Rapport de stage
Développement d'un noyau de programmation synchrone

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

25 Mars 2019

Table des matières

1 Remerciement

Avant tout développement sur mon sujet de stage, j'aimerai remercié 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 sureté 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 lié à mes recherches "préliminaires" qui ne sont pas lié directement à la programmation réactive. Elle regroupera l'article [2] et [3]. La second sera dédié à l'article [1] qui est lui complètement axé sur le 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...* 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 connaisons 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 $\boldsymbol{n} = \alpha \cup \beta \cup \eta$.

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 $X_3 \notin FV(M_1)\backslash X_1$
— $(M_1\ M_2)[X \leftarrow M_3]\ = (M_1[X \leftarrow M_3]\ M_2[X \leftarrow M_3])$

Simplification : Afin d'allèger l'écriture en enlevement des parenthèses, il y a des règles de prioritées qui sont les suivantes :

— Application associative à gauche : $M1\ M2\ M3\ = ((M1\ M2)M3)$
— Application prioritaire par rapport au abstraction : $\lambda X.M1\ M2\ = \lambda X.(M1\ M2)$
— Les abstractions consécutives peuvent être regroupé : $\lambda XYZ.M\ = (\lambda X.(\lambda Y.(\lambda Z.M)))$

Version écriture lourde	Version écriture légère
$((\lambda x.((\lambda z.z)\ x))\ \underline{(\lambda x.x)})$	$\lambda x.(\lambda z.z)\ x)\ \underline{\lambda x.x}$
$\rightarrow^{\alpha}_{n}\ \underline{((\lambda x.((\lambda z.z)\ x))}\ (\lambda y.y))$	$\rightarrow^{\alpha}_{n}\ \underline{\lambda x.(\lambda z.z)\ x)}\ \lambda y.y$
$\rightarrow^{\eta}_{n}\ \underline{((\lambda z.z)\ (\lambda y.y))}$	$\rightarrow^{\eta}_{n}\ \underline{(\lambda z.z)\ \lambda y.y}$
$\rightarrow^{\beta}_{n}\ (\lambda y.y)$	$\rightarrow^{\beta}_{n}\ \lambda y.y$

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

Théorème de la forme normale : Si on peut réduire L tels que $L =_n M$ et $L =_n N$ et que N et M sont en forme 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 $M \twoheadrightarrow n_n L'$ et $N \twoheadrightarrow n_n L'$.

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 :

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

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

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.

3.1.3 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 à fortement influencé les autres langages qui l'ont suivi, principalement dans la programmation focntionnelle.

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

$| \ (o^n\ M\ ...\ N)$ 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 :

— $b[X \longleftarrow M] = b$

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

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

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

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 renommera 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.

3.1.4 CC Machine

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

— la partie du λ-calcul que l'on traite

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

Elle utilise le language ISWIM.

Les règles définit pour cette machine sont les suivantes :

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

La machine peut s'arrêter dans 3 états différents :

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

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

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

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

3.1.5 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 en exploitant les information du contexte (**Context**).

Les règles qui définisse la machine SCC sont les suivantes :

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

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

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

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

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

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

3.1.6 CK Machine

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

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

κ = mt

| $\langle fun, V, \kappa\rangle$

| $\langle arg, N, \kappa\rangle$

| $\langle opd, \langle V, ..., V, o^n\rangle, \langle N, ...\rangle, \kappa\rangle$

Cette structure est nommé **la continuation**.

Les règles qui définisse la machine CK sont les suivantes :

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

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

$\longrightarrow$ on a une **constante b** tels que $\langle M, mt\rangle \twoheadrightarrow_{ck} \langle b, mt\rangle$;

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

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

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

3.1.7 CEK Machine

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

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

On a alors :

ε = une fonction $\{\langle X, c\rangle, ...\}$ c = $\{\langle M, \varepsilon\rangle \mid FV(M) \subset dom(\varepsilon)\}$ v = $\{\langle V, \varepsilon\rangle \mid \langle V, \varepsilon\rangle \in c\}$

$\varepsilon[X \leftarrow c] = \{\langle X, c\rangle\} \cup \{\langle Y, c'\rangle \mid \langle Y, c'\rangle \in \varepsilon \text{ et } Y \neq X\}$

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

$\overline{\kappa}$ = mt

$\mid \langle fun, v, \overline{\kappa}\rangle$

$\mid \langle arg, c, \overline{\kappa}\rangle$

$\mid \langle opd, \langle v, ..., v, o^n\rangle, \langle c, ...\rangle, \overline{\kappa}\rangle$

Les règles qui définisse la machine CEK sont les suivantes :

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

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

$\longrightarrow$ on a une **constante b** tels que $\langle\langle M, \emptyset\rangle, mt\rangle \twoheadrightarrow_{cek} \langle\langle b, \varepsilon\rangle, mt\rangle$;

$\longrightarrow$ on a une **abstraction function** tels que $\langle\langle M, \emptyset\rangle, mt\rangle \twoheadrightarrow_{cek} \langle\langle \lambda X.N, \varepsilon\rangle, mt\rangle$;

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

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

3.1.8 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 $\widehat{D}$ pour laisser un espace de travail. Par contre pour la machine CEK, le contexte est créé quant on évalue une application ou un argument indépendamment de la complexité de celui-ci.

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

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

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

Les règles qui définisse la machine SECD sont les suivantes :

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

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

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

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

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

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

3.2 Le cas du Réactive ML

Modèle de programmation $\longrightarrow$ concurrence coopérative

L'analyse est découpé en 2 sous analyse :

— **statique** : système de type et d'effet

— **réactive** : détecter les erreurs de concurrence

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

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

Points fort :

— *implémentation séquentielle efficace*

— *pas de problème de parallélisme*

Points faible :

— *responsabilité de la réactivité au dev*

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

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

Exemple

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

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

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

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

4 Langage fonctionnel réactif

Le but est de réutilisé la une des machines étudié et de la rendre réactive. Les 1ère machines sont trop simple pour être utilisé ce qui nous reste la machine CEK et la machine SECD. Cependant il nous reste une contrainte qui va réussir à nous départager. En effet, il nous faut une machine qui fait un appel par valeur, c'est-à-dire que les paramètres seront évalués avant la fonction. La machine SECD remplit ce critère donc c'est celle que nous utiliserons comme base.

Mes encadrants m'ont donné des ajouts à faire par petites parties afin de structurer mon avancer. Je vais donc redonner les ajouts pour chaque partie ainsi que l'explication de leurs implantations.

4.1 Les bases de la réactivité de la machine

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'incrémente 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

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

$SSI(i)$ une fonction qui retourne le couple lié à un signal et un thread dans les signaux partagés.
Exemple : $SSI(i) = \langle CI, IL\rangle$

$\gamma(\langle CI, IL\rangle)$ 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

$S = \emptyset$
| $V\ S$

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

$C = \epsilon$

$\mid b\ C$	(une constante)
$\mid X\ C$	(une variable)
$\mid signal\ C$	(un signal)
$\mid \langle X, C'\rangle\ C$	(une abstraction)
$\mid ap\ C$	(une application)
$\mid prim_{o^n}\ C$	(un opérateur)
$\mid spawn\ C$	(créateur d'un nouveau thread)
$\mid present\ C$	(le test de présence d'un signal)
$\mid init\ C$	(initialise un signal pour une chaîne de contrôle donné)
$\mid put\ C$	(insère une valeur dans la liste de valeurs d'un signal)
$\mid get\ C$	(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 :** $\forall\ ssi \in SSI \mid ssi = \langle I, \langle CI, IL\rangle\rangle$ 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.

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

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

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

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 vérifier **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 pour $SI(s) = \langle emit, CS, SSI\rangle$ on a $SSI(b) = \langle CI, IL\rangle$ avec $I \notin IL$ alors $\gamma(SSI(b)) = V$

à vérifier **Prendre une valeur partagée (impossible) :** On prends dans la liste de valeurs d'un signal partagé lié à un identifant une constante via la fonction γ mais si on a déjà tout pris on retourne la valeur par défaut

$\langle\langle I, s\ b\ n\ \langle\langle X, C'\rangle, E'\rangle\ S, E, get\ C, D\rangle TL, SI, IP\rangle \longrightarrow_{secdv5} \langle\langle I, \emptyset, E'[X \leftarrow n], C', \langle S, E, C, D\rangle\rangle, TL, SI\rangle$
si pour $SI(s) = \langle emit, CS, SSI\rangle$ on a $SSI(b) = \langle CI, IL\rangle$ avec $I \in IL$ alors on prends n la valeur par défaut

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

$\langle\langle I, S, E, init\ C, D\rangle, TL, SI\rangle \longrightarrow_{secdv4} \langle\langle I, signal\ S, E, C, D\rangle, TL, SI'\rangle$ 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.

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

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.

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

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.

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

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**

5 Conclusion

6 Annexes

6.1 Les Exemples des machines étudiées

6.1.1 Exemple de fonctionnement de la machine CC

Voici un exemple de fonctionnement de la machine CC :

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

6.1.2 Exemple de fonctionnement de la machine SCC

Voici un exemple de fonctionnement de la machine SCC :

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

6.1.3 Exemple de fonctionnement de la machine CK

Voici un exemple de fonctionnement de la machine CK :

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.1.4 Exemple de fonctionnement de la machine CEK

Voici un exemple de fonctionnement de la machine CEK :

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.1.5 Exemple de fonctionnement de la machine SECD

Voici un exemple de fonctionnement de la machine SECD :

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.2 Les différentes versions faite pour rendre la machine SECD concurrente

6.2.1 1ère version des règles de la machine SECD Concurrente

Cette version ajoute les prémisse de la concurrence dans la machine SECD avec la possibilité de créér des threads d'initialiser des signaux,les émettre où encore de tester la présence d'un signal. Cette version est un condensé de 2 versions.

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

$\widehat{S} = \epsilon$
| $\widehat{V}\ \widehat{S}$
| $\langle C', C'' \rangle\ \widehat{S}$
| Remp $\widehat{S}$

$\widehat{\varepsilon}$ = une fonction $\{\langle X, \widehat{V} \rangle, ...\}$

$\widehat{C} = \epsilon$
| b $\widehat{C}$
| X $\widehat{C}$
| ap $\widehat{C}$
| $prim_{o^n}\ \widehat{C}$
| $\langle X, \widehat{C} \rangle\ \widehat{C}$
| bspawn $\widehat{C}$
| espawn $\widehat{C}$
| $\langle s, C', C'' \rangle\ \widehat{C}$
| $\langle s, \widehat{C'} \rangle\ \widehat{C}$
| $emit_s\ \widehat{C}$

$\widehat{D} = \epsilon$
| $\langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D} \rangle$

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

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

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

Les nouvelles règles sont les suivantes :

Partie de base de la machine SECD

Constante : On a une constante, on la déplace dans la pile.
$\langle \widehat{S}, \widehat{\varepsilon}, b\ \widehat{C}, \widehat{D}, \widehat{W}, \widehat{ST}, \widehat{SI} \rangle \longmapsto_{secdv1} \langle b\ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D}, \widehat{W}, \widehat{ST}, \widehat{SI} \rangle$

Substitution : On a une variable, on prends la substitution dans l'environnement et on la mets dans la pile.
$\langle \widehat{S}, \widehat{\varepsilon}, X\ \widehat{C}, \widehat{D}, \widehat{W}, \widehat{ST}, \widehat{SI} \rangle \longmapsto_{secdv1} \langle \widehat{V}\ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D}, \widehat{W}, \widehat{ST}, \widehat{SI} \rangle$ où $\widehat{V} = \varepsilon(X)$

Opération : On a un opérateur et le nombre de constante nécessaire dans la pile, via la fonction δ et in retourne le résultat dans la pile.
$\langle b_1\ ...\ b_n \widehat{S}, \widehat{\varepsilon}, prim_{o^n}\ \widehat{C}, \widehat{D}, \widehat{W}, \widehat{ST}, \widehat{SI} \rangle \longmapsto_{secdv1} \langle \widehat{V}\ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D}, \widehat{W}, \widehat{ST}, \widehat{SI} \rangle$ où $\widehat{V} = \delta(o^n, b_1, ...b_n)$

Abstraction : On a une abstraction, on crée une fermeture comportant l'abstraction et l'environnement courant et on mets la fermeture dans la pile
$\langle \widehat{S}, \widehat{\varepsilon}, \langle X, C' \rangle\ \widehat{C}, \widehat{D}, \widehat{W}, \widehat{ST}, \widehat{SI} \rangle \longmapsto_{secdv1} \langle \langle \langle X, C' \rangle, \varepsilon \rangle\ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D}, \widehat{W}, \widehat{ST}, \widehat{SI} \rangle$

Application : On a une application, donc on dauvegarde dans le dépôt, on ajoute une substitution et on remplace la chaîne de contrôle et l'environnement par ceux présent dans la fermeture.
$\langle \widehat{V}\ \langle \langle X, C' \rangle, \varepsilon' \rangle\ \widehat{S}, \widehat{\varepsilon}, ap\ \widehat{C}, \widehat{D}, \widehat{W}, \widehat{ST}, \widehat{SI} \rangle \longmapsto_{secdv1} \langle \epsilon, \varepsilon'[X \leftarrow \widehat{V}], C', \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D} \rangle, \widehat{W}, \widehat{ST}, \widehat{SI} \rangle$

Récupération de sauvegarde : On a rien mais le dépôt comporte une sauvegarde donc on prends celle-ci.
$\langle \widehat{V}\ \widehat{S}, \widehat{\varepsilon}, \emptyset, \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D} \rangle, \widehat{W}, \widehat{ST}, \widehat{SI} \rangle \longmapsto_{secdv1} \langle \widehat{V}\ \widehat{S'}, \widehat{\varepsilon'}, \widehat{C'}, \widehat{D}, \widehat{W}, \widehat{ST}, \widehat{SI} \rangle$

Partie pour la concurrence

Création thread : On veut créer un nouveau thread
$\langle \widehat{S}, \widehat{\varepsilon}, bspawn\ \widehat{C'}\ espawn\ \widehat{C}, \widehat{D}, \widehat{W}, \widehat{ST}, \widehat{SI} \rangle \longmapsto_{secdv1} \langle \text{ Remp } \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D}, \widehat{W} \langle \widehat{S}, \widehat{E}, \widehat{C'}, \widehat{D} \rangle, \widehat{ST}, \widehat{SI} \rangle$

Récupération dans la file d'attente : On a plus rien à traiter et on a aucune sauvegarde, du coup on change de thread courant par le thread en tête de la file d'attente.
$\langle \widehat{S}, \widehat{\varepsilon}, \emptyset, \emptyset, \langle \widehat{S'}, \widehat{\varepsilon'}, \widehat{C}, \widehat{D} \rangle\ \widehat{W}, \widehat{ST}, \widehat{SI} \rangle \longmapsto_{secdv1} \langle \widehat{S'}, \widehat{\varepsilon'}, \widehat{C}, \widehat{D}, \widehat{W}, \widehat{ST}, \widehat{SI} \rangle$

Fin d'instant logique : On a plus rien à traiter et on a plus rien dans la file d'attente. C'est la fin de l'instant logique
$\langle \widehat{S}, \widehat{\varepsilon}, \emptyset, \emptyset, \emptyset, \widehat{ST}, \widehat{SI} \rangle \longmapsto_{secdv1} \langle \widehat{S}, \widehat{\varepsilon}, \emptyset, \emptyset, \widehat{W}, \emptyset, \emptyset, \emptyset \rangle$
avec $\widehat{W}$ = tout les éléments de $\widehat{ST}$ qui prennent leurs 2nd choix

Émettre : on émet un signal
$\langle \widehat{S}, \widehat{\varepsilon}, emit_s\ \widehat{C}, \widehat{D}, \widehat{W}, \widehat{ST}, \widehat{SI} \rangle \longmapsto_{secdv1} \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D}, \widehat{W'}, \widehat{ST'}, \widehat{SI} \rangle$
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

Présence d'un signal : On teste la présence d'un signal et il l'est donc on prends la 1ère option.
$\langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle\ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \longmapsto_{secdv1} \langle \widehat{S}, \widehat{\varepsilon}, C'\ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle$
si s $\in \widehat{SI}$ et s $\in \widehat{\varepsilon}$

Thread bloqué remplacé : On teste la présence d'un signal et il ne l'est pas donc on le remplace par le thread en tête de la file d'attente.
$\langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle\ \widehat{C}, \langle \widehat{S'}, \widehat{\varepsilon'}, C''', \widehat{D'} \rangle \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \longmapsto_{secdv1} \langle \widehat{S'}, \widehat{\varepsilon'}, \widehat{C'''}, \widehat{W}, \widehat{ST}\ \langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle\ \widehat{C}, \widehat{D} \rangle, \widehat{SI}, \widehat{D'} \rangle$
si s $\notin \widehat{SI}$ et s $\in \widehat{\varepsilon}$

Thread bloqué non remplacé : On teste la présence d'un signal et il ne l'est pas et la file est vide mest juste le thread courant dans la liste de threads bloqués.
$\langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle\ \widehat{C}, \emptyset, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \longmapsto_{secdv1} \langle \epsilon, \emptyset, \emptyset, \emptyset, \widehat{ST}\ \langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle\ \widehat{C}, \widehat{D} \rangle, \widehat{SI}, \emptyset \rangle$
si s $\notin \widehat{SI}$ et s $\in \widehat{\varepsilon}$

Initialisation signal : On initialise le signal
$\langle \widehat{S}, \widehat{\varepsilon}, \langle s, C' \rangle\ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \longmapsto_{secdv1} \langle \epsilon, \widehat{\varepsilon}[init \leftarrow s], C', \widehat{W}, \widehat{ST}, \widehat{SI}, \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D} \rangle \rangle$

Partie commune

Application neutre droite : on a une application avec un neutre dans la pile donc on l'enlève
$\langle \widehat{V} \text{ Remp } \widehat{S}, \widehat{\varepsilon}, ap\ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \longmapsto_{secdv1} \langle \widehat{V}\ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle$

Application neutre gauche : on a une application avec un neutre dans la pile donc on l'enlève
$\langle \text{Remp } \widehat{V}\ \widehat{S}, \widehat{\varepsilon}, ap\ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle \longmapsto_{secdv1} \langle \widehat{V}\ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle$

la machine SECD version 1 peut s'arrêter dans 4 états différents :

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

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

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

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

6.2.2 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 :***

$\widehat{S} = ...$
$\widehat{\varepsilon} = ...$
$\widehat{C} = ...$
$\quad | \ throw_e \ \widehat{C}$
$\quad | \ \langle e, \langle \widehat{C'}, \langle X, \widehat{C''} \rangle \rangle \rangle \ \widehat{C}$
$\widehat{W} = ...$
$\widehat{ST} = ...$
$\widehat{SI} = ...$
$\widehat{D} = ...$
$\widehat{H} = \epsilon$
$\quad | \ \langle e, \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \rangle$
$\widehat{V} = ...$
$\quad | \ erreur_e$

Les nouvelles règles sont les suivantes :

1. $\langle \widehat{S}, \widehat{\varepsilon}, b \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \longmapsto_{secdv3} \langle b \ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$

2. $\langle \widehat{S}, \widehat{\varepsilon}, X \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \longmapsto_{secdv3} \langle \widehat{V} \ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$
   où $\widehat{V} = \varepsilon(X)$

3. $\langle b_1 \ ... \ b_n \widehat{S}, \widehat{\varepsilon}, prim_{o^n} \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \longmapsto_{secdv3} \langle \widehat{V} \ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$
   où $\widehat{V} = \delta(o^n, b_1, ...b_n)$

4. $\langle \widehat{S}, \widehat{\varepsilon}, throw_e \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \longmapsto_{secdv3} \langle erreur_e \ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$

5. $\langle \widehat{S}, \widehat{\varepsilon}, \langle X, C' \rangle \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \longmapsto_{secdv3} \langle \langle \langle X, C' \rangle, \varepsilon \rangle \ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$

6. $\langle \widehat{V} \ \langle \langle X, C' \rangle, \varepsilon' \rangle \ \widehat{S}, \widehat{\varepsilon}, ap \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \longmapsto_{secdv3} \langle \epsilon, \varepsilon'[X \leftarrow \widehat{V}], C', \widehat{W}, \widehat{ST}, \widehat{SI}, \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D} \rangle, \widehat{H} \rangle$

7. $\langle \widehat{V} \ \widehat{S}, \widehat{\varepsilon}, \emptyset, \langle \widehat{S'}, \widehat{\varepsilon'}, \widehat{C'}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D} \rangle, \widehat{H} \rangle \longmapsto_{secdv3} \langle \widehat{V} \ \widehat{S'}, \widehat{\varepsilon'}, \widehat{C'}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$

8. $\langle \widehat{V} \text{ Remp } \widehat{S}, \widehat{\varepsilon}, ap \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \longmapsto_{secdv3} \langle \widehat{V} \ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$

9. $\langle \text{ Remp } \widehat{V} \widehat{S}, \widehat{\varepsilon}, ap \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \longmapsto_{secdv3} \langle \widehat{V} \ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$

10. $\langle \text{ Remp } \widehat{V} \widehat{S}, \widehat{\varepsilon}, ap \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \longmapsto_{secdv3} \langle \widehat{V} \ \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$

11. $\langle \widehat{S}, \widehat{\varepsilon}, bspawn \ \widehat{C'} \ espawn \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \longmapsto_{secdv3} \langle \text{ Remp } \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W} \langle \widehat{S}, \widehat{E}, \widehat{C'}, \widehat{D} \rangle, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$

12. $\langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \longmapsto_{secdv3} \langle \widehat{S}, \widehat{\varepsilon}, C' \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$
    si s $\in \widehat{SI}$ et s $\in \widehat{\varepsilon}$

13. $\langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle \ \widehat{C}, \langle \widehat{S'}, \widehat{\varepsilon'}, C''', \widehat{D'} \rangle \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \longmapsto_{secdv3} \langle \widehat{S'}, \widehat{\varepsilon'}, \widehat{C'''}, \widehat{W}, \widehat{ST} \ \langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle \ \widehat{C}, \widehat{D} \rangle, \widehat{SI}, \widehat{D'}, \widehat{H} \rangle$
    si s $\notin \widehat{SI}$ et s $\in \widehat{\varepsilon}$

14. $\langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle \ \widehat{C}, \emptyset, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \longmapsto_{secdv3} \langle \epsilon, \emptyset, \emptyset, \emptyset, \widehat{ST} \ \langle \widehat{S}, \widehat{\varepsilon}, \langle s, C', C'' \rangle \ \widehat{C}, \widehat{D} \rangle, \widehat{SI}, \emptyset, \widehat{H} \rangle$
si s $\notin \widehat{SI}$ et s $\in \widehat{\varepsilon}$

15. $\langle \widehat{S}, \widehat{\varepsilon}, \langle s, C' \rangle \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \longmapsto_{secdv3} \langle \epsilon, \widehat{\varepsilon}[init \leftarrow s], C', \widehat{W}, \widehat{ST}, \widehat{SI}, \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{D} \rangle, \widehat{H} \rangle$

16. $\langle \widehat{S}, \widehat{\varepsilon}, \emptyset, \langle \widehat{S'}, \widehat{\varepsilon'}, \widehat{C'}, \widehat{D'} \rangle \ \widehat{W}, \widehat{ST}, \widehat{SI}, \emptyset, \widehat{H} \rangle \longmapsto_{secdv3} \langle \widehat{S'}, \widehat{\varepsilon'}, \widehat{C'}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D'}, \widehat{H} \rangle$

17. $\langle \widehat{S}, \widehat{\varepsilon}, \emptyset, \emptyset, \widehat{ST}, \widehat{SI}, \emptyset, \widehat{H} \rangle \longmapsto_{secdv3} \langle \widehat{S}, \widehat{\varepsilon}, \emptyset, \widehat{W}, \emptyset, \emptyset, \emptyset, \widehat{H} \rangle$
avec $\widehat{W}$ = tout les éléments de $\widehat{ST}$ qui prennent leurs 2nd choix

18. $\langle \widehat{S}, \widehat{\varepsilon}, emit_s \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \longmapsto_{secdv3} \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W'}, \widehat{ST'}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle$
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. $\langle \widehat{S}, \widehat{\varepsilon}, \langle e, \langle \widehat{C'}, \langle X, \widehat{C''} \rangle \rangle \rangle \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \longmapsto_{secdv3} \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C'} \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \langle e, \langle \widehat{S}, \widehat{\varepsilon}, \langle X, \widehat{C''} \rangle \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \rangle \rangle$

20. $\langle \widehat{S}, \widehat{\varepsilon}, throw_e \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \langle e, \langle \widehat{S'}, \widehat{\varepsilon'}, \langle X, \widehat{C''} \rangle \ \widehat{C'}, \widehat{W'}, \widehat{ST'}, \widehat{SI'}, \widehat{D'}, \widehat{H} \rangle \rangle \rangle$
$\longmapsto_{secdv3} \langle \widehat{S'}, \widehat{\varepsilon'}, \langle X, \widehat{C''} \rangle \ throw_e \ \widehat{C}, \widehat{W'}, \widehat{ST'}, \widehat{SI'}, \widehat{D'}, \widehat{H} \rangle$

21. $\langle \widehat{S}, \widehat{\varepsilon}, throw_e \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \langle e', \langle \widehat{S'}, \widehat{\varepsilon'}, \langle X, \widehat{C''} \rangle \ \widehat{C'}, \widehat{W'}, \widehat{ST'}, \widehat{SI'}, \widehat{D'}, \widehat{H} \rangle \rangle \rangle$
$\longmapsto_{secdv3}$ on regarde dans $\widehat{H}$ récursivement si il y a un gestionnaire pour l'erreur e

22. $\langle \widehat{S}, \widehat{\varepsilon}, throw_e \ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \emptyset \rangle \longmapsto_{secdv3} \langle \widehat{S}, \widehat{\varepsilon}, throw_e, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$

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

$\longrightarrow$ on a une **constante b** tels que $\langle \epsilon, \emptyset, [M]_{secdv3}, \emptyset, \emptyset, \emptyset, \epsilon, \emptyset \rangle \twoheadrightarrow_{secdv3} \langle b, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon, \widehat{H} \rangle$;

$\longrightarrow$ on a une **abstraction function** tels que $\langle \epsilon, \emptyset, [M]_{secdv3}, \emptyset, \emptyset, \emptyset, \epsilon, \emptyset \rangle \twoheadrightarrow_{secdv3} \langle \langle \langle X, \widehat{C} \rangle, \widehat{\varepsilon'} \rangle, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon, \widehat{H} \rangle$;

$\longrightarrow$ on a un **remplacement** tels que $\langle \epsilon, \emptyset, [M]_{secdv3}, \emptyset, \emptyset, \emptyset, \epsilon, \emptyset \rangle \twoheadrightarrow_{secdv3} \langle Remp, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon, \widehat{H} \rangle$;

$\longrightarrow$ on a une **erreur e** tels que $\langle \epsilon, \emptyset, [M]_{secdv3}, \emptyset, \emptyset, \emptyset, \epsilon, \emptyset \rangle \twoheadrightarrow_{secdv3} \langle erreur_e, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon, \widehat{H} \rangle$;

6.2.3 4ème version des règles 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 : $\rho(\{..., \langle s, \{..., \langle id, valeur\rangle, ...\}, emit\rangle, ...\}, v, s, id) = \{..., \langle s, \{..., \langle id, valeur\ v\rangle, ...\}, emit\rangle, ...\}$

$\gamma(l, s, i, i')$ = la fonction qui pour une liste de valeurs partagées classés par signals et par thread l, un signal s, l'identifiant du thread courant et l'identifiant du thread auquels on veut accédé, 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 :
$\gamma(\{..., \langle s, \{..., \langle id, \{..., \langle b, \{..., id', ...\}\rangle, \langle n, \{...\}\rangle, ...\}, \{...\}\rangle, ...\}\rangle, ...\}, s, id, id') =$
$\langle b, \{..., \langle s, \{..., \langle id, \{..., \langle b, \{...\}\rangle, \langle n, \{..., id'\}\rangle, ...\}, \{...\}\rangle, ...\}\rangle, ...\}\rangle$

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

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

$\iota(l, s, i)$ = 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 $\langle s, \{..., \langle id, \{..., b, ...\}\rangle, ...\}, true\rangle$, on ajoute x dans SSI
$\forall\ x \in CS$ tels que $\langle s, \{..., \langle id, \{..., b, ...\}\rangle, ...\}, emit\rangle$ on le remplace par $\langle s, \{..., \langle id, \{\}\rangle, ...\}, faux\rangle$

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 $= \langle CS, SSI\rangle$

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

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

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

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.

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

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

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.

$\langle I, b_n, ..., b_1\ S, E, prim_{o^n}\ C, TL, SI, D, H, IP\rangle \longrightarrow_{secdv4} \langle I, V\ S, E, C, TL, SI, D, H, IP\rangle$
avec $\delta(o^n\ b_1 ... b_n) = V$

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.

$\langle I, throw\ erreur_e\ S, E, prim_{o^{1_e}}\ C, TL, SI, D, H, IP\rangle \longrightarrow_{secdv4} \langle I, V\ S, E, C, TL, SI, D, H, IP\rangle$
avec $\delta(o^{1_e}\ erreur_e) = V$

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 = un élément de $C \setminus prim_{o^{1_e}}$

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.

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

Partie pour la concurrence

Création thread : On veut créer un nouveau thread.

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

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

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

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

$\langle I, s\ b\ S, E, put\ C, TL, \langle CS, SSI\rangle, D, H, IP\rangle \longrightarrow_{secdv4} \langle I, S, E, C, TL, \langle CS', SSI\rangle, D, H, IP\rangle$
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 identifant une constante via la fonction γ .

$\langle I, s\ b\ \langle\langle X, C'\rangle, E'\rangle\ S, E, get\ C, TL, \langle CS, SSI\rangle, D, H, IP\rangle$
$\longrightarrow_{secdv4} \langle I, \emptyset, E'[X \leftarrow V], C', TL, \langle CS, SSI'\rangle, \langle S, E, C, D\rangle, H, IP\rangle$
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 identifant une constante via la fonction γ mais on a déjà tout pris donc on lève une erreur.

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

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

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

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

$\langle I, s\ S, E, \langle C', C''\rangle\ C, \langle\emptyset, ST\rangle, \langle CS, SSI\rangle, D, H, IP\rangle$
$\longrightarrow_{secdv4} \langle IP, \emptyset, \epsilon, \emptyset, \langle W, ST\langle s, \langle I, s\ S, E, \langle C', C''\rangle\ C, D\rangle\rangle\rangle, \langle CS, SSI\rangle, \emptyset, H, IP + 1\rangle$
avec $\beta(CS, s) = faux$

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

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

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

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.

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

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

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

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

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

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

7 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*