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

3 Journal de bord

3.1 Mars

3.1.1 Semaine 25 au 29 Mars

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

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

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

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

2 semaines m'ont été accordé pour réaliser ces objectifs dans le langage de programmation *Ocaml*. J'ai eu, pour cela, un article traitant des λ-calculs[3] afin de m'aider dans ma démarche.
Il a été décidé que tous les programmes seront partagé sur un github qui a pour adresse :
https ://github.com/JordanIschard/StageL3.
Le reste de la journée a été utilisé pour commencer ce dit article.

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

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

3.2 Avril

3.2.1 Semaine du 1 au 5 Avril

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

$$(\lambda \mathrm{X}_1.\mathrm{M}_1)[\mathrm{X}_2 \leftarrow \mathrm{M}_2] = (\lambda \mathrm{X}_3.\mathrm{M}_1[\mathrm{X}_1 \leftarrow \mathrm{X}_3][\mathrm{X}_2 \leftarrow \mathrm{M}_2])$$
$$\text{où } \mathrm{X}_1 \neq \mathrm{X}_2,\ \mathrm{X}_3 \notin \mathrm{FV}(\mathrm{M}_2) \text{ et } \mathrm{X}_3 \notin \mathrm{FV}(\mathrm{M}_1) \backslash \mathrm{X}_1$$

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

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

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

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

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

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

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

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

3.2.2 Semaine du 8 au 12

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

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

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

1. Implémenter une concurrence dans la machine SECD avec les notions suivantes :
   (a) **spawn t** : lance un nouveau thread et le mets en attente
   (b) **present s in t1 t2** : si le signal s est présent prends t1 sinon attend le prochain temps logique et prends t2
   (c) **emit s** : émet le signal s
   (d) **signal s in t** : initialise le signal s pour t
2. Implémenter une gestion d'erreur dans la machine SECD

Ces 2 objectifs doivent être atteint en 3 semaines maximum.

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

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

— ceux qui attendent

— ceux qui sont bloquées

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

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

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

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

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

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

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

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

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

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

3.2.3 Semaine du 15 au 19

La 1ère version des règles de la nouvelle machine SECD fût terminé le lundi et vous pouvez trouvez ces dites règles en Annexe. Un ajout a été fait en plus dans la 1ère version, quand on tombe sur un élément comme **spawn** ou **emit** il faut continuer de faire fonctionner la machine. Donc une constante **Remp** a été ajouté dans l'optique de ne pas casser le fonctionnement de la machine SECD de base.
Cette version est inutilisable car elle part dans l'optique que tous les signaux sont déjà initialisé.

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

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

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

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

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

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

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

Pour cela il faut :

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

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

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

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

— une **liste de valeurs** liés à un signal et un thread précis : Cela implique de modifier la liste des signaux pour ajouter une liste de valeur et un identifiant de thread liés.

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

  $\widehat{SI} = \{\langle id, \{\langle signal, \{valeur, ...\}, init, emit\rangle\}\rangle, ...\}$
  avec id = un identifiant de thread
  avec init = un booléen indiquant si il est initialisé ou non
  avec emit = un booléen indiquant si il est émit ou non

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

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

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

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

3.2.4 Semaine du 22 au 26

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

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

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

— trier par threads puis par signaux est moins pratique que trier par signaux puis par threads. Du coup la structure de $\widehat{CS}$ sera $\{\langle signal, \{\langle id, \{valeur, ...\}, init\rangle\}, emit\rangle, ...\}$.

— quand on créé un nouveau thread, j'avais omis de dupliqué les signaux initialisé plus tôt pour le nouveau thread créé dans la liste des signaux. Ceci a été fixé.

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

3.3 Mai

3.3.1 Semaine du 29 Avril au 3

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

1. Faire une preuve prouvant le déterminisme de la machine

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

> **Définition du déterminisme d'un automate** : Un automate fini déterministe, parfois abrégé en AFD (en anglais deterministic finite automaton, abrégé en DFA) est un automate fini dont les transitions à partir de chaque état sont déterminées de façon unique par le symbole d'entrée. [5]
>
> **Définition du déterminisme d'une machine via un algorithme** : Un algorithme déterministe est un algorithme qui, étant donné une entrée particulière, produira toujours la même sortie, avec la machine sous-jacente passant toujours par la même séquence d'états.
>
> Formellement, un algorithme déterministe calcule une fonction mathématique ; une fonction ayant une valeur unique pour n'importe quelle entrée dans son ensemble de définition, l'algorithme produit cette valeur en sortie. [6]

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

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

3.3.2 Semaine du 6 au 10

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

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

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

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

3.3.3 Semaine du 13 au 17

3.3.4 Semaine du 20 au 24

4 Recherche d'Informations

4.1 Le Réactive ML

Modèle de programmation ⟶ 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∗ ⟶ *chaque processus va régulièrement "laisser la main aux autres"*

Ordonnancement préemptif ⟶ *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**.

4.2 Les λ-calculs

4.2.1 Les règles de β-réduction

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

4.2.2 Les règles de réduction générale

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

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

4.2.3 Les règles de priorité

- Application associative à gauche : M1 M2 M3 = ((M1 M2)M3)
- Application prioritaire par rapport au abstraction : λX.M1 M2 = λX.(M1 M2)
- Les abstractions consécutives peuvent être regroupé : λXYZ.M = (λX.(λY.(λZ.M)))

4.2.4 Comment savoir si on a une forme normal ?

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

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

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

Certaines lambda calcul expression n'a pas de forme normal comme : ((λx.x x) (λx.x x).
D'autres en ont une mais on peut rentrer dans une boucle infini de réduction si on choisi la mauvaise réduction.

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

- M $\longrightarrow_{\bar{n}}$ N if M β N
- M $\longrightarrow_{\bar{n}}$ N if M η N
- (λX.M) $\longrightarrow_{\bar{n}}$ (λX.N)
- (M N) $\longrightarrow_{\bar{n}}$ (M' N) if M $\longrightarrow_{\bar{n}}$ M'
  et ∀L, (M N) β L impossible et (M N) η L impossible
- (M N) $\longrightarrow_{\bar{n}}$ (M N') if N $\longrightarrow_{\bar{n}}$ N'
  et M est une forme normale
  et ∀L, (M N) β L impossible et (M N) η L impossible

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

4.3 ISWIM

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

M,N,L,K =

| X (les variables)

| (λX.M)

| (M M)

| b (les constantes b)

| (o^n M ... M) avec o^n les fonctions primitives

V,U,W =

| b

| X

| $(\lambda X.M)$

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

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

- b[X $\longleftarrow$ M] = b
- (o^n M$_1$ ... M$_n$)[X $\longleftarrow$ M] = (o^n M$_1$[X $\longleftarrow$ M] ... M$_n$[X $\longleftarrow$ M])

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

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

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

4.4 Les différentes machines traitées

4.4.1 CC Machine

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

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

Elle utilise le language ISWIM.

Les règles définit pour cette machine sont les suivantes :

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

La machine peut s'arrêter dans 3 états différents :

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

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

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

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

4.4.2 SCC Machine

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

Les règles qui définisse la machine SCC sont les suivantes :

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

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

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

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

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

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

4.4.3 CK Machine

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

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

$\kappa = \text{mt}$
$\mid \langle fun, V, \kappa\rangle$
$\mid \langle arg, N, \kappa\rangle$
$\mid \langle opd, \langle V, ..., V, o^n\rangle, \langle N, ...\rangle, \kappa\rangle$

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.

4.4.4 CEK Machine

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

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

On a alors :

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

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

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

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.

4.4.5 SECD Machine

La différence entre la machine CEK et SECD est la façon dont le contexte est sauvegarder pendant que les sous-expressions sont évaluées.
En effet, dans la machine SECD le contexte est créer par un appel de fonction, quand toute est stocké dans $\widehat{D}$ 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 $\{\langle X, \widehat{V}\rangle, ...\}$

$\widehat{C} = \epsilon \mid \text{b}\ \widehat{C} \mid \text{X}\ \widehat{C} \mid \text{ap}\ \widehat{C} \mid prim_{o^n}\ \widehat{C} \mid \langle X, \widehat{C}\rangle\ \widehat{C}$

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

$\widehat{V} = \text{b} \mid \langle\langle X, \widehat{C}\rangle, \widehat{\varepsilon}\rangle$

$[b]_{secd} = \text{b}$

$[X]_{secd} = \text{X}$

$[(M_1\ M_2)]_{secd} = [M_1]_{secd}\ [M_2]_{secd}$ 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.

5 Dernière version de la machine SECD concurrente

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

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

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

$\gamma(l, s, i, i')$ = 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 :
$\varepsilon(\{..., \langle s, \{...\}, faux\rangle, ...\}, s) = \{..., \langle s, \{...\}, vraie\rangle, ...\}$

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

$V = b$
$\mid \langle\langle X, C'\rangle E\rangle$
$\mid erreur_e$

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

$S = \emptyset$
$\mid VS$
$\mid signal\ S$
$\mid throw\ S$

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

$C = \epsilon$
$\mid b\ C$ (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)
$\mid bspawn\ C$ (début d'un nouveau thread)
$\mid espawn\ C$ (fin d'un nouveau thread)
$\mid \langle C', C''\rangle\ C$ (le test de présence d'un signal)
$\mid emit\ C$ (émet 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)
$\mid erreur_e\ C$ (une erreur)
$\mid throw\ C$ (lève une erreur)
$\mid \langle C', \langle X, C''\rangle\rangle\ C$ (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 $= \emptyset$
$\mid \langle S, E, C, D\rangle$ (une sauvegarde liée à une abstraction)

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

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 patagé

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
SI' = $\langle CS', SSI'\rangle$ et SI = $\langle CS, SSI\rangle$

CS' = CS avec les listes de valeurs remise à zéro ainsi que le booléen représentant l'émission est mis à faux

SSI' = SSI avec tous les signaux émit à l'instant qui vient de finir et leurs listes de valeurs associé

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 $\langle 0, \emptyset, \emptyset, [M]_{secdv4}, \langle \emptyset, \emptyset\rangle, \langle \emptyset, \emptyset\rangle, \emptyset, \emptyset, \emptyset, 1\rangle$
$\twoheadrightarrow_{secdv4} \langle I, erreur_e\ S, E, \epsilon, \langle \emptyset, \emptyset\rangle, SI, \emptyset, H, IP\rangle$;

6 Conclusion

7 Annexes

7.1 Les Exemples des machines étudiées

7.1.1 Exemple de fonctionnement de la machine CC

Voici un exemple de fonctionnement de la machine CC :

CC machine : $\langle (((\lambda f.\lambda x.f\ x)\ \lambda y.(+\ y\ y))\ \ulcorner 1 \urcorner), [] \rangle$
$> \langle (M\ N), E \rangle \longmapsto_{cc} \langle M, E[([]\ N)] \rangle\ siM \notin V$
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 : $\langle \ulcorner 2 \urcorner, [] \rangle$

7.1.2 Exemple de fonctionnement de la machine SCC

Voici un exemple de fonctionnement de la machine SCC :

SCC machine : $\langle (((\lambda f.\lambda x.f\ x)\ \lambda y.(+\ y\ y))\ \ulcorner 1 \urcorner), [] \rangle$
$> \langle (M\ N), E \rangle \longmapsto_{scc} \langle M, E[([]\ N)] \rangle$
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 : $\langle \ulcorner 2 \urcorner, [] \rangle$

7.1.3 Exemple de fonctionnement de la machine CK

Voici un exemple de fonctionnement de la machine CK :

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

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

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

7.1.4 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 : $\langle\langle\ulcorner 2\urcorner,\emptyset\rangle,mt\rangle$

7.1.5 Exemple de fonctionnement de la machine SECD

Voici un exemple de fonctionnement de la machine SECD :

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

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

7.2 Les différentes versions faite pour rendre la machine SECD concurrente

7.2.1 1ère version des règles de la machine SECD Concurrente

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

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

$\widehat{\varepsilon} = ...$

$\widehat{C} = ...$

$\mid \text{bspawn}\ \widehat{C}$

$\mid \text{espawn}\ \widehat{C}$

$\mid \langle C', C'' \rangle\ \widehat{C}$

$\mid present_s\ \widehat{C}$

$\mid emit_s\ \widehat{C}$

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

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

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

$\widehat{D} = ...$

Les nouvelles règles sont les suivantes :

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

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

$\longrightarrow$ 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**.

7.2.2 2ème version des règles de la machine SECD Concurrente

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

$\widehat{S} = ...$

$\quad |$ ~~$\langle C', C'' \rangle\ \widehat{S}$~~

$\widehat{\varepsilon} = ...$

$\quad |$ s (un signal initialisé)

$\widehat{C} = ...$

$\quad |$ [$\langle C', C'' \rangle\ \widehat{C}$

$\quad |$ $present_s\ \widehat{C}$] changé en $\langle s, C', C'' \rangle\ \widehat{C}$

$\quad |$ $\langle s, \widehat{C'} \rangle\ \widehat{C}$

$\widehat{W} = ...$

$\widehat{ST} = ...$

$\widehat{SI} = ...$

$\widehat{D} = ...$

Les nouvelles règles sont les suivantes :

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

la machine SECD version 2 peut s'arrêter dans 4 états différents :

$\longrightarrow$ on a une **constante b** tels que $\langle \epsilon, \emptyset, [M]_{secdv2}, \emptyset, \emptyset, \emptyset, \epsilon \rangle \twoheadrightarrow_{secdv2} \langle b, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon \rangle$;

$\longrightarrow$ on a une **abstraction function** tels que $\langle \epsilon, \emptyset, [M]_{secdv2}, \emptyset, \emptyset, \emptyset, \epsilon \rangle \twoheadrightarrow_{secdv2} \langle \langle \langle X, \widehat{C} \rangle, \widehat{\varepsilon'} \rangle, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon \rangle$;

$\longrightarrow$ on a un **remplacement** tels que $\langle \epsilon, \emptyset, [M]_{secdv2}, \emptyset, \emptyset, \emptyset, \epsilon \rangle \twoheadrightarrow_{secdv2} \langle Remp, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon \rangle$;

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

7.2.3 3ème version des règles de la machine SECD Concurrente

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

$\widehat{S} = ...$

$\widehat{\varepsilon} = ...$

$\widehat{C} = ... \mid throw_e\ \widehat{C} \mid \langle e, \langle \widehat{C'}, \langle X, \widehat{C''} \rangle \rangle \rangle\ \widehat{C}$

$\widehat{W} = ...$

$\widehat{ST} = ...$

$\widehat{SI} = ...$

$\widehat{D} = ...$

$\widehat{H} = \epsilon \mid \langle e, \langle \widehat{S}, \widehat{\varepsilon}, \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \rangle$

$\widehat{V} = ... \mid erreur_e$

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. $\langle \widehat{V}$ 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. $\langle$ 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. $\langle$ 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. $\langle \widehat{S}, \widehat{\varepsilon}, bspawn\ \widehat{C'}\ espawn\ \widehat{C}, \widehat{W}, \widehat{ST}, \widehat{SI}, \widehat{D}, \widehat{H} \rangle \longmapsto_{secdv3} \langle$ 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 $\langle \epsilon, \emptyset, [M]_{secdv3}, \emptyset, \emptyset, \emptyset, \epsilon, \emptyset \rangle \twoheadrightarrow_{secdv3} \langle erreur_e, \widehat{\varepsilon}, \epsilon, \emptyset, \emptyset, \widehat{SI}, \epsilon, \widehat{H} \rangle$;

8 Bibliographie

[1] *Réactivité des systèmes coopératifs : le cas Réactive ML* de Louis Mandrel et Cédric Pasteur

[2] *The ZINC experiment : an economical implementation of the ML language* de Xavier Leroy

[3] *Programming Languages And Lambda Calculi* de Mathias Felleisen et Matthew Flatt

[4] *https ://www.irif.fr/ carton/Enseignement/Complexite/MasterInfo/Cours/turing.html*

[5] *https ://fr.wikipedia.org/wiki/Automate_fini_ déterministe*

[6] *https ://fr.wikipedia.org/wiki/Algorithme_ déterministe*