# A verified abstract machine for functional coroutines – Coq formalization (companion technical report)

by Tristan Crolard

CNAM Paris, France

*Email:* `tristan.crolard@cnam.fr`

**Abstract**

Functional coroutines are a restricted form of control mechanism, where each continuation comes with its local environment. This restriction was originally obtained by considering a constructive version of Parigot's classical natural deduction which is sound and complete for the Constant Domain logic. In this article, we present a refinement of de Groote's abstract machine which is proved to be correct for functional coroutines. Therefore, this abstract machine also provides a direct computational interpretation of the Constant Domain logic.

This companion technical report contains the full Coq formalization (all the proofs are available in the Coq source file). The section numbering is the same as in the article for easy cross referencing.

## 1 Introduction

## 2 Dependency relations

## 3 Abstract machines

**Require Import** *Utf8*.
**Add LoadPath** "~/Sources/Coq/Tactics".
**Require Import** *CpdtTactics LibTactics TwelfTactics*.

**Require Import** *Arith List String*.
**Require Import** *Coq.extraction.ExtrOcamlString*.

**Definition** *ble_nat* ($n$ : *nat*) ($m$ : *nat*) : *bool* := *beq_nat* ($n - m$) 0.
**Notation** "$n == m$" := (*beq_nat n m*) (at level 70, no associativity).
**Notation** "$n \leqq m$" := (*ble_nat n m*) (at level 70, no associativity).

**Reserved Notation** "$m_1 - m_2 = m$" (at level 30).

**Inductive** *minus* : *nat* → *nat* → *nat* → *Prop* :=

| $Minus_1$ : ∀ $m_1$,

$$m_1 - 0 = m_1$$

| $Minus_2$ : ∀ $m_1$ $m_2$ $m$,

$$\frac{m_1 - m_2 = m}{S(m_1) - S(m_2) = m}$$

**where** "$m_1 - m_2 = m$" := (*minus* $m_1$ $m_2$ $m$).

**Fixpoint** $f$ ($m$ : *nat*) : ($m - m = 0$) :=
 **match** $m$ **with**
 | 0 => $Minus_1$ 0
 | $S(n)$ => $Minus_2$ $n$ $n$ 0 ($f$ $n$)
 **end**.

**Lemma** *minus_trivial* : ∀ $n$, *not* ($0 = Sn$).

**Lemma** *minus_id* : ∀ $m$, $m - m = 0$.

**Hint Resolve** *minus_id*.

**Lemma** *minus_unique* : ∀ $m_2$ $m_1$ $m$ $m'$, $m_1 - m_2 = m \to m_1 - m_2 = m' \to m = m'$.

**Hint Resolve** *minus_unique* : *gen_subst_db*.

**Lemma** *minus_succ* : ∀ $n_3$ $n_2$ $n_1$, $n_1 - (Sn_2) = n_3 \to n_1 - n_2 = (Sn_3)$.

**Lemma** *minus_swap* : ∀ $n_3$ $n_2$ $n_1$, $n_1 - n_2 = n_3 \to n_1 - n_3 = n_2$.

**Definition** *vector* := *list nat*.
**Definition** *table* := *list vector*.

**Reserved Notation** "$\mathcal{I}(n) = m$" (at level 30).

**Inductive** *fetch* $\{A : Type\}$ : *list* $A \to nat \to$ A $\to Prop$ :=

| $Fetch_1$ : ∀ $\mathcal{I}$ ($n$:$A$),

$$(n{::}\mathcal{I})(0) = n$$

| $Fetch_2$ : ∀ $\mathcal{I}$ ($n$:$A$) $n_1$ $n_2$,

$$\frac{\mathcal{I}(n_1) = n_2}{(n{::}\mathcal{I})(Sn_1) = n_2}$$

**where** "$\mathcal{I}(n) = m$" := (@*fetch* _ $\mathcal{I}$ $n$ $m$).

**Lemma** *fetch_unique* : ∀ $A$ $n$ $\mathcal{I}$ ($y$:$A$) ($y'$:$A$), $\mathcal{I}(n) = y \to \mathcal{I}(n) = y' \to y = y'$.

**Hint Resolve** *fetch_unique* : *gen_subst_db*.

**Reserved Notation** "$n - \mathcal{I}\ (\ l\ ) = g$" (at level 30).

**Inductive** *compute* : *list nat* $\to$ *nat* $\to$ *nat* $\to$ *nat* $\to$ *Prop* :=

| $Compute_1$ : ∀ $\mathcal{I}$ $n$ $l$ $g$ $k$,

$$\frac{\mathcal{I}(l) = k \qquad n - k = g}{n - \mathcal{I}(l) = g}$$

**where** "$n - \mathcal{I}\ (\ l\ ) = g$" := (*compute* $\mathcal{I}$ $n$ $l$ $g$).

**Lemma** *compute_unique* : ∀ $\mathcal{I}$ $n$ $l$ $y$ $y'$, $n - \mathcal{I}(l) = y \to n - \mathcal{I}(l) = y' \to y = y'$.

**Hint Resolve** *compute_unique* : *gen_subst_db*.

**Reserved Notation** "$n \in \mathscr{I}$" (at level 30).

**Inductive** *member* : *list nat* → *nat* → *Prop* :=

| $Member_1$ : $\forall\ \mathscr{I}\ n$,

$$n \in (n :: \mathscr{I})$$

| $Member_2$ : $\forall\ \mathscr{I}\ n\ n'$,

$$\frac{n \in \mathscr{I}}{n \in (n' :: \mathscr{I})}$$

**where** "$n \in \mathscr{I}$" := (*member* $\mathscr{I}$ *n*).

**Lemma** *domain* : $\forall\ \mathscr{I}\ k,\ k \in \mathscr{I} \rightarrow \exists n, \mathscr{I}(n) = k$.

## 3.1 Safe $\lambda_{ct}$-terms

**Inductive** *term* : *Type* :=
| *Var* (*n* : *nat*)
| *App* ($t_1$ : *term*) ($t_2$ : *term*)
| *Lambda* (*t* : *term*)
| *Catch* (*t* : *term*)
| *Throw* ($\alpha$ : *nat*) (*t* : *term*).

**Notation** "'\`' *x* '\`'" := (*Var x*) (at level 80).
**Notation** "$t_1$ " $t_2$" := (*App* $t_1$ $t_2$) (at level 80).
**Notation** "'λ' *t*" := (*Lambda t*) (at level 80).

**Notation** "'**catch**'" := *Catch* (at level 80).
**Notation** "'**throw**'" := *Throw* (at level 80).
**Notation** "'**get-context**'" := *Catch* (at level 80).
**Notation** "'**set-context**'" := *Throw* (at level 80).

**Reserved Notation** "'*Safe_*' '(' *n* ')^(' $\mathscr{I}$ , $\mathscr{I}_\mu$ ')' ( *t* )" (at level 30).

**Inductive** *safe* : *term* → *list nat* → *list* (*list nat*) → *nat* → *Prop* :=

| $Safe_1$ : $\forall\ \mathscr{I}\ \mathscr{I}_\mu\ n\ g\ k$,

$$\frac{n - g = k \qquad k \in \mathscr{I}}{Safe_n^{\mathscr{I},\mathscr{I}_\mu}(\text{`}g\text{`})}$$

| $Safe_2$ : $\forall\ \mathscr{I}\ \mathscr{I}_\mu\ n\ t\ u$,

$$\frac{Safe_n^{\mathscr{I},\mathscr{I}_\mu}(t) \qquad Safe_n^{\mathscr{I},\mathscr{I}_\mu}(u)}{Safe_n^{\mathscr{I},\mathscr{I}_\mu}(t\,u)}$$

| $Safe_3$ : $\forall\ \mathscr{I}\ \mathscr{I}_\mu\ n\ t$,

$$\frac{Safe_{Sn}^{(Sn::\mathscr{I}),\mathscr{I}_\mu}(t)}{Safe_n^{\mathscr{I},\mathscr{I}_\mu}(\lambda t)}$$

| $Safe_4$ : $\forall\ \mathscr{I}\ \mathscr{I}_\mu\ n\ t$,

$$\frac{Safe_n^{\mathscr{I},(\mathscr{I}::\mathscr{I}_\mu)}(t)}{Safe_n^{\mathscr{I},\mathscr{I}_\mu}(\textbf{catch}\ t)}$$

| $Safe_5 : \forall\ \mathcal{I}\ \mathcal{I}'\ \mathcal{I}_\mu\ n\ t\ \alpha,$

$$\frac{\mathcal{I}_\mu(\alpha)=\mathcal{I}' \qquad Safe_n^{\mathcal{I}',\mathcal{I}_\mu}(t)}{Safe_n^{\mathcal{I},\mathcal{I}_\mu}(\textbf{throw}\ \alpha\ t)}$$

**where** "'*Safe_*' '(' *n* ')^(' $\mathcal{I}$ , $\mathcal{I}_\mu$ ')' ( *t* )" := (*safe t* $\mathcal{I}$ $\mathcal{I}_\mu$ *n*).

## 3.2 From local indices to global indices

**Reserved Notation** "'↓_' '(' *n* ')^(' $\mathcal{I}$ , $\mathcal{I}\mu$ ')' ( $t_1$ ) = $t_2$" (at level 30).

**Inductive** *litogi* : *term* → *list nat* → *list* (*list nat*) → *nat* → *term* → *Prop* :=

| $Litogi_1 : \forall\ \mathcal{I}\ \mathcal{I}_\mu\ n\ g\ l,$

$$\frac{n-\mathcal{I}(l)=g}{\downarrow_n^{\mathcal{I},\mathcal{I}_\mu}(`l`)=(`g`)}$$

| $Litogi_2 : \forall\ \mathcal{I}\ \mathcal{I}_\mu\ n\ t\ u\ t'\ u',$

$$\frac{\downarrow_n^{\mathcal{I},\mathcal{I}_\mu}(t)=t' \qquad \downarrow_n^{\mathcal{I},\mathcal{I}_\mu}(u)=u'}{\downarrow_n^{\mathcal{I},\mathcal{I}_\mu}(t\,u)=(t'u')}$$

| $Litogi_3 : \forall\ \mathcal{I}\ \mathcal{I}_\mu\ n\ t\ t',$

$$\frac{\downarrow_{Sn}^{(Sn::\mathcal{I}),\mathcal{I}_\mu}(t)=t'}{\downarrow_n^{\mathcal{I},\mathcal{I}_\mu}(\lambda t)=(\lambda t')}$$

| $Litogi_4 : \forall\ \mathcal{I}\ \mathcal{I}_\mu\ n\ t\ t',$

$$\frac{\downarrow_n^{\mathcal{I},(\mathcal{I}::\mathcal{I}_\mu)}(t)=t'}{\downarrow_n^{\mathcal{I},\mathcal{I}_\mu}(\textbf{get-context}\ t)=(\textbf{catch}\ t')}$$

| $Litogi_5 : \forall\ \mathcal{I}\ \mathcal{I}'\ \mathcal{I}_\mu\ n\ t\ t'\ \alpha,$

$$\frac{\mathcal{I}_\mu(\alpha)=\mathcal{I}' \qquad \downarrow_n^{\mathcal{I}',\mathcal{I}_\mu}(t)=t'}{\downarrow_n^{\mathcal{I},\mathcal{I}_\mu}(\textbf{set-context}\ \alpha\ t)=(\textbf{throw}\ \alpha\ t')}$$

**where** "'↓_' '(' *n* ')^(' $\mathcal{I}$ , $\mathcal{I}_\mu$ ')' ( $t_1$ ) = $t_2$" := (*litogi* $t_1$ $\mathcal{I}$ $\mathcal{I}_\mu$ *n* $t_2$).

**Lemma** *litogi_unique* : $\forall\ \mathcal{I}\ \mathcal{I}_\mu\ n\ x\ y\ y',\ \downarrow_n^{\mathcal{I},\mathcal{I}_\mu}(x)=y \rightarrow \downarrow_n^{\mathcal{I},\mathcal{I}_\mu}(x)=y' \rightarrow y=y'.$

**Hint Resolve** *litogi_unique* : *gen_subst_db*.

**Lemma** *safe_image* : $\forall\ \mathcal{I}\ \mathcal{I}_\mu\ n\ t',\ Safe_n^{\mathcal{I},\mathcal{I}_\mu}(t')\rightarrow \exists t, \downarrow_n^{\mathcal{I},\mathcal{I}_\mu}(t)=t'.$

## 3.3 Abstract machine for $\lambda_{ct}$-terms

### 3.3.1 Closure, environment, stack and state

**Inductive** *clos* :=
| *Cl* (*t* : *term*) ($\mathcal{E}$ : *list clos*) ($\mathcal{E}_\mu$ : *list* (*list clos*)).

**Definition** *stack* := *list clos*.
**Definition** *c_env* := *list clos*.
**Definition** *k_env* := *list stack*.

**Inductive** *state* :=

| *St* ($t$ : *term*) ($\mathscr{E}$ : *c_env*) ($\mathscr{E}_\mu$ : *k_env*) ($\mathcal{S}$ : *stack*).

**Notation** "$\langle\ t\ ,\ \mathscr{E}\ ,\ \mathscr{E}_\mu\ ,\ \mathcal{S}\ \rangle$" := (*St* $t$ $\mathscr{E}$ $\mathscr{E}_\mu$ $\mathcal{S}$) (at level 80).
**Notation** "[ $t$ , $\mathscr{E}$ , $\mathscr{E}_\mu$ ]" := (*Cl* $t$ $\mathscr{E}$ $\mathscr{E}_\mu$) (at level 80).

### 3.3.2 Evaluation rules

**Reserved Notation** "$\sigma_1 \rightsquigarrow \sigma_2$" (at level 30).

**Inductive** *step* : *state* → *state* → *Prop* :=

| *K_var* : ∀ $k\ t\ \mathscr{E}\ \mathscr{E}'\ \mathscr{E}_\mu\ \mathscr{E}'_\mu\ \mathcal{S}$,

$$\frac{\mathscr{E}(k) = [t, \mathscr{E}', \mathscr{E}'_\mu]}{\langle\ `k`, \mathscr{E}, \mathscr{E}_\mu, \mathcal{S}\rangle \rightsquigarrow \langle t, \mathscr{E}', \mathscr{E}'_\mu, \mathcal{S}\rangle}$$

| *K_app* : ∀ $t\ u\ \mathscr{E}\ \mathscr{E}_\mu\ \mathcal{S}$,

$$\langle (t\,u), \mathscr{E}, \mathscr{E}_\mu, \mathcal{S}\rangle \rightsquigarrow \langle t, \mathscr{E}, \mathscr{E}_\mu, [u, \mathscr{E}, \mathscr{E}_\mu] :: \mathcal{S}\rangle$$

| *K_abs* : ∀ $t\ c\ \mathscr{E}\ \mathscr{E}_\mu\ \mathcal{S}$,

$$\langle \lambda t, \mathscr{E}, \mathscr{E}_\mu, c :: \mathcal{S}\rangle \rightsquigarrow \langle t, (c :: \mathscr{E}), \mathscr{E}_\mu, \mathcal{S}\rangle$$

| *K_catch* : ∀ $t\ \mathscr{E}\ \mathscr{E}_\mu\ \mathcal{S}$,

$$\langle \mathbf{catch}\ t, \mathscr{E}, \mathscr{E}_\mu, \mathcal{S}\rangle \rightsquigarrow \langle t, \mathscr{E}, (\mathcal{S} :: \mathscr{E}_\mu), \mathcal{S}\rangle$$

| *K_throw* : ∀ $t\ \alpha\ \mathscr{E}\ \mathscr{E}_\mu\ \mathcal{S}\ \mathcal{S}'$,

$$\frac{\mathscr{E}_\mu(\alpha) = \mathcal{S}'}{\langle \mathbf{throw}\ \alpha\ t, \mathscr{E}, \mathscr{E}_\mu, \mathcal{S}\rangle \rightsquigarrow \langle t, \mathscr{E}, \mathscr{E}_\mu, \mathcal{S}'\rangle}$$

**where** "$\sigma_1 \rightsquigarrow \sigma_2$" := (*step* $\sigma_1$ $\sigma_2$).

**Lemma** *step_unique* : ∀ $\sigma_1\ \sigma_2\ \sigma_2'$, $\sigma_1 \rightsquigarrow \sigma_2 \to \sigma_1 \rightsquigarrow \sigma_2' \to \sigma_2 = \sigma_2'$.

**Hint Resolve** *step_unique* : *gen_subst_db*.

## 3.4 Abstract machine for safe $\lambda_{ct}$-terms (with local environments)

### 3.4.1 Closure, environment, stack and state

**Inductive** *clos_l* :=
| *Cl_l* ($t$ : *term*) ($\mathscr{L}$ : *list clos_l*) ($\mathscr{L}_\mu$ : *list* (*list clos_l*)) ($\mathscr{E}_\mu$ : *list* (*list clos_l*)).

**Definition** *stack_l* := *list clos_l*.
**Definition** *l_env_l* := *list clos_l*.
**Definition** *m_env_l* := *list l_env_l*.
**Definition** *k_env_l* := *list stack_l*.

**Inductive** *state_l* :=
| *St_l* ($t$ : *term*) ($\mathscr{L}$ : *l_env_l*) ($\mathscr{L}_\mu$ : *m_env_l*) ($\mathscr{E}_\mu$ : *k_env_l*) ($\mathcal{S}$ : *stack_l*).

**Notation** "$\langle\ t\ ,\ \mathscr{L}\ ,\ \mathscr{L}_\mu\ ,\ \mathscr{E}_\mu\ ,\ \mathcal{S}\ \rangle$" := (*St_l* $t$ $\mathscr{L}$ $\mathscr{L}_\mu$ $\mathscr{E}_\mu$ $\mathcal{S}$) (at level 80).
**Notation** "[ $t$ , $\mathscr{L}$ , $\mathscr{L}_\mu$ , $\mathscr{E}_\mu$ ]" := (*Cl_l* $t$ $\mathscr{L}$ $\mathscr{L}_\mu$ $\mathscr{E}_\mu$) (at level 80).

### 3.4.2 Evaluation rules

**Reserved Notation** "$\sigma_1 \rightsquigarrow$ '^$l$' $\sigma_2$" (at level 30).

**Inductive** *step_l* : *state_l* → *state_l* → *Prop* :=

| *L_var* : ∀ $k\ t\ \mathscr{L}\ \mathscr{L}'\ \mathscr{L}_\mu\ \mathscr{L}'_\mu\ \mathscr{E}_\mu\ \mathscr{E}'_\mu\ \mathcal{S}$,

$$\frac{\mathscr{L}(k) = [t, \mathscr{L}', \mathscr{L}'_\mu, \mathscr{E}'_\mu]}{\langle `k`, \mathscr{L}, \mathscr{L}_\mu, \mathscr{E}_\mu, \mathcal{S}\rangle \rightsquigarrow^l \langle t, \mathscr{L}', \mathscr{L}'_\mu, \mathscr{E}'_\mu, \mathcal{S}\rangle}$$

| *L_app* : ∀ $t\ u\ \mathscr{L}\ \mathscr{L}_\mu\ \mathscr{E}_\mu\ \mathcal{S}$,

$$\langle (t\,u), \mathscr{L}, \mathscr{L}_\mu, \mathscr{E}_\mu, \mathcal{S}\rangle \rightsquigarrow^l \langle t, \mathscr{L}, \mathscr{L}_\mu, \mathscr{E}_\mu, [u, \mathscr{L}, \mathscr{L}_\mu, \mathscr{E}_\mu] :: \mathcal{S}\rangle$$

| *L_abs* : ∀ $t\ c\ \mathscr{L}\ \mathscr{L}_\mu\ \mathscr{E}_\mu\ S'$,

$$\langle \lambda t, \mathscr{L}, \mathscr{L}_\mu, \mathscr{E}_\mu, c :: S'\rangle \rightsquigarrow^l \langle t, (c :: \mathscr{L}), \mathscr{L}_\mu, \mathscr{E}_\mu, S'\rangle$$

| *L_catch* : ∀ $t\ \mathscr{L}\ \mathscr{L}_\mu\ \mathscr{E}_\mu\ \mathcal{S}$,

$$\langle \textbf{get-context}\ t, \mathscr{L}, \mathscr{L}_\mu, \mathscr{E}_\mu, \mathcal{S}\rangle \rightsquigarrow^l \langle t, \mathscr{L}, (\mathscr{L} :: \mathscr{L}_\mu), (\mathcal{S} :: \mathscr{E}_\mu), \mathcal{S}\rangle$$

| *L_throw* : ∀ $t\ \alpha\ \mathscr{L}\ \mathscr{L}'\ \mathscr{L}_\mu\ \mathscr{E}_\mu\ \mathcal{S}\ \mathcal{S}'$,

$$\frac{\mathscr{L}_\mu(\alpha) = \mathscr{L}' \qquad \mathscr{E}_\mu(\alpha) = \mathcal{S}'}{\langle \textbf{set-context}\ \alpha\ t, \mathscr{L}, \mathscr{L}_\mu, \mathscr{E}_\mu, \mathcal{S}\rangle \rightsquigarrow^l \langle t, \mathscr{L}', \mathscr{L}_\mu, \mathscr{E}_\mu, \mathcal{S}'\rangle}$$

**where** "$\sigma_1 \rightsquigarrow$ '^$l$' $\sigma_2$" := (*step_l* $\sigma_1\ \sigma_2$).

**Lemma** *step_l_unique* : ∀ $\sigma_1\ \sigma_2\ \sigma'_2$, $\sigma_1 \rightsquigarrow^l \sigma_2 \to \sigma_1 \rightsquigarrow^l \sigma'_2 \to \sigma_2 = \sigma'_2$.

**Hint Resolve** *step_l_unique* : *gen_subst_db*.

# 4 Bisimulations

## 4.1 Abstract machine for safe $\lambda_{\mathbf{ct}}$-terms (with indirection tables)

### 4.1.1 Closure, environment, stack and state

**Inductive** *clos_i* :=
| *Cl_i* ($t$ : *term*) ($n$ : *nat*) ($\mathscr{I}$ : *vector*) ($\mathscr{I}_\mu$ : *table*) ($\mathscr{E}$ : *list clos_i*) ($\mathscr{E}_\mu$ : *list* (*list clos_i*)).

**Definition** *stack_i* := *list clos_i*.
**Definition** *c_env_i* := *list clos_i*.
**Definition** *k_env_i* := *list stack_i*.

**Inductive** *state_i* :=
| *St_i* ($t$ : *term*) ($n$ : *nat*) ($\mathscr{I}$ : *vector*) ($\mathscr{I}_\mu$ : *table*) ($\mathscr{E}$ : *c_env_i*) ($\mathscr{E}_\mu$ : *k_env_i*) ($\mathcal{S}$ : *stack_i*).

**Notation** "$\langle\ t\ ,\ n\ ,\ \mathscr{I}\ ,\ \mathscr{I}_\mu\ ,\ \mathscr{E}\ ,\ \mathscr{E}_\mu\ ,\ \mathcal{S}\ \rangle$" := (*St_i* $t\ n\ \mathscr{I}\ \mathscr{I}_\mu\ \mathscr{E}\ \mathscr{E}_\mu\ \mathcal{S}$) (at level 80).
**Notation** "[ $t\ ,\ n\ ,\ \mathscr{I}\ ,\ \mathscr{I}_\mu\ ,\ \mathscr{E}\ ,\ \mathscr{E}_\mu$ ]" := (*Cl_i* $t\ n\ \mathscr{I}\ \mathscr{I}_\mu\ \mathscr{E}\ \mathscr{E}_\mu$) (at level 80).

### 4.1.2 Evaluation rules

**Reserved Notation** "$\sigma_1 \rightsquigarrow$ '^$i$' $\sigma_2$" (at level 30).

**Inductive** *step_i* : *state_i* → *state_i* → *Prop* :=

| *I_var* : $\forall\ n\ n'\ l\ g\ t\ \mathcal{I}\ \mathcal{I}'\ \mathcal{I}_\mu\ \mathcal{I}'_\mu\ \mathcal{E}\ \mathcal{E}'\ \mathcal{E}_\mu\ \mathcal{E}'_\mu\ \mathcal{S}$,

$$\frac{n - \mathcal{I}(l) = g \qquad \mathcal{E}(g) = [t, n', \mathcal{I}', \mathcal{I}'_\mu, \mathcal{E}', \mathcal{E}'_\mu]}{\langle `l`, n, \mathcal{I}, \mathcal{I}_\mu, \mathcal{E}, \mathcal{E}_\mu, \mathcal{S}\rangle \rightsquigarrow^i \langle t, n', \mathcal{I}', \mathcal{I}'_\mu, \mathcal{E}', \mathcal{E}'_\mu, \mathcal{S}\rangle}$$

| *I_app* : $\forall\ n\ t\ u\ \mathcal{I}\ \mathcal{I}_\mu\ \mathcal{E}\ \mathcal{E}_\mu\ \mathcal{S}$,

$$\langle (t\,u), n, \mathcal{I}, \mathcal{I}_\mu, \mathcal{E}, \mathcal{E}_\mu, \mathcal{S}\rangle \rightsquigarrow^i \langle t, n, \mathcal{I}, \mathcal{I}_\mu, \mathcal{E}, \mathcal{E}_\mu, [u, n, \mathcal{I}, \mathcal{I}_\mu, \mathcal{E}, \mathcal{E}_\mu] :: \mathcal{S}\rangle$$

| *I_abs* : $\forall\ n\ t\ c\ \mathcal{I}\ \mathcal{I}_\mu\ \mathcal{E}\ \mathcal{E}_\mu\ S'$,

$$\langle \lambda t, n, \mathcal{I}, \mathcal{I}_\mu, \mathcal{E}, \mathcal{E}_\mu, c :: S'\rangle \rightsquigarrow^i \langle t, (Sn), ((Sn) :: \mathcal{I}), \mathcal{I}_\mu, c :: \mathcal{E}, \mathcal{E}_\mu, S'\rangle$$

| *I_catch* : $\forall\ n\ t\ \mathcal{I}\ \mathcal{I}_\mu\ \mathcal{E}\ \mathcal{E}_\mu\ \mathcal{S}$,

$$\langle \textbf{get-context}\ t, n, \mathcal{I}, \mathcal{I}_\mu, \mathcal{E}, \mathcal{E}_\mu, \mathcal{S}\rangle \rightsquigarrow^i \langle t, n, \mathcal{I}, (\mathcal{I} :: \mathcal{I}_\mu), \mathcal{E}, (\mathcal{S} :: \mathcal{E}_\mu), \mathcal{S}\rangle$$

| *I_throw* : $\forall\ n\ t\ \alpha\ \mathcal{I}\ \mathcal{I}'\ \mathcal{I}_\mu\ \mathcal{E}\ \mathcal{E}_\mu\ \mathcal{S}\ \mathcal{S}'$,

$$\frac{\mathcal{I}_\mu(\alpha) = \mathcal{I}' \qquad \mathcal{E}_\mu(\alpha) = \mathcal{S}'}{\langle \textbf{set-context}\ \alpha\ t, n, \mathcal{I}, \mathcal{I}_\mu, \mathcal{E}, \mathcal{E}_\mu, \mathcal{S}\rangle \rightsquigarrow^i \langle t, n, \mathcal{I}', \mathcal{I}_\mu, \mathcal{E}, \mathcal{E}_\mu, \mathcal{S}'\rangle}$$

**where** "$\sigma_1 \rightsquigarrow$ '^$i$' $\sigma_2$" := (*step_i* $\sigma_1\ \sigma_2$).

**Lemma** *step_i_unique* : $\forall\ \sigma_1\ \sigma_2\ \sigma'_2,\ \sigma_1 \rightsquigarrow^i \sigma_2 \rightarrow \sigma_1 \rightsquigarrow^i \sigma'_2 \rightarrow \sigma_2 = \sigma'_2$.

**Hint Resolve** *step_i_unique* : *gen_subst_db*.

## 4.2 Lock-step simulation (−)

**Reserved Notation** "$c$ '^★=_$c$' $c'$" (at level 30).
**Reserved Notation** "$\mathcal{S}$ '^★=_$s$' $\mathcal{S}'$" (at level 30).
**Reserved Notation** "$\mathcal{E}$ '^★=_$e$' $\mathcal{E}'$" (at level 30).
**Reserved Notation** "$\mathcal{E}_\mu$ '^★=_$k$' $\mathcal{E}'_\mu$" (at level 30).

**Inductive** *translate_clos* : *clos_i* → *clos* → *Prop* :=
| *T_cl* : $\forall\ n\ t\ u\ \mathcal{I}\ \mathcal{I}_\mu\ \mathcal{E}\ \mathcal{E}'\ \mathcal{E}_\mu\ \mathcal{E}'_\mu$,

$$\frac{\downarrow_n^{\mathcal{I},\mathcal{I}_\mu}(t) = u \qquad \mathcal{E}^\star =_e \mathcal{E}' \qquad \mathcal{E}_\mu^\star =_k \mathcal{E}'_\mu}{[t, n, \mathcal{I}, \mathcal{I}_\mu, \mathcal{E}, \mathcal{E}_\mu]^\star =_c [u, \mathcal{E}', \mathcal{E}'_\mu]}$$

**where** "$c$ '^★=_$c$' $c'$" := (*translate_clos* $c\ c'$)

**with** *translate_stack* : *stack_i* → *stack* → *Prop* :=
| $T\_stack_1$ :

$$nil^\star =_s nil$$

| $T\_stack_2$ : $\forall\ c\ c'\ \mathcal{S}\ \mathcal{S}'$,

$$\frac{c^\star =_c c' \qquad \mathcal{S}^\star =_s \mathcal{S}'}{(c :: \mathcal{S})^\star =_s (c' :: \mathcal{S}')}$$

**where** "$\mathcal{S}$ '^★=_$s$' $\mathcal{S}'$" := (*translate_stack* $\mathcal{S}\ \mathcal{S}'$)

**with** *translate_c_env* : *c_env_i* → *c_env* → *Prop* :=

| $T\_c\_env_1$ :

$$nil^\star =_e nil$$

| $T\_c\_env_2$ : $\forall\ c\ c'\ \mathscr{E}\ \mathscr{E}'$,

$$\frac{c^\star =_c c' \qquad \mathscr{E}^\star =_e \mathscr{E}'}{(c :: \mathscr{E})^\star =_e (c' :: \mathscr{E}')}$$

**where** "$\mathscr{E}$ '^★=_e' $\mathscr{E}'$" := (*translate_c_env* $\mathscr{E}$ $\mathscr{E}'$)

**with** *translate_k_env* : *k_env_i* → *k_env* → *Prop* :=
| $T\_k\_env_1$ :

$$nil^\star =_k nil$$

| $T\_k\_env_2$ : $\forall\ \mathcal{S}\ \mathcal{S}'\ \mathscr{E}_\mu\ \mathscr{E}'_\mu$,

$$\frac{\mathcal{S}^\star =_s \mathcal{S}' \qquad \mathscr{E}_\mu^\star =_k \mathscr{E}'_\mu}{(\mathcal{S} :: \mathscr{E}_\mu)^\star =_k (\mathcal{S}' :: \mathscr{E}'_\mu)}$$

**where** "$\mathscr{E}_\mu$ '^★=_k' $\mathscr{E}'_\mu$" := (*translate_k_env* $\mathscr{E}_\mu$ $\mathscr{E}'_\mu$).

**Reserved Notation** "$\sigma$ '^★=_σ' $\sigma'$" (at level 30).

**Inductive** *translate_st* : *state_i* → *state* → *Prop* :=
| *T_st* : $\forall\ n\ t\ u\ \mathcal{I}\ \mathcal{I}_\mu\ \mathscr{E}\ \mathscr{E}'\ \mathscr{E}_\mu\ \mathscr{E}'_\mu\ \mathcal{S}\ \mathcal{S}'$,

$$\frac{[t, n, \mathcal{I}, \mathcal{I}_\mu, \mathscr{E}, \mathscr{E}_\mu]^\star =_c [u, \mathscr{E}', \mathscr{E}'_\mu] \qquad \mathcal{S}^\star =_s \mathcal{S}'}{\langle t, n, \mathcal{I}, \mathcal{I}_\mu, \mathscr{E}, \mathscr{E}_\mu, \mathcal{S}\rangle^\star =_\sigma \langle u, \mathscr{E}', \mathscr{E}'_\mu, \mathcal{S}'\rangle}$$

**where** "$\sigma$ '^★=_σ' $\sigma'$" := (*translate_st* $\sigma$ $\sigma'$).

**Scheme** *translate_clos_mut* := **Induction for** *translate_clos* **Sort** *Prop*
  **with** *translate_stack_mut* := **Induction for** *translate_stack* **Sort** *Prop*
  **with** *translate_c_env_mut* := **Induction for** *translate_c_env* **Sort** *Prop*
  **with** *translate_k_env_mut* := **Induction for** *translate_k_env* **Sort** *Prop*.

**Lemma** *translate_clos_unique* : $\forall\ c\ c_2\ c'_2,\ c^\star =_c c_2 \rightarrow c^\star =_c c'_2 \rightarrow c_2 = c'_2$
  **with** *translate_stack_unique* : $\forall\ s\ s_2\ s'_2,\ s^\star =_s s_2 \rightarrow s^\star =_s s'_2 \rightarrow s_2 = s'_2$
  **with** *translate_c_env_unique* : $\forall\ e\ e_2\ e'_2,\ e^\star =_e e_2 \rightarrow e^\star =_e e'_2 \rightarrow e_2 = e'_2$
  **with** *translate_k_env_unique* : $\forall\ k\ k_2\ k'_2,\ k^\star =_k k_2 \rightarrow k^\star =_k k'_2 \rightarrow k_2 = k'_2$.

**Hint Resolve** *translate_clos_unique* : *gen_subst_db*.
**Hint Resolve** *translate_stack_unique* : *gen_subst_db*.
**Hint Resolve** *translate_c_env_unique* : *gen_subst_db*.
**Hint Resolve** *translate_k_env_unique* : *gen_subst_db*.

**Lemma** *translate_st_unique* : $\forall\ \sigma\ \sigma_2\ \sigma'_2,\ \sigma^\star =_\sigma \sigma_2 \rightarrow \sigma^\star =_\sigma \sigma'_2 \rightarrow \sigma_2 = \sigma'_2$.

**Hint Resolve** *translate_st_unique* : *gen_subst_db*.

### 4.2.1 Soundness

**Lemma** *fetch_sound* : $\forall\ \mathscr{E}\ \mathscr{E}',\ \mathscr{E}^\star =_e \mathscr{E}' \rightarrow \forall\ n\ c,\ \mathscr{E}(n) = c \rightarrow \exists c', c^\star =_c c' \wedge \mathscr{E}'(n) = c'$.

**Lemma** *fetch_mu_sound* : $\forall\ n\ \mathcal{S}\ \mathscr{E}_\mu,\ \mathscr{E}_\mu(n) = \mathcal{S} \rightarrow \forall \mathscr{E}'_\mu, \mathscr{E}_\mu^\star =_k \mathscr{E}'_\mu \rightarrow \exists \mathcal{S}', \mathcal{S}^\star =_s \mathcal{S}' \wedge \mathscr{E}'_\mu(n) = \mathcal{S}'$.

**Theorem** *soundness* : $\forall\ \sigma_1\ \sigma_2\ \sigma'_1,\ \sigma_1 \rightsquigarrow^i \sigma_2\ \rightarrow\ \sigma_1^\star =_\sigma \sigma'_1\ \rightarrow\ \exists \sigma'_2,\ \sigma'_1 \rightsquigarrow \sigma'_2\ \wedge\ \sigma_2{}^\star =_\sigma \sigma'_2$.

#### 4.2.2 Completeness

**Lemma** *fetch_complete* : $\forall\ \mathscr{E}\ \mathscr{E}',\ \mathscr{E}^\star =_e \mathscr{E}' \to \forall\ n\ c',\ \mathscr{E}'(n) = c' \to \exists c, c^\star =_c c' \land \mathscr{E}(n) = c.$

**Lemma** *fetch_mu_complete* : $\forall\ n\ \mathcal{S}'\ \mathscr{E}'_\mu,\ \mathscr{E}'_\mu(n) = \mathcal{S}' \to \forall \mathscr{E}_\mu, \mathscr{E}_\mu^\star =_k \mathscr{E}'_\mu \to \exists \mathcal{S}, \mathcal{S}^\star =_s \mathcal{S}' \land \mathscr{E}_\mu(n) = \mathcal{S}.$

**Theorem** *completeness* : $\forall\ \sigma_1'\ \sigma_2'\ \sigma_1,\ \sigma_1' \rightsquigarrow \sigma_2' \ \to\ \sigma_1^\star =_\sigma \sigma_1' \ \to\ \exists \sigma_2,\ \sigma_1 \rightsquigarrow^i \sigma_2 \ \land\ \sigma_2^\star =_\sigma \sigma_2'.$

### 4.3 Lock-step simulation $(-)^\diamond$

**Reserved Notation** "$\mathscr{E}$ ( $n -\ k$ ) = $c$" (at level 30).

**Inductive** *compute_l* : *c_env_i* → *nat* → *nat* → *clos_i* → *Prop* :=

| *Compute_l*$_1$ : $\forall\ \mathscr{E}\ n\ g\ k\ c$,

$$\frac{n - k = g \qquad \mathscr{E}(g) = c}{\mathscr{E}(n-k) = c}$$

**where** "$\mathscr{E}$ ( $n -\ k$ ) = $c$" := (*compute_l* $\mathscr{E}$ *n k c*).

**Lemma** *compute_l_unique* : $\forall\ \mathscr{E}\ n\ k\ c\ c',\ \mathscr{E}(n-k) = c \to \mathscr{E}(n-k) = c' \to c = c'.$

**Hint Resolve** *compute_l_unique* : *gen_subst_db*.

**Reserved Notation** "$c$ '^⋄=_$c$' $c'$" (at level 30).
**Reserved Notation** "$\mathcal{S}$ '^⋄=_$s$' $\mathcal{S}'$" (at level 30).
**Reserved Notation** "$\mathscr{E}_\mu$ '^⋄=_$k$' $\mathscr{E}'_\mu$" (at level 30).

**Inductive** *flatten* : *nat* → *c_env_i* → *vector* → *l_env_l* →*Prop* :=
| *T_flatten*$_1$ : $\forall\ n\ \mathscr{E}$,

$$\textit{flatten}\ n\ \mathscr{E}\ \textit{nil}\ \textit{nil}$$

| *T_flatten*$_2$ : $\forall\ c\ c'\ n\ k\ \mathscr{E}\ \mathscr{I}\ \mathscr{L}$,

$$\frac{\mathscr{E}(n-k) = c \qquad c^\diamond =_c c' \qquad \textit{flatten}\ n\ \mathscr{E}\ \mathscr{I}\ \mathscr{L}}{\textit{flatten}\ n\ \mathscr{E}\ (k :: \mathscr{I})\ (c' :: \mathscr{L})}$$

**with** *map_flatten* : *nat* → *c_env_i* → *table* → *m_env_l* →*Prop* :=
| *T_map_flatten*$_1$ : $\forall\ n\ \mathscr{E}$,

$$\textit{map\_flatten}\ n\ \mathscr{E}\ \textit{nil}\,\textit{nil}$$

| *T_map_flatten*$_2$ : $\forall\ n\ \mathscr{E}\ \mathscr{I}\ \mathscr{L}\ \mathscr{I}_\mu\ \mathscr{L}_\mu$,

$$\frac{\textit{flatten}\ n\ \mathscr{E}\ \mathscr{I}\ \mathscr{L} \qquad \textit{map\_flatten}\ n\ \mathscr{E}\ \mathscr{I}_\mu\ \mathscr{L}_\mu}{\textit{map\_flatten}\ n\ \mathscr{E}\ (\mathscr{I} :: \mathscr{I}_\mu)\ (\mathscr{L} :: \mathscr{L}_\mu)}$$

**with** *translate_clos_l* : *clos_i* → *clos_l* → *Prop* :=
| *T_cl_l* : $\forall\ n\ t\ \mathscr{I}\ \mathscr{I}_\mu\ \mathscr{L}\ \mathscr{L}_\mu\ \mathscr{E}\ \mathscr{E}_\mu\ \mathscr{E}'_\mu$,

$$\frac{\textit{flatten}\ n\ \mathscr{E}\ \mathscr{I}\ \mathscr{L} \qquad \textit{map\_flatten}\ n\ \mathscr{E}\ \mathscr{I}_\mu\ \mathscr{L}_\mu \qquad \mathscr{E}_\mu^\diamond =_k \mathscr{E}'_\mu}{[t, n, \mathscr{I}, \mathscr{I}_\mu, \mathscr{E}, \mathscr{E}_\mu]^\diamond =_c [t, \mathscr{L}, \mathscr{L}_\mu, \mathscr{E}'_\mu]}$$

**where** "$c$ '^⋄=_$c$' $c'$" := (*translate_clos_l c c'*)

**with** *translate_stack_l* : *stack_i* → *stack_l* → *Prop* :=

| *T_stack_l*$_1$ :

$$nil^{\diamond} =_s nil$$

| *T_stack_l*$_2$ : $\forall\ c\ c'\ \mathcal{S}\ \mathcal{S}'$,

$$\frac{c^{\diamond} =_c c' \qquad \mathcal{S}^{\diamond} =_s \mathcal{S}'}{(c :: \mathcal{S})^{\diamond} =_s (c' :: \mathcal{S}')}$$

**where** "$\mathcal{S}$ '^⋄=_*s*' $\mathcal{S}'$" := (*translate_stack_l* $\mathcal{S}\ \mathcal{S}'$)

**with** *translate_k_env_l* : *k_env_i* → *k_env_l* → *Prop* :=
| *T_k_env_l*$_1$ :

$$nil^{\diamond} =_k nil$$

| *T_k_env_l*$_2$ : $\forall\ \mathcal{S}\ \mathcal{S}'\ \mathcal{E}_\mu\ \mathcal{E}'_\mu$,

$$\frac{\mathcal{S}^{\diamond} =_s \mathcal{S}' \qquad \mathcal{E}_\mu^{\diamond} =_k \mathcal{E}'_\mu}{(\mathcal{S} :: \mathcal{E}_\mu)^{\diamond} =_k (\mathcal{S}' :: \mathcal{E}'_\mu)}$$

**where** "$\mathcal{E}_\mu$ '^⋄=_*k*' $\mathcal{E}'_\mu$" := (*translate_k_env_l* $\mathcal{E}_\mu\ \mathcal{E}'_\mu$).

**Reserved Notation** "$\sigma$ '^⋄=_$\sigma$' $\sigma'$" (at level 30).

**Inductive** *translate_st_l* : *state_i* → *state_l* → *Prop* :=
| *T_st_l* : $\forall\ n\ t\ u\ \mathcal{I}\ \mathcal{I}_\mu\ \mathcal{E}\ \mathcal{L}\ \mathcal{L}_\mu\ \mathcal{E}_\mu\ \mathcal{E}'_\mu\ \mathcal{S}\ \mathcal{S}'$,

$$\frac{[t, n, \mathcal{I}, \mathcal{I}_\mu, \mathcal{E}, \mathcal{E}_\mu]^{\diamond} =_c [u, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}'_\mu] \qquad \mathcal{S}^{\diamond} =_s \mathcal{S}'}{\langle t, n, \mathcal{I}, \mathcal{I}_\mu, \mathcal{E}, \mathcal{E}_\mu, \mathcal{S}\rangle^{\diamond} =_\sigma \langle u, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}'_\mu, \mathcal{S}'\rangle}$$

**where** "$\sigma$ '^⋄=_$\sigma$' $\sigma'$" := (*translate_st_l* $\sigma\ \sigma'$).

**Scheme** *flatten_mut* := **Induction for** *flatten* **Sort** *Prop*
  **with** *map_flatten_mut* := **Induction for** *map_flatten* **Sort** *Prop*
  **with** *translate_clos_l_mut* := **Induction for** *translate_clos_l* **Sort** *Prop*
  **with** *translate_stack_l_mut* := **Induction for** *translate_stack_l* **Sort** *Prop*
  **with** *translate_k_env_l_mut* := **Induction for** *translate_k_env_l* **Sort** *Prop*.

**Lemma** *flatten_unique* : $\forall\ n\ \mathcal{E}\ \mathcal{I}\ \mathcal{L}\ \mathcal{L}', \mathit{flatten}\ n\ \mathcal{E}\ \mathcal{I}\ \mathcal{L} \to \mathit{flatten}\ n\ \mathcal{E}\ \mathcal{I}\ \mathcal{L}' \to \mathcal{L} = \mathcal{L}'$
  **with** *map_flatten_unique* : $\forall\ n\ \mathcal{E}\ \mathcal{I}_\mu\ \mathcal{L}_\mu\ \mathcal{L}'_\mu$,
    $\mathit{map\_flatten}\ n\ \mathcal{E}\ \mathcal{I}_\mu\ \mathcal{L}_\mu \to \mathit{map\_flatten}\ n\ \mathcal{E}\ \mathcal{I}_\mu\ \mathcal{L}'_\mu \to \mathcal{L}_\mu = \mathcal{L}'_\mu$
  **with** *translate_clos_l_unique* : $\forall\ c\ c_2\ c'_2,\ c^{\diamond} =_c c_2 \to c^{\diamond} =_c c'_2 \to c_2 = c'_2$
  **with** *translate_stack_l_unique* : $\forall\ s\ s_2\ s'_2,\ s^{\diamond} =_s s_2 \to s^{\diamond} =_s s'_2 \to s_2 = s'_2$
  **with** *translate_k_env_l_unique* : $\forall\ k\ k_2\ k'_2,\ k^{\diamond} =_k k_2 \to k^{\diamond} =_k k'_2 \to k_2 = k'_2$.

**Hint Resolve** *flatten_unique* : *gen_subst_db*.
**Hint Resolve** *map_flatten_unique* : *gen_subst_db*.
**Hint Resolve** *translate_clos_l_unique* : *gen_subst_db*.
**Hint Resolve** *translate_stack_l_unique* : *gen_subst_db*.
**Hint Resolve** *translate_k_env_l_unique* : *gen_subst_db*.

**Lemma** *translate_st_l_unique* : $\forall\ \sigma\ \sigma_2\ \sigma'_2,\ \sigma^{\diamond} =_\sigma \sigma_2 \to \sigma^{\diamond} =_\sigma \sigma'_2 \to \sigma_2 = \sigma'_2$.

**Hint Resolve** *translate_st_l_unique* : *gen_subst_db*.

### 4.3.1 Soundness

**Lemma** *minus_next* : $\forall\ n\ k\ g,\ n - k = g \to Sn - k = Sg$.

**Lemma** *weaken_flatten* : $\forall\ \{n\ \mathscr{E}\ \mathscr{I}\ \mathscr{L}\}, \forall\ c'$, *flatten* $n\ \mathscr{E}\ \mathscr{I}\ \mathscr{L} \rightarrow$ *flatten* $(Sn)\ (c' :: \mathscr{E})\ \mathscr{I}\ \mathscr{L}$.

**Lemma** *weaken_map_flatten* : $\forall\ \{n\ \mathscr{E}\ \mathscr{I}_\mu\ \mathscr{L}_\mu\}, \forall\ c'$,
*map_flatten* $n\ \mathscr{E}\ \mathscr{I}_\mu\ \mathscr{L}_\mu \rightarrow$ *map_flatten* $(Sn)\ (c' :: \mathscr{E})\ \mathscr{I}_\mu\ \mathscr{L}_\mu$.

**Lemma** *map_flatten_sound* : $\forall\ n\ \mathscr{E}\ \mathscr{I}'\ \mathscr{I}_\mu\ \alpha$,
$\mathscr{I}_\mu(\alpha) = \mathscr{I}' \rightarrow \forall\ \mathscr{L}_\mu$, *map_flatten* $n\ \mathscr{E}\ \mathscr{I}_\mu\ \mathscr{L}_\mu \rightarrow \exists\ \mathscr{L}'$, *flatten* $n\ \mathscr{E}\ \mathscr{I}'\ \mathscr{L}' \land\ \mathscr{L}_\mu(\alpha) = \mathscr{L}'$.

**Lemma** *fetch_sound_l* : $\forall\ n\ l\ k\ c\ \mathscr{I}$,
$\mathscr{I}(l) = k \rightarrow \forall \mathscr{E}, \mathscr{E}(n-k) = c \rightarrow \forall \mathscr{L}$, *flatten* $n\ \mathscr{E}\ \mathscr{I}\ \mathscr{L} \rightarrow \exists c', c^\diamond =_c c' \land \mathscr{L}(l) = c'$.

**Lemma** *fetch_mu_sound_l* : $\forall\ n\ \mathscr{S}\ \mathscr{E}_\mu, \mathscr{E}_\mu(n) = \mathscr{S} \rightarrow \forall \mathscr{E}'_\mu, \mathscr{E}^\diamond_\mu =_k \mathscr{E}'_\mu \rightarrow \exists \mathscr{S}', \mathscr{S}^\diamond =_s \mathscr{S}' \land \mathscr{E}'_\mu(n) = \mathscr{S}'$.

**Theorem** *soundness_l* : $\forall\ \sigma_1\ \sigma_2\ \sigma'_1, \sigma_1 \rightsquigarrow^i \sigma_2\ \rightarrow\ \sigma_1^\diamond =_\sigma \sigma'_1\ \rightarrow\ \exists \sigma'_2,\ \sigma'_1 \rightsquigarrow^l \sigma'_2\ \land\ \sigma_2{}^\diamond =_\sigma \sigma'_2$.

#### 4.3.2 Completeness

**Lemma** *map_flatten_complete* : $\forall\ \{n\ \mathscr{E}\ \mathscr{L}'\ \mathscr{L}_\mu\ \alpha\}$,
$\mathscr{L}_\mu(\alpha) = \mathscr{L}' \rightarrow \forall\ \mathscr{I}_\mu$, *map_flatten* $n\ \mathscr{E}\ \mathscr{I}_\mu\ \mathscr{L}_\mu \rightarrow \exists\ \mathscr{I}'$, *flatten* $n\ \mathscr{E}\ \mathscr{I}'\ \mathscr{L}' \land \mathscr{I}_\mu(\alpha) = \mathscr{I}'$.

**Lemma** *fetch_complete_l* : $\forall\ \{n\ l\ c'\ \mathscr{I}\ \mathscr{E}\ \mathscr{L}\}$,
*flatten* $n\ \mathscr{E}\ \mathscr{I}\ \mathscr{L} \rightarrow \mathscr{L}(l) = c' \rightarrow \exists\ k\ c,\ \mathscr{E}(n-k) = c\ \land\ c^\diamond =_c c' \land \mathscr{I}(l) = k$.

**Lemma** *fetch_mu_complete_l* : $\forall\ \{n\ \mathscr{S}'\ \mathscr{E}'_\mu\}$,
$\mathscr{E}'_\mu(n) = \mathscr{S}' \rightarrow \forall \mathscr{E}_\mu, \mathscr{E}^\diamond_\mu =_k \mathscr{E}'_\mu \rightarrow \exists \mathscr{S}, \mathscr{S}^\diamond =_s \mathscr{S}' \land \mathscr{E}_\mu(n) = \mathscr{S}$.

**Theorem** *completeness_l* : $\forall\ \sigma'_1\ \sigma_2{}'\ \sigma_1, \sigma'_1 \rightsquigarrow^l \sigma'_2\ \rightarrow\ \sigma_1^\diamond =_\sigma \sigma'_1\ \rightarrow\ \exists \sigma_2,\ \sigma_1 \rightsquigarrow^i \sigma_2\ \land\ \sigma_2{}^\diamond =_\sigma \sigma'_2$.

# 5 Conclusion and future work