Defining co-inductive types in second-order subtractive logic
(and getting a coroutine-based implementation of streams)

Tristan Crolard

CNAM – CEDRIC/CPR

Séminaire Deducteam
16 Novembre 2012
Outline

I. Duality in intuitionistic logic
   - Semantics
   - Proof theory
   - Computational content

II. Environment machines
   - Continuations
   - Coroutines

III. Second-order subtractive logic
   - Encoding inductive and co-inductive types
   - Streams and state-based generators
   - Streams and coroutine-based generators
I. Duality in intuitionistic logic
Motivation

\[
\begin{array}{cccc}
\Rightarrow & - \\
\land & \lor \\
\top & \bot \\
\forall^2 & \exists^2 \\
\forall & \exists \\
\bot \\
\end{array}
\]

\[(A - B)^\perp \equiv (B^\perp \Rightarrow A^\perp)\]

In classical logic, \( A \Rightarrow B \equiv A^\perp \lor B \) and \( A - B \equiv A \land B^\perp \)

In intuitionistic logic, subtraction not definable.
Terminology

**Subtractive Logic** = Intuitionistic Logic + subtraction
= Heyting-Brouwer Logic
= Dual intuitionistic Logic
= Bi-intuitionistic logic

≠ Classical logic (+ subtraction)
Related works

- Algebraic, topological and Kripke semantics (C. Rauszer 1974)
- Cut elimination for a deduction system à la Gentzen (C. Rauszer 1980)
- Extension with modalities (F. Wolter 1998)
- The duality of computation (H. Herbelin and P.-L. Curien 2000)
- Logic for pragmatics (G. Bellin 2002)
- Display calculus (R. Goré 2000, L. Pinto and T. Uustalu 2009)
- Labelled sequent calculus (D. Galmiche and D. Méry 2011)
- Dual-intuitionistic Nets (O. Laurent 2011)

Terminology

subtraction = pseudo-difference = co-implication
(Skolem) (Rauszer) (Wolter)
Heyting-Brouwer algebras

Bounded preorder

\[ \bot \leq x \quad x \leq \top \quad x \leq x \]

\[ x \leq y \quad y \leq z \quad \frac{\cancel{x \leq y}}{x \leq z} \]

Meet

(least upper bound)

\[ x \sqcap y \leq x \quad x \sqcap y \leq y \]

\[ z \leq x \quad z \leq y \quad \frac{\cancel{z \leq x}}{z \leq x \sqcap y} \]

Join

(greatest lower bound)

\[ x \leq x \sqcup y \quad y \leq x \sqcup y \]

\[ x \leq z \quad y \leq z \quad \frac{\cancel{x \leq z}}{x \sqcup y \leq z} \]

Implication

(relative pseudo-complement)

\[ (y \Rightarrow x) \sqcap y \leq x \]

\[ z \sqcap y \leq x \quad \frac{\cancel{z \sqcap y \leq x}}{z \leq y \Rightarrow x} \]

Subtraction

\[ x \leq (x - y) \sqcup y \]

\[ x \leq y \sqcup z \quad \frac{\cancel{x \leq y \sqcup z}}{x - y \leq z} \]
A categorical sequent calculus

\[
\begin{align*}
\bot & \vdash A \\
A & \vdash \top \\
A & \vdash A \\
\frac{A \vdash B \quad B \vdash C}{A \vdash C} \\
A \land B & \vdash A \\
A \land B & \vdash B \\
\frac{C \vdash A \quad C \vdash B}{C \vdash A \land B} \\
A & \vdash A \lor B \\
B & \vdash A \lor B \\
\frac{A \vdash C \quad B \vdash C}{A \lor B \vdash C} \\
(B \Rightarrow A) \land B & \vdash A \\
\frac{C \land B \vdash A}{C \vdash B \Rightarrow A} \\
A & \vdash (A - B) \lor B \\
\frac{A \vdash B \lor C}{A - B \vdash C}
\end{align*}
\]
Topological spaces

Any topological space \((X, \mathcal{O})\) is a Heyting algebra where:

\[
\begin{align*}
\bot & \equiv \emptyset \\
\top & \equiv X \\
A \cap B & \equiv A \cap B \\
A \cup B & \equiv A \cup B \\
A \Rightarrow B & \equiv \text{int}(A^c \cup B)
\end{align*}
\]

Since,

\[
A \subseteq B \iff B^c \subseteq A^c
\]

any co-topological space (defined by the closed sets) is a Brouwer algebra.

**Definition.** A bi-topological space is a topological space whose dual is also a topological space.
Bi-topological semantics

\[
\begin{align*}
\llbracket \bot \rrbracket &\equiv \emptyset \\
\llbracket \top \rrbracket &\equiv X \\
\llbracket A \rrbracket &\equiv \nu(A) \text{ if } A \text{ is atomic} \\
\llbracket A \land B \rrbracket &\equiv \llbracket A \rrbracket \cap \llbracket B \rrbracket \\
\llbracket A \lor B \rrbracket &\equiv \llbracket A \rrbracket \cup \llbracket B \rrbracket \\
\llbracket A \Rightarrow B \rrbracket &\equiv \text{int}(\llbracket A \rrbracket^c \cup \llbracket B \rrbracket) \\
\llbracket A - B \rrbracket &\equiv \text{cov}(\llbracket A \rrbracket \cup \llbracket B \rrbracket^c)
\end{align*}
\]

where

- \( \text{cov}(A) \equiv \) “the smallest open set containing \( A \)”
- \( \nu \) is a valuation function mapping atomic formulas to open sets.
Kripke semantics

Alexandroff Topology. The open sets of a bi-topological space $(X, \mathcal{O})$ are exactly the final sections of the pre-order defined by:

$$x \leq y \equiv \forall S \in \mathcal{O} (x \in S \Rightarrow y \in S)$$

and we thus obtain Kripke semantics (where the accessibility relation is precisely this pre-order)

- $x \models A \equiv x \in \nu(A)$ if $A$ is atomic
- $x \models A \land B \equiv x \models A \text{ and } x \models B$
- $x \models A \lor B \equiv x \models A \text{ or } x \models B$
- $x \models A \Rightarrow B \equiv \forall y \geq x (y \not\models A \text{ or } y \models B)$
- $x \models A - B \equiv \exists y \leq x (y \models A \text{ and } y \not\models B)$

In other words:

$$x \models A \iff x \in \llbracket A \rrbracket$$
Weak negation

We define \( \neg A \equiv A \rightarrow \bot \) and by duality \( \sim A \equiv \top \mathbin{\sim} A \).

Derived rules

\[
\begin{align*}
A \land \neg A & \vdash \bot \\
T & \vdash \sim A \lor A \\
B \land A & \vdash \bot \\
& \vdash \sim A \lor B
\end{align*}
\]

Semantics

\[
\begin{align*}
\llbracket \neg A \rrbracket & \equiv \text{int} (\llbracket A \rrbracket^c) \\
x \models \neg A & \equiv \forall y \geq x (y \not\models A) \\
\llbracket \sim A \rrbracket & \equiv \text{cov} (\llbracket A \rrbracket^c) \\
x \models \sim A & \equiv \exists y \leq x (y \not\models A)
\end{align*}
\]

Remark. The sequent \( \sim \sim A \land \sim A \vdash \bot \) is not valid in subtractive logic (since its dual \( T \vdash \sim \neg A \lor \sim A \) is not valid in intuitionistic logic), but it is true in all trees.
Subtraction is undefinable from weak negation

**Theorem.** *Subtraction is not definable from the weak negation (and the other usual connectives).*

**Proof.** In the following Kripke model, the semantics of \( A - B \) is different from the semantics of any other formula:

\[
\begin{array}{c}
A, B \gamma \\
\downarrow \\
A \beta \delta A, B \\
\downarrow \\
\alpha \\
\end{array}
\]
Semantics

<table>
<thead>
<tr>
<th>Intuitionistic logic</th>
<th>Subtractive logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heyting algebras</td>
<td>Heyting-Brouwer algebras</td>
</tr>
<tr>
<td>Topological spaces</td>
<td>Bi-topological spaces</td>
</tr>
<tr>
<td>Kripke models (finite trees)</td>
<td>Kripke models (finite preorders)</td>
</tr>
</tbody>
</table>

- Propositionnal case: soundness and completeness
  \( \implies \) conservativity over intuitionistic logic.

- First-order case: no conservativity
  \( \implies \) subtraction is not definable.
First-order subtractive logic

• in intuitionistic logic, proof of $\exists x A(x) \land B \vdash \exists x (A(x) \land B)$

$$
\frac{A(x) \land B \vdash A(x) \land B}{A(x) \land B \vdash \exists x (A(x) \land B)}
\frac{A(x) \vdash B \Rightarrow \exists x (A(x) \land B)}{\exists x A(x) \vdash B \Rightarrow \exists x (A(x) \land B)}
\frac{\exists x A(x) \vdash B \Rightarrow \exists x (A(x) \land B)}{\exists x A(x) \land B \vdash \exists x (A(x) \land B)}
$$

• in subtractive logic: dual proof of $\forall x (A(x) \lor B) \vdash \forall x A(x) \lor B$ (DIS)

$$
\frac{A(x) \lor B \vdash A(x) \lor B}{\forall x (A(x) \lor B) \vdash A(x) \lor B}
\frac{\forall x (A(x) \lor B) \vdash A(x) \lor B \Rightarrow -B \vdash A(x)}{\forall x (A(x) \lor B) \vdash A(x) \lor B \Rightarrow -B \vdash \forall x A(x)}
\frac{\forall x (A(x) \lor B) \vdash A(x) \lor B \Rightarrow -B \vdash \forall x A(x)}{\forall x (A(x) \lor B) \vdash \forall x A(x) \lor B}
$$
Intuitionistic logic + DIS is a theory for the Constant Domain Logic (CDL) (where Kripke models have the same domain in all worlds)

Theorem. (Rauszer 1974) Subtractive logic is sound and complete with respect to Constant Domain Kripke models.

Corollary. Subtractive logic is conservative over CDL.

Remark. (Görnemann 1971) CDL is a constructive logic (disjunction and existence properties hold in this logic).

Note. CDL is also axiomatized by Barcan formula and its converse in system S4.
Subtractive arithmetics

- Adding DIS to Heyting arithmetics yields Peano arithmetics [Troelstra, 1973]

\[ HA + \forall x(A(x) \lor B) \vdash \forall x A(x) \lor B \equiv PA \]

(prove \( \neg A \lor A \) by induction on \( A \))

- Formulas with relativized quantifiers are conservative over Heyting arithmetics. For instance, the relativized version of DIS is **not** derivable:

\[ \forall x(nat(x) \Rightarrow (A(x) \lor B)) \vdash \forall x(nat(x) \Rightarrow A(x) \lor B) \]

(standard trick used to embed intuitionistic logic into CDL)

**Note.** By the way, what is the dual of \( nat \)?
Computational content

<table>
<thead>
<tr>
<th>→</th>
<th>¬</th>
</tr>
</thead>
<tbody>
<tr>
<td>∧</td>
<td>∀</td>
</tr>
<tr>
<td>⊤</td>
<td>⊥</td>
</tr>
<tr>
<td>∀^2</td>
<td>∃^2</td>
</tr>
<tr>
<td>∀</td>
<td>∃</td>
</tr>
</tbody>
</table>

Where _⊥_ represents:

- duality in intuitionistic logic and negation in classical logic
Where \( \perp \) represents:

- duality in intuitionistic logic and negation in classical logic
- classical negation permits to type first-class continuations
(a) function
(a) product
(b, d) polymorphism
(a, d) dependent product

\[
\begin{array}{ccc}
  \rightarrow & \neg \\
  \wedge & \vee \\
  \top & \bot \\
  \forall^2 & \exists^2 \\
  \forall & \exists \\
\end{array}
\]

disjoint sum (a)
void (a)
abstract datatype (b, d)
dependent sum (a, d)

Where \( \bot \) represents:

- duality in intuitionistic logic and negation in classical logic
- classical negation permits to type first-class continuations

Références:

a) [Curry and Feys, 1958] [Howard, 1969]
b) [Girard, 1972] [Reynolds, 1974] [Mitchell and Plotkin, 1985]
c) [Griffin, 1990] [Murthy, 1990]
d) [Leivant, 1990] [Krivine and Parigot, 1990] [Parigot, 1992]
Computational content of duality

Overview

• Consider a deduction system with multi-conclusion sequents: Parigot’s CND ($\lambda\mu$-calculus) or Gentzen’s LK (and Herbelin’s $\bar{\lambda}\mu\tilde{\mu}$-calculus)
• Restrict the system to intuitionistic logic
• Consider the restriction on proof-terms
• Improve in order to enjoy stability under reduction (cut-elimination)
• Define $A - B$ as $A \land \neg B$
• Check the corresponding intro./elim. rules
• Restrict the calculus to subtractive logic
Restricting CND to intuitionistic logic

\[ \Gamma, A \vdash \Delta; A \]

\[ \frac{\Gamma, A \vdash \Delta; B}{\Gamma \vdash \Delta; A \rightarrow B} (I\rightarrow) \quad \frac{\Gamma \vdash \Delta; A \rightarrow B}{\Gamma \vdash \Delta; B} (E\rightarrow) \]

\[ \frac{\Gamma \vdash \Delta; A}{\Gamma \vdash \Delta, A; B} (W_R) \quad \frac{\Gamma \vdash \Delta, A; A}{\Gamma \vdash \Delta; A} (C_R) \]

**Remark.** Restricting \((I\rightarrow)\) to an empty \(\Delta\) yields intuitionistic logic but the system is not stable under proof reduction.

**Note.** In the first-order framework, we get CDL: schema DIS is derivable unless we restrict also \((I\forall)\).
Dependency relations for CND

Consider the sequent $A, B, C \vdash D, E, F, G$ with the following dependencies:

\[
\begin{array}{cccccc}
A, & B, & C & \vdash & D, & E, \\
& & & & F, & G
\end{array}
\]

Using named hypotheses $A^x, B^y, C^z \vdash D, E, F, G$, this sequent may be represented as:

$A^x, B^y, C^z \vdash \{z\}: D, \{x, z\}: E, \{\}: F, \{x, z\}: G$

Intuitionistic rule for $\rightarrow$-intro:

\[
\frac{\Gamma, A^x \vdash S_1: \Delta_1, \ldots, S_n: \Delta_n, V: B}{\Gamma \vdash S_1: \Delta_1, \ldots, S_n: \Delta_n, V \setminus \{x\}: (A \rightarrow B)}
\]

when $x \notin S_1 \cup \ldots \cup S_n$

where the side condition says: no conclusion other than $B$ can depend on $A$
Dependency relations

- Constructive restrictions of classical deduction systems using sequents with *multiple conclusions*:
  - Cut-elimination for the *Constant Domain Logic* [Kashima, 1991]
  - Full Intuitionistic Linear Logic [Hyland and de Paiva, 1993]
  - Restriction of Parigot’s *Classical Natural Deduction* [Crolard, 1996]

- Based on *dependency relations* between hypotheses and conclusions.
- Alternative “top-down” definition of the restriction more convenient for proof-search [Pym and Ritter, 2004] [Brede, 2009]
- Both variants can be applied *directly on proof-terms* (“safe” $\lambda\mu$-terms).
- The “top-down” definition also more convenient for proving the correctness of an environment machine.
Dependency relations

- Constructive restrictions of classical deduction systems using sequents with *multiple conclusions*:
  - Cut-elimination for the *Constant Domain Logic* [Kashima, 1991]
  - Full Intuitionistic Linear Logic [Hyland and de Paiva, 1993]
  - Restriction of Parigot’s *Classical Natural Deduction* [Crolard, 1996]
- Based on *dependency relations* between hypotheses and conclusions.
  - Alternative “top-down” definition of the restriction more convenient for proof-search [Pym and Ritter, 2004] [Brede, 2009]
  - Both variants can be applied *directly on proof-terms* (“safe” $\lambda\mu$-terms).
  - The “top-down” definition also more convenient for proving the correctness of an environment machine.
Dependency relations

- Constructive restrictions of classical deduction systems using sequents with *multiple conclusions*:
  - Cut-elimination for the *Constant Domain Logic* [Kashima, 1991]
  - Full Intuitionistic Linear Logic [Hyland and de Paiva, 1993]
  - Restriction of Parigot’s *Classical Natural Deduction* [Crolard, 1996]

- Based on *dependency relations* between hypotheses and conclusions.

- Alternative “top-down” definition of the restriction more convenient for proof-search [Pym and Ritter, 2004] [Brede, 2009]

- Both variants can be applied *directly on proof-terms* (“safe” $\lambda\mu$-terms).

- The “top-down” definition also more convenient for proving the corectness of an environment machine.
Dependency relations

- Constructive restrictions of classical deduction systems using sequents with *multiple conclusions*:
  - Cut-elimination for the *Constant Domain Logic* [Kashima, 1991]
  - Full Intuitionistic Linear Logic [Hyland and de Paiva, 1993]
  - Restriction of Parigot’s *Classical Natural Deduction* [Crolard, 1996]

- Based on *dependency relations* between hypotheses and conclusions.

- Alternative “top-down” definition of the restriction more convenient for proof-search [Pym and Ritter, 2004] [Brede, 2009]

- Both variants can be applied *directly on proof-terms* (“safe” $\lambda\mu$-terms).

- The “top-down” definition also more convenient for proving the correctness of an environment machine.
Dependency relations

- Constructive restrictions of classical deduction systems using sequents with *multiple conclusions*:
  - Cut-elimination for the *Constant Domain Logic* [Kashima, 1991]
  - Full Intuitionistic Linear Logic [Hyland and de Paiva, 1993]
  - Restriction of Parigot’s *Classical Natural Deduction* [Crolard, 1996]
- Based on *dependency relations* between hypotheses and conclusions.
- Alternative “top-down” definition of the restriction more convenient for proof-search [Pym and Ritter, 2004] [Brede, 2009]
- Both variants can be applied *directly on proof-terms* (“safe” $\lambda\mu$-terms).
- The “top-down” definition also more convenient for proving the correctness of an environment machine.
Proof terms for CND

\[ x: \Gamma, A^x \vdash \Delta; A \]

\[ \frac{t: \Gamma, A^x \vdash \Delta; B}{\lambda x.t: \Gamma \vdash \Delta; A \rightarrow B} \quad (I\rightarrow) \]

\[ \frac{t: \Gamma \vdash \Delta; A \rightarrow B \quad u: \Gamma \vdash \Delta; A}{(t \ u): \Gamma \vdash \Delta; B} \quad (E\rightarrow) \]

\[ \frac{t: \Gamma \vdash \Delta; A}{\text{throw } \alpha t: \Gamma \vdash \Delta, A^\alpha; B} \quad (W_R) \]

\[ \frac{t: \Gamma \vdash \Delta, A^\alpha; A}{\text{catch } \alpha t: \Gamma \vdash \Delta; A} \quad (C_R) \]

Remark. Operators \textit{catch} and \textit{throw} are definable in the \(\lambda\mu\)-calculus as:

\[
\begin{align*}
\text{catch } \alpha t & \equiv \mu \alpha. [\alpha] t \\
\text{throw } \alpha t & \equiv \mu \bot. [\alpha] t
\end{align*}
\]
Safety

Dependency relations are defined by induction on $t$ as follows:

- $S_\emptyset(x) = \{x\}$
  $S_\delta(x) = \emptyset$

- $S_\emptyset(\lambda x. u) = S_\emptyset(u) \setminus \{x\}$
  $S_\delta(\lambda x. u) = S_\delta(u) \setminus \{x\}$

- $S_\emptyset(u v) = S_\emptyset(u) \cup S_\emptyset(v)$
  $S_\delta(u v) = S_\delta(u) \cup S_\delta(v)$

- $S_\emptyset(\text{catch } \alpha u) = S_\emptyset(u) \cup S_\alpha(u)$
  $S_\delta(\text{catch } \alpha u) = S_\delta(u)$

- $S_\emptyset(\text{throw } \alpha u) = \emptyset$
  $S_\alpha(\text{throw } \alpha u) = S_\alpha(u) \cup S_\emptyset(u)$
  $S_\delta(\text{throw } \alpha u) = S_\delta(u)$ for any $\delta \neq \alpha$

Definition. A term $t$ is safe iff for any subterm of $t$ which has the form $\lambda x. u$, for any free $\mu$-variable $\delta$ of $u$, $x \notin S_\delta(u)$.
Safety (example)

If this sequent is derivable for some term $t$:

$$t: A^x, B^y, C^z \vdash \{z\}: D^\alpha, \{x, z\}: E^\beta, \{\} : F^\gamma; \{x, z\} : G$$

then we have:

- $S_\alpha(t) = \{z\}$
- $S_\beta(t) = \{x, z\}$
- $S_\gamma(t) = \{\}$
- $S_[](t) = \{x, z\}$

Remark. You can thus decide a posteriori if a proof in CND is intuitionistic simply by checking if the (untyped) proof-term is safe.
Safety revisited

Define $\text{Safe}^{\mathcal{V}, \mathcal{V}_\mu}(t)$ by induction on $t$ as follows:

\[
\begin{align*}
\text{Safe}^{\mathcal{V}, \mathcal{V}_\mu}(x) &= x \in \mathcal{V} \\
\text{Safe}^{\mathcal{V}, \mathcal{V}_\mu}(tu) &= \text{Safe}^{\mathcal{V}, \mathcal{V}_\mu}(t) \land \text{Safe}^{\mathcal{V}, \mathcal{V}_\mu}(u) \\
\text{Safe}^{\mathcal{V}, \mathcal{V}_\mu}(\lambda x.t) &= \text{Safe}^{(x :: \mathcal{V}), \mathcal{V}_\mu}(t) \\
\text{Safe}^{\mathcal{V}, \mathcal{V}_\mu}(\text{catch } \alpha t) &= \text{Safe}^{\mathcal{V}, (\alpha \mapsto \mathcal{V} :: \mathcal{V}_\mu)}(t) \\
\text{Safe}^{\mathcal{V}, \mathcal{V}_\mu}(\text{throw } \alpha t) &= \text{Safe}^{\mathcal{V}', \mathcal{V}_\mu}(t) \quad \text{when} \quad \mathcal{V}' = \mathcal{V}_\mu(\alpha)
\end{align*}
\]

where:

- $\mathcal{V}$ is a list of variables
- $\mathcal{V}_\mu$ maps $\mu$-variables onto lists of variables

Note. This definition assumes that variables are distincts.

Remark. This is similar to the two usual ways of defining a closed term:

- either build the set of free variables and check that it is empty
- or define a function which takes as argument the set of bound variables
Example

Safe:

\[
\lambda x \\
\quad \text{catch } \alpha \\
\quad \lambda y \\
\quad \emptyset \\
\quad y \quad \text{throw } \alpha \\
\]

Not Safe:

\[
\lambda x \\
\quad \text{catch } \alpha \\
\quad \lambda y \\
\quad \emptyset \\
\quad y \quad \text{throw } \alpha \\
\]

Note. In safe terms, catch/throw are renamed get-context/set-context.
Example

Safe:

\[ \lambda x \]
\[ \text{get-context } \alpha \]
\[ \lambda y \]
\[ \@ \]
\[ y \text{ set-context } \alpha \]
\[ x \]

Not Safe:

\[ \lambda x \]
\[ \text{catch } \alpha \]
\[ \lambda y \]
\[ \@ \]
\[ y \text{ throw } \alpha \]
\[ y \]
II. Environments machines
Regular Krivine abstract machine with control

Defined for the $\lambda\mu$-calculus in [de Groote, 1998] and [Streicher and Reus, 1998].

- A closure is a tuple $(t, E, E_\mu)$ where:
  - $E$ maps variables onto closures
  - $E_\mu$ maps $\mu$-variables onto stacks of closures
- A state is a tuple $\langle t, E, E_\mu, S \rangle$ where $(t, E, E_\mu)$ is a closure and $S$ is a stack.
- Evaluation rules:

\[
\begin{align*}
\langle x, E, E_\mu, S \rangle & \rightsquigarrow \langle t, E', E'_\mu, S \rangle & \text{when } E(x) = (t, E', E'_\mu) \\
\langle t \ u, E, E_\mu, S \rangle & \rightsquigarrow \langle t, E, E_\mu, (u, E, E_\mu) :: S \rangle \\
\langle \lambda x.t, E, E_\mu, c :: S \rangle & \rightsquigarrow \langle t, (x \mapsto c; E), E_\mu, S \rangle \\
\langle \textbf{catch} \ \alpha \ t, E, E_\mu, S \rangle & \rightsquigarrow \langle t, E, (\alpha \mapsto S; E_\mu), S \rangle \\
\langle \textbf{throw} \ \alpha \ t, E, E_\mu, S \rangle & \rightsquigarrow \langle t, E, E_\mu, S' \rangle & \text{when } E_\mu(\alpha) = S'
\end{align*}
\]
**Modified machine for functional coroutines**

- A closure is a tuple $(t, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu)$ where:
  - $\mathcal{L}$ is a local environment (maps variables onto closures)
  - $\mathcal{L}_\mu$ maps $\mu$-variables onto local environments
  - $\mathcal{E}_\mu$ maps $\mu$-variables onto stacks of closures
- A state is a tuple $\langle t, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu, S \rangle$ if $(t, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu)$ is a closure and $S$ is a stack.
- Evaluation rules:
  
  \[
  \begin{align*}
  \langle x, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu, S \rangle & \xrightarrow{\text{eval}} \langle t, \mathcal{L}', \mathcal{L}_\mu', \mathcal{E}_\mu', S \rangle \quad \text{when } \mathcal{L}(x) = (t, \mathcal{L}', \mathcal{L}_\mu', \mathcal{E}_\mu') \\
  \langle t \ u, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu, S \rangle & \xrightarrow{\text{eval}} \langle t, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu, (u, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu) :: S \rangle \\
  \langle \lambda x.t, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu, c :: S \rangle & \xrightarrow{\text{eval}} \langle t, (x \mapsto c; \mathcal{L}), \mathcal{L}_\mu, \mathcal{E}_\mu, S \rangle \\
  \langle \text{get-context} \ \alpha \ t, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu, S \rangle & \xrightarrow{\text{eval}} \langle t, \mathcal{L}, (\alpha \mapsto \mathcal{L}; \mathcal{L}_\mu), (\alpha \mapsto S; \mathcal{E}_\mu), S \rangle \\
  \langle \text{set-context} \ \alpha \ t, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu, S \rangle & \xrightarrow{\text{eval}} \langle t, \mathcal{L}', \mathcal{L}_\mu, \mathcal{E}_\mu, S' \rangle \quad \text{when } \mathcal{L}_\mu(\alpha) = \mathcal{L}', \ \mathcal{E}_\mu(\alpha) = S'
  \end{align*}
  \]
Moving to de Bruijn indices

- Usual de Bruijn indices are not correct for local environments.
- Need to introduce a notion of local indices.
- Define a translation from local indices to global indices.

\[
\begin{array}{ccc}
\lambda & \Rightarrow & \lambda \\
\text{get-context} & & \text{catch} \\
\lambda & & \lambda \\
@ & & @ \\
0 & & 0 \\
\text{set-context} & & \text{throw} \\
0 & & 0 \\
0 & & 1 \\
\end{array}
\]
Bisimulation

- Define an intermediate machine with local indices, global environment and indirection tables.

- Define two functional bi-simulations \((-)^*\) and \((-)^\diamond\) showing that this intermediate machine:
  - bi-simulates the regular machine with global indices
  - bi-simulates the modified machine with local environments

\[\begin{align*}
\text{Regular machine} & : ~ \tilde{\sigma}_0^* \rightarrow \cdots \rightarrow \tilde{\sigma}_n^* \rightarrow \tilde{\sigma}_{n+1}^* \rightarrow \cdots \\
 & \quad \uparrow^* \quad \uparrow^* \quad \uparrow^* \\
\text{Intermediate machine} & : ~ \tilde{\sigma}_0 \rightarrow \cdots \rightarrow \tilde{\sigma}_n \rightarrow \tilde{\sigma}_{n+1} \rightarrow \cdots \\
 & \quad \downarrow^\diamond \quad \downarrow^\diamond \quad \downarrow^\diamond \\
\text{Modified machine} & : ~ \tilde{\sigma}_0^\diamond \rightarrow \cdots \rightarrow \tilde{\sigma}_n^\diamond \rightarrow \tilde{\sigma}_{n+1}^\diamond \rightarrow \cdots
\end{align*}\]

- Proof completely formalized in Twelf.
First-class coroutines

Introduction rule ($I_-$)

\[
\frac{t: \Gamma \vdash \Delta; A}{\text{make-coroutine } t \, \beta: \Gamma \vdash \Delta, B^\beta; A - B}
\]

Elimination rule ($E_-$)

\[
\frac{t: \Gamma \vdash \Delta; A - B \quad u: \Gamma, A^x \vdash \Delta; B}{\text{resume } t \text{ with } x \mapsto u: \Gamma \vdash \Delta; C}
\]

Remark. By duality, the constructive restriction (the safety requirement) is over ($E_-$) and says that there is no dependency between $\Gamma$ and $B$. In other words, the initial environment for the resumed coroutine is given as $x$. 
Defining $A - B$ as $A \land \neg B$

\[
\text{make-coroutine } t \beta \equiv (t, \lambda x. \text{set-context } \beta x) \\
\text{resume } t \text{ with } x \mapsto u \equiv \text{match } t \text{ with } (x, k) \rightarrow \text{abort } (k \ u)
\]

Derivation of the introduction rule

\[
x: \Gamma, B^x \vdash \Delta; B \\
\hline
\text{set-context } \beta x: \Gamma \vdash \Delta, B^\beta; \bot \\
\hline
\lambda x. \text{set-context } \beta x: \Gamma \vdash \Delta, B^\beta; \neg D \\
\hline
(t, \lambda x. \text{set-context } \beta x): \Gamma \vdash \Delta, B^\beta; A - B
\]

Derivation of the elimination rule

\[
k: \neg B^k \vdash \neg B \\
u: \Gamma, A^x \vdash \Delta; B \\
\hline
(k \ u): \Gamma, A^x, \neg B^k \vdash \Delta; \bot \\
\hline
\text{match } t \text{ with } (x, k) \mapsto \text{abort } (k \ u): \Gamma \vdash \Delta; C
\]
III. Second-order subtractive logic
Encoding inductive and co-inductive types

Inductive types
Recall that if $F(X)$ is a type where $X$ occurs only positively, the least fixpoint is definable as:

$$\mu X. F(X) \equiv \forall X. (F(X) \to X) \to X$$

Co-inductive types
By duality. The greatest fixpoint is definable as:

$$\nu X. F(X) \equiv \exists X. X \times (X \to F(X))$$

Remark. The usual encoding as a state machine is given by the following definition:

$$\nu X. F(X) \equiv \exists X. X \times (X \to F(X))$$
Usual encoding of co-inductive types

(* $F$ is the functor derived from type $F(X)$ *)

val $F : (X \to Y) \to (F(X) \to F(Y))$

type $stream = \exists X.(X \to F(X)) \times X$

let $unfold : \forall X.(X \to F(X)) \to X \to stream =$
  fun $X f x \to (X, x, f)$

let $out : stream \to F(stream) =$
  fun $(X, next, current) = F (unfold X next) (next current)$

(* Special case $F(X) = int \times X *$)

let $head : stream \to int = fun s \to fst (out s)$

let $tail : stream \to stream = fun s \to snd (out s)$
Implementation obtained by duality

type generator = \exists X. X - (X - F(X))

(* unfold = fold \bot *)
let unfold : \forall X. (X \to F(X)) \to X \to generator =
  fun X k s \rightarrow
  get-context \alpha
  resume (swap-context \beta \alpha (X, make-coroutine (s, \beta)))
  with k

(* out = in \bot *)
let out : generator \to F(generator) =
  fun g \rightarrow
  match g with (s, \kappa) \rightarrow
  let k s = swap-context \beta \kappa (make-coroutine (s, \beta))
  in F (unfold k) (k s)
Bibliography


