, H, IP \rangle \\ \rightarrow_{secdv4} \langle I, S, E, C' C, TL, SI, D, \langle e, \langle I, erreur_e S, E, \langle X, C'' \rangle C, TL, SI, D, H, IP \rangle \rangle, IP \rangle$$

Partie pour la concurrence

Création thread : On veut créer un nouveau thread.

$$\langle I, S, E, bspawn C' espawn C, \langle W, ST \rangle, SI, D, H, IP \rangle \\ \rightarrow_{secdv4} \langle I, S, E, C, \langle W \langle IP, S, E, C', D \rangle, ST \rangle, SI, D, H, IP + 1 \rangle$$

Signal : On a un signal, on le déplace dans la pile.

$$\langle I, S, E, signal C, TL, SI, D, H, IP \rangle \rightarrow_{secdv4} \langle I, signal S, E, C, TL, SI, D, H, IP \rangle$$

Ajouter dans un signal : On ajoute une constante dans une liste de valeurs d'un signal via la fonction ρ

$$\langle I, s b S, E, put C, TL, \langle CS, SSI \rangle, D, H, IP \rangle \rightarrow_{secdv4} \langle I, S, E, C, TL, \langle CS', SSI \rangle, D, H, IP \rangle$$

avec $CS' = \rho(CS, b, s, I)$ et s initialisé

Prendre une valeur partagée (possible) : On prends dans la liste de valeurs d'un signal partagé lié à un identifiant une constante via la fonction γ .

$$\langle I, s b \langle \langle X, C' \rangle, E' \rangle S, E, get C, TL, \langle CS, SSI \rangle, D, H, IP \rangle \\ \rightarrow_{secdv4} \langle I, \emptyset, E'[X \leftarrow V], C', TL, \langle CS, SSI' \rangle, \langle S, E, C, D \rangle, H, IP \rangle$$

avec $\gamma(SSI, s, I, b) = \langle V, SSI' \rangle$ si il reste une valeur à prendre et s partagé

Prendre une valeur partagée (impossible) : On prends dans la liste de valeurs d'un signal partagé lié à un identifiant une constante via la fonction γ mais on a déjà tout pris donc on lève une erreur.

$$\langle I, s b \langle \langle X, C' \rangle, E' \rangle S, E, get C, TL, \langle CS, SSI \rangle, D, H, IP \rangle \\ \rightarrow_{secdv4} \langle I, throw erreur_e S, E, C, TL, \langle CS, SSI' \rangle, D, H, IP \rangle$$

avec $\gamma(SSI, s, I, b) = throw erreur_e$ si il reste aucune valeur à prendre et s partagé

Initialisation signal : On initialise le signal via la fonction ι .

$$\langle I, s S, E, init C, TL, \langle CS, SSI \rangle, D, H, IP \rangle \rightarrow_{secdv4} \langle I, S, E, C, TL, \langle CS', SSI \rangle, D, H, IP \rangle$$

avec $\iota(CS, s) = CS'$

Conditionnelle signal : On teste la présence d'un signal, via la fonction β on sait qu'il est émit donc on prends le 1er choix.

$$\langle I, s S, E, \langle C', C'' \rangle C, TL, \langle CS, SSI \rangle, D, H, IP \rangle \rightarrow_{secdv4} \langle I, S, E, C' C, TL, \langle CS, SSI \rangle, D, H, IP \rangle$$

avec $\beta(CS, s) = vraie$

Thread bloqué remplacé : On teste la présence d'un signal, via la fonction β on sait qu'il n'est pas émit et il y a un thread dans la file d'attente donc on mets ce thread dans la liste de threads bloqués et on prends le thread en tête de la file.

$$\langle I, s S, E, \langle C', C'' \rangle C, \langle \langle I', S', E', C''', D' \rangle W, ST \rangle, \langle CS, SSI \rangle, D, H, IP \rangle \\ \rightarrow_{secdv4} \langle I', S', E', C''', \langle W, ST \langle s, \langle I, s S, E, \langle C', C'' \rangle C, D \rangle \rangle \rangle, \langle CS, SSI \rangle, D', H, IP \rangle$$

avec $\beta(CS, s) = faux$

Thread bloqué non remplacé : On teste la présence d'un signal, via la fonction β on sait qu'il n'est pas émit donc on mets ce thread dans la liste de threads bloqués.

$$\langle I, s S, E, \langle C', C'' \rangle C, \langle \emptyset, ST \rangle, \langle CS, SSI \rangle, D, H, IP \rangle \\ \rightarrow_{secdv4} \langle IP, \emptyset, \epsilon, \emptyset, \langle W, ST \langle s, \langle I, s S, E, \langle C', C'' \rangle C, D \rangle \rangle \rangle, \langle CS, SSI \rangle, \emptyset, H, IP + 1 \rangle$$

avec $\beta(CS, s) = faux$

Émission : On émet un signal donc on mets dans la file d'attente tous les threads attendant le signal.

$\langle I, s, S, E, emit\ C, TL, \langle CS, SSI \rangle, D, H, IP \rangle \rightarrow_{secdv4} \langle I, Unit\ S, E, C, TL', \langle CS', SSI \rangle, D, H, IP \rangle$
avec $\varepsilon(CS, s) = CS'$ et $TL' = \langle W', ST' \rangle$ et $TL = \langle W, ST \rangle$:

$W' = W \cup$ les éléments de ST qui attendent le signal s

$ST' = ST \setminus$ les éléments de ST qui attendent le signal s

Récupération dans la file d'attente : On a plus rien à traité et on a aucune sauvegarde, du coup on change le thread courant par le thread en tête de la file d'attente.

$\langle I, V\ S, E, \epsilon, \langle \langle I', S', E', C, D \rangle W, ST \rangle, SI, \emptyset, H, IP \rangle \rightarrow_{secdv4} \langle I', V\ S', E', C, \langle W, ST \rangle, SI, D, H, IP \rangle$

Fin d'instant logique : On a plus rien à traiter, on a aucune sauvegarde et on a plus rien dans la file d'attente, c'est la fin d'un instant logique.

$\langle I, V\ S, E, \epsilon, \langle \emptyset, ST \rangle, SI, \emptyset, H, IP \rangle \rightarrow_{secdv4} \langle I, V\ S, E, \epsilon, \langle W, \emptyset \rangle, SI', \emptyset, H, IP \rangle$

avec $W = ST$ avec tous ces éléments qui prennent en compte l'absence de l'émission du signal attendu et $\alpha(SI) = SI'$

Partie commune

Application neutre : On a une application sur rien, cela revient juste à rien faire.

$\langle I, S, E, ap\ C, TL, SI, D, H, IP \rangle \rightarrow_{secdv4} \langle I, S, E, C, TL, SI, D, H, IP \rangle$

la machine SECD version 4 peut s'arrêter dans 3 états différents :

on a une **constante b** tels que $\langle 0, \emptyset, \emptyset, [M]_{secdv4}, \langle \emptyset, \emptyset \rangle, \langle \emptyset, \emptyset \rangle, \emptyset, \emptyset, \emptyset, 1 \rangle$
 $\rightarrow_{secdv4} \langle I, b\ S, E, \epsilon, \langle \emptyset, \emptyset \rangle, SI, \emptyset, H, IP \rangle$;

on a une **abstraction function** tels que $\langle 0, \emptyset, \emptyset, [M]_{secdv4}, \langle \emptyset, \emptyset \rangle, \langle \emptyset, \emptyset \rangle, \emptyset, \emptyset, \emptyset, 1 \rangle$
 $\rightarrow_{secdv4} \langle I, \langle \langle X, C \rangle, E' \rangle\ S, E, \epsilon, \langle \emptyset, \emptyset \rangle, SI, \emptyset, H, IP \rangle$;

on a une **erreur e** tels que $\langle 0, \emptyset, \emptyset, [M]_{secdv4}, \langle \emptyset, \emptyset \rangle, \langle \emptyset, \emptyset \rangle, \emptyset, \emptyset, \emptyset, 1 \rangle$
 $\rightarrow_{secdv4} \langle I, erreur_e\ S, E, \epsilon, \langle \emptyset, \emptyset \rangle, SI, \emptyset, H, IP \rangle$;

8 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
- [4] <https://www.irif.fr/~carton/Enseignement/Complexite/MasterInfo/Cours/turing.html>
- [5] https://fr.wikipedia.org/wiki/Automate_fini_déterministe
- [6] https://fr.wikipedia.org/wiki/Algorithme_déterministe

9 Journal de bord

9.1 Mars

9.1.1 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.

9.2 Avril

9.2.1 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 terminée 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.

9.2.2 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 :

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

$$\widehat{ST} = \{\langle s, \widehat{D} \rangle, \dots\} \text{ avec } s \text{ un signal}$$

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 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.

9.2.3 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 privilégiera 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 ne 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.

9.2.4 Semaine du 22 au 26

L'implémentation de la nouvelle structure a pris le mardi au complet et le mercredi matin. Mais manquait de plusieurs choses et avait des failles.

La liste de signal \widehat{SI} était un bon commencement mais il me manquait la liste de valeurs partagées. Pour cela \widehat{SI} va devenir un couple de liste de la forme $\langle \widehat{CS}, \widehat{SSI} \rangle$. Les signaux courants comme 1ère élément qui a la structure de l'ancien \widehat{SI} et le 2nd élément qui va regrouper les valeurs partagées. Avant de parler de sa structure, il faut remettre en question la structure de \widehat{CS} .

Après avoir fini l'implémentation et mis en phase de test la machine j'ai remarqué 2 problèmes :

- trier par threads puis par signaux est moins pratique que trier par signaux puis par threads. Du coup la structure de \widehat{CS} sera $\{\langle signal, \{\langle id, \{valeur, \dots\}, init \rangle\}, emit \rangle, \dots\}$.
- quand on crée un nouveau thread, j'avais omis de dupliqué les signaux initialisé plus tôt pour le nouveau thread créé dans la liste des signaux. Ceci a été fixé.

Une 4ème version a donc été écrit avec les put et get et les améliorations et peut être retrouvé dans la Dernière version de la machine SECD concurrente. La définition des éléments de la machine y sont aussi. Toutes ces modifications ont été faites durant le reste de la semaine.

9.3 Mai

9.3.1 Semaine du 29 Avril au 3

Ayant terminé l'implémentation de la 4ème version, les tests ont été fait le lundi en attendant la réunion hebdomadaire. La réunion a été reporté du coup par mail j'ai demandé la suite du travail à produire. Madame Bousdira m'a donc répondu et voici le nouvelle objectif :

1. Faire une preuve prouvant le déterminisme de la machine

Ne sachant pas comment mis prendre, les recherches ont été faite sur internet et j'ai trouvé cet article[4] montrant le déterminisme de la machine de Turing par un automate déterministe. Les différentes tentatives fut des échecs car il me manquait la différenciation entre le déterminisme d'un automate et d'une machine, je m'explique :

Définition du déterminisme d'un automate : Un automate fini déterministe, parfois abrégé en AFD (en anglais deterministic finite automaton, abrégé en DFA) est un automate fini dont les transitions à partir de chaque état sont déterminées de façon unique par le symbole d'entrée. [5]

Définition du déterminisme d'une machine via un algorithme : Un algorithme déterministe est un algorithme qui, étant donné une entrée particulière, produira toujours la même sortie, avec la machine sous-jacente passant toujours par la même séquence d'états.

Formellement, un algorithme déterministe calcule une fonction mathématique ; une fonction ayant une valeur unique pour n'importe quelle entrée dans son ensemble de définition, l'algorithme produit cette valeur en sortie. [6]

Ce problème de compréhension a été détecté par Mme Bousdira le jeudi matin et réglé le jeudi après-midi avec Mme Bousdira et Mr Dabrowski. Une preuve par induction a été indiqué comme une solution.

Avant cela la réécriture des nouvelles règles au propre a été nécessaire et fait le vendredi pour que mes professeurs puissent vérifier mon travail avant de commencer la preuve.

9.3.2 Semaine du 6 au 10

La réunion s'est déroulé le lundi matin. Les règles ont été vérifié par mes professeurs et certains point modifiés et ce sont les suivants :

- **signal s in C** a été changé par **init s**, c'est-à-dire que le signal n'est plus initialisé pour une chaîne de contrôle précise mais pour l'entièreté de la machine durant son fonctionnement.
- **get** appliquera directement l'élément pris dans la liste de valeur à l'abstraction au lieu de le faire passivement en ajoutant dans la chaîne de contrôle.
- **Unit** n'est pas nécessaire car quand on convertit le langage ISWIM en langage SECD on vérifie la syntaxe du coup les applications sont forcément bon syntaxiquement.
- **get** retournera une erreur quand il aura fini de prendre dans la liste de valeurs.
- **throw erreur** sera propagé plutôt que vérifié directement.

Il a été demandé aussi de mettre un nom explicite pour chaque règles ainsi que des commentaires explicatifs.

Ces modifications ont été faite le lundi après-midi et mardi. Un mail a été envoyé pour avoir l'approbation des nouvelles règles avant de commencer à faire la preuve du déterminisme. Quelques améliorations ont été faites le jeudi et une relance par mail a été faite.

Après une entre-vue avec Mme Bousdira, j'ai débuté la preuve par induction le vendredi, grâce celle-ci j'ai pu me rendre compte du manque de la règle **Lever erreur** que j'ai rajouté . Cependant ma première preuve me parait sommaire donc j'attends la vérification de mes professeurs la semaine prochaine.

9.3.3 Semaine du 13 au 17

9.3.4 Semaine du 20 au 24