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

$\Rightarrow$	$-$
$\wedge$	$\vee$
$\top$	$\bot$
$\forall^2$	$\exists^2$
$\forall$	$\exists$

$\bot$

$$(A - B)^\bot \equiv (B^\bot \Rightarrow A^\bot)$$

In classical logic, $A \Rightarrow B \equiv A^\bot \vee B$ and $A - B \equiv A \wedge B^\bot$
In intuitionistic logic, subtraction not definable.

Terminology

$$
\begin{aligned}
\textbf{Subtractive Logic} &= \text{Intuitionistic Logic} + \text{subtraction} \\
&= \text{Heyting-Brouwer Logic} \\
&= \text{Dual intuitionistic Logic} \\
&= \text{Bi-intuitionistic logic} \\
\\
&\neq \text{Classical logic } (+ \text{ subtraction})
\end{aligned}
$$

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	$\bot \leqslant x \qquad x \leqslant \top \qquad x \leqslant x$	$\dfrac{x \leqslant y \qquad y \leqslant z}{x \leqslant z}$
Meet (least upper bound)	$x \sqcap y \leqslant x \qquad x \sqcap y \leqslant y$	$\dfrac{z \leqslant x \qquad z \leqslant y}{z \leqslant x \sqcap y}$
Join (greatest lower bound)	$x \leqslant x \sqcup y \qquad y \leqslant x \sqcup y$	$\dfrac{x \leqslant z \qquad y \leqslant z}{x \sqcup y \leqslant z}$
Implication (relative pseudo-complement)	$(y \Rightarrow x) \sqcap y \leqslant x$	$\dfrac{z \sqcap y \leqslant x}{z \leqslant y \Rightarrow x}$
Subtraction	$x \leqslant (x - y) \sqcup y$	$\dfrac{x \leqslant y \sqcup z}{x - y \leqslant z}$

A categorical sequent calculus

$$\bot \vdash A \qquad A \vdash \top \qquad A \vdash A \qquad \frac{A \vdash B \qquad B \vdash C}{A \vdash C}$$

$$A \wedge B \vdash A \qquad A \wedge B \vdash B \qquad \frac{C \vdash A \qquad C \vdash B}{C \vdash A \wedge B}$$

$$A \vdash A \vee B \qquad B \vdash A \vee B \qquad \frac{A \vdash C \qquad B \vdash C}{A \vee B \vdash C}$$

$$(B \Rightarrow A) \wedge B \vdash A \qquad \frac{C \wedge B \vdash A}{C \vdash B \Rightarrow A}$$

$$A \vdash (A - B) \vee B \qquad \frac{A \vdash B \vee C}{A - B \vdash C}$$

Topological spaces

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

$$\begin{aligned} \bot &\equiv \emptyset \\ \top &\equiv X \\ A \sqcap B &\equiv A \cap B \\ A \sqcup B &\equiv A \cup B \\ A \Rightarrow B &\equiv int(A^c \cup B) \end{aligned}$$

Since,

$$A \subseteq B \quad \text{iff} \quad B^c \subseteq A^c$$

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

$$
\begin{array}{rcl}
[\![\bot]\!] & \equiv & \emptyset \\
[\![\top]\!] & \equiv & X \\
[\![A]\!] & \equiv & \nu(A) \text{ if } A \text{ is atomic} \\
[\![A \wedge B]\!] & \equiv & [\![A]\!] \cap [\![B]\!] \\
[\![A \vee B]\!] & \equiv & [\![A]\!] \cup [\![B]\!] \\
[\![A \Rightarrow B]\!] & \equiv & int([\![A]\!]^c \cup [\![B]\!]) \\
[\![A - B]\!] & \equiv & cov([\![A]\!] \cup [\![B]\!]^c)
\end{array}
$$

where

- $cov(A) \equiv$ "the smallest open set containing A"
- ν 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)

$$\begin{array}{lcl}
x \Vdash A & \equiv & x \in \nu(A) \text{ if } A \text{ is atomic} \\
x \Vdash A \wedge B & \equiv & x \Vdash A \textit{ and } x \Vdash B \\
x \Vdash A \vee B & \equiv & x \Vdash A \textit{ or } x \Vdash B \\
x \Vdash A \Rightarrow B & \equiv & \forall y \geq x \, (y \nVdash A \textit{ or } y \Vdash B) \\
x \Vdash A - B & \equiv & \exists y \leq x \, (y \Vdash A \textit{ and } y \nVdash B)
\end{array}$$

In other words:

$$x \Vdash A \quad \text{iff} \quad x \in [\![A]\!]$$

Weak negation

We define $\neg A \equiv A \rightarrow \bot$ and by duality ${\sim}A \equiv \top - A$.

Derived rules

$$A \wedge \neg A \vdash \bot \qquad \top \vdash {\sim}A \vee A$$

$$\frac{B \wedge A \vdash \bot}{B \vdash \neg A} \qquad \frac{\top \vdash A \vee B}{{\sim}A \vdash B}$$

Semantics

$$[\![\neg A]\!] \equiv int([\![A]\!]^c) \qquad x \Vdash \neg A \equiv \forall y \geq x (y \not\Vdash A)$$

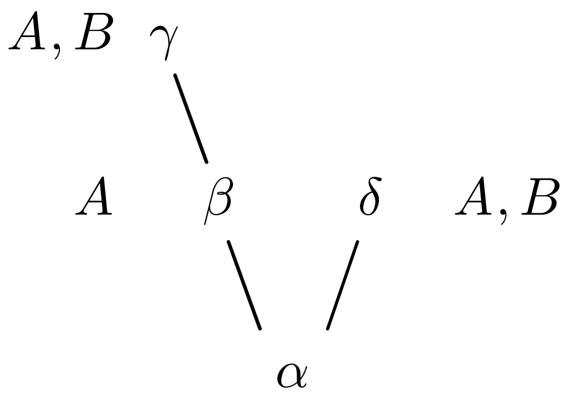
$$[\![{\sim}A]\!] \equiv cov([\![A]\!]^c) \qquad x \Vdash {\sim}A \equiv \exists y \leq x (y \not\Vdash A)$$

Remark. The sequent ${\sim}{\sim}A \wedge {\sim}A \vdash \bot$ is not valid in subtractive logic (since its dual $\top \vdash \neg\neg A \vee \neg A$ 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:



∎

Semantics

Intuitionistic logic	**Subtractive logic**
Heyting algebras Topological spaces Kripke models (finite trees)	Heyting-Brouwer algebras Bi-topological spaces Kripke models (finite preorders)

- Propositionnal case : soundness and completeness
  $\Longrightarrow$ conservativity over intuitionistic logic.
- First-order case : no conservativity
  $\Longrightarrow$ subtraction is not definable.

First-order subtractive logic

- in intuitionistic logic, proof of $\exists x\, A(x) \wedge B \vdash \exists x(A(x) \wedge B)$

$$\cfrac{\cfrac{\cfrac{\cfrac{A(x) \wedge B \vdash A(x) \wedge B}{A(x) \wedge B \vdash \exists x(A(x) \wedge B)}}{A(x) \vdash B \Rightarrow \exists x(A(x) \wedge B)}}{\exists x\, A(x) \vdash B \Rightarrow \exists x(A(x) \wedge B)}}{\exists x\, A(x) \wedge B \vdash \exists x(A(x) \wedge B)}$$

- in subtractive logic: dual proof of $\forall x(A(x) \vee B) \vdash \forall x\, A(x) \vee B$ (DIS)

$$\cfrac{\cfrac{\cfrac{\cfrac{A(x) \vee B \vdash A(x) \vee B}{\forall x(A(x) \vee B) \vdash A(x) \vee B}}{\forall x(A(x) \vee B) - B \vdash A(x)}}{\forall x(A(x) \vee B) - B \vdash \forall x\, A(x)}}{\forall x(A(x) \vee B) \vdash \forall x\, A(x) \vee B}$$

Constant Domain Logic

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]

$$HA \; + \; \forall x(A(x) \vee B) \vdash \forall x \, A(x) \vee B \quad \equiv \quad PA$$

(prove $\neg A \vee A$ by induction on A)

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

$$\forall x(nat(x) \Rightarrow (A(x) \vee B)) \vdash \forall x(nat(x) \Rightarrow A(x) \vee B$$

(standard trick used to embed intuitionistic logic into CDL)

Note. By the way, what is the dual of nat ?

Computational content

$\rightarrow$	$-$
$\wedge$	$\vee$
$\top$	$\perp$
$\forall^2$	$\exists^2$
$\forall$	$\exists$

$\perp$

Where $\_^{\perp}$ represents:

- duality in intuitionistic logic and negation in classical logic

Computational content

function	$\rightarrow$	$-$	
product	$\wedge$	$\vee$	disjoint sum
unit	$\top$	$\bot$	void
polymorphism	$\forall^2$	$\exists^2$	abstract datatype
dependent product	$\forall$	$\exists$	dependent sum

$$\underbrace{\hspace{2em}}_{\bot}$$

Where $\_^{\bot}$ represents:

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

Computational content

(a) function	$\rightarrow$	$-$	
(a) product	$\wedge$	$\vee$	disjoint sum (a)
(a) unit	$\top$	$\bot$	void (a)
(b, d) polymorphism	$\forall^2$	$\exists^2$	abstract datatype (b, d)
(a, d) dependent product	$\forall$	$\exists$	dependent sum (a, d)

$^{\perp}$ (c)

Where $\_^{\perp}$ 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 $A \wedge \neg B$
- Check the corresponding intro./elim. rules
- Restrict the calculus to subtractive logic

Restricting CND to intuitionistic logic

$$\Gamma, A \vdash \Delta; A$$

$$\frac{\Gamma, A \vdash \Delta; B}{\Gamma \vdash \Delta; A \to B}\,(I_{\to}) \qquad\qquad \frac{\Gamma \vdash \Delta; A \to B \qquad \Gamma \vdash \Delta; A}{\Gamma \vdash \Delta; B}\,(E_{\to})$$

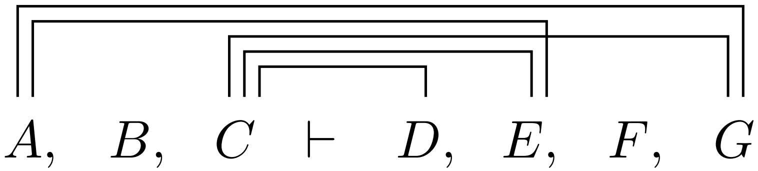
$$\frac{\Gamma \vdash \Delta; A}{\Gamma \vdash \Delta, A; B}\,(W_R) \qquad \frac{\Gamma \vdash \Delta, A; A}{\Gamma \vdash \Delta; A}\,(C_R)$$

Remark. Restricting $(I_{\to})$ to an empty Δ 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:



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:

$$\frac{\Gamma, A^x \vdash S_1: \Delta_1, ..., S_n: \Delta_n, V: B}{\Gamma \vdash S_1: \Delta_1, ..., S_n: \Delta_n, V \setminus \{x\}: (A \rightarrow B)} \quad \text{when } x \notin S_1 \cup ... \cup S_n$$

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 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 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 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 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 corectness of an environment machine.

Proof terms for CND

$$x\colon \Gamma, A^x \vdash \Delta; A$$

$$\frac{t\colon \Gamma, A^x \vdash \Delta; B}{\lambda x.t\colon \Gamma \vdash \Delta; A \to B}\,(I_\to) \qquad \frac{t\colon \Gamma \vdash \Delta; A \to B \quad u\colon \Gamma \vdash \Delta; A}{(t\,u)\colon \Gamma \vdash \Delta; B}\,(E_\to)$$

$$\frac{t\colon \Gamma \vdash \Delta; A}{\textbf{throw}\ \alpha\ t\colon \Gamma \vdash \Delta, A^\alpha; B}\,(W_R) \qquad \frac{t\colon \Gamma \vdash \Delta, A^\alpha; A}{\textbf{catch}\ \alpha\ t\colon \Gamma \vdash \Delta; A}\,(C_R)$$

Remark. Operators **catch** and **throw** are definable in the $\lambda\mu$-calculus as:

$$\begin{aligned} \textbf{catch}\,\alpha\ t &\equiv \mu\alpha.[\alpha]t \\ \textbf{throw}\,\alpha\ t &\equiv \mu\_.[\alpha]t \end{aligned}$$

Safety

Dependency relations are defined by induction on t as follows:

- $\mathcal{S}_{[]}(x) = \{x\}$
  $\mathcal{S}_{\delta}(x) = \emptyset$
- $\mathcal{S}_{[]}(\lambda x.u) = \mathcal{S}_{[]}(u) \backslash \{x\}$
  $\mathcal{S}_{\delta}(\lambda x.u) = \mathcal{S}_{\delta}(u) \backslash \{x\}$
- $\mathcal{S}_{[]}(u\,v) = \mathcal{S}_{[]}(u) \cup \mathcal{S}_{[]}(v)$
  $\mathcal{S}_{\delta}(u\,v) = \mathcal{S}_{\delta}(u) \cup \mathcal{S}_{\delta}(v)$
- $\mathcal{S}_{[]}(\textbf{catch}\, \alpha\ u) = \mathcal{S}_{[]}(u) \cup \mathcal{S}_{\alpha}(u)$
  $\mathcal{S}_{\delta}(\textbf{catch}\, \alpha\ u) = \mathcal{S}_{\delta}(u)$
- $\mathcal{S}_{[]}(\textbf{throw}\, \alpha\ u) = \emptyset$
  $\mathcal{S}_{\alpha}(\textbf{throw}\, \alpha\ u) = \mathcal{S}_{\alpha}(u) \cup \mathcal{S}_{[]}(u)$
  $\mathcal{S}_{\delta}(\textbf{throw}\, \alpha\ u) = \mathcal{S}_{\delta}(u)$ 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 μ-variable δ of u, $x \notin \mathcal{S}_{\delta}(u)$.*

Safety (example)

If this sequent is derivable for some term t:

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

then we have:

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

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

$$
\begin{array}{rcl}
Safe^{\mathcal{V},\mathcal{V}_\mu}(x) & = & x \in \mathcal{V} \\
Safe^{\mathcal{V},\mathcal{V}_\mu}(t\,u) & = & Safe^{\mathcal{V},\mathcal{V}_\mu}(t) \wedge Safe^{\mathcal{V},\mathcal{V}_\mu}(u) \\
Safe^{\mathcal{V},\mathcal{V}_\mu}(\lambda x.t) & = & Safe^{(x\,::\,\mathcal{V}),\mathcal{V}_\mu}(t) \\
Safe^{\mathcal{V},\mathcal{V}_\mu}(\textbf{catch}\ \alpha\ t) & = & Safe^{\mathcal{V},(\alpha\mapsto\mathcal{V};\mathcal{V}_\mu)}(t) \\
Safe^{\mathcal{V},\mathcal{V}_\mu}(\textbf{throw}\ \alpha\ t) & = & Safe^{\mathcal{V}',\mathcal{V}_\mu}(t) \quad when \quad \mathcal{V}' = \mathcal{V}_\mu(\alpha)
\end{array}
$$

where:

– $\mathcal{V}$ is a list of variables

– $\mathcal{V}_\mu$ maps μ-variables onto lists of variables

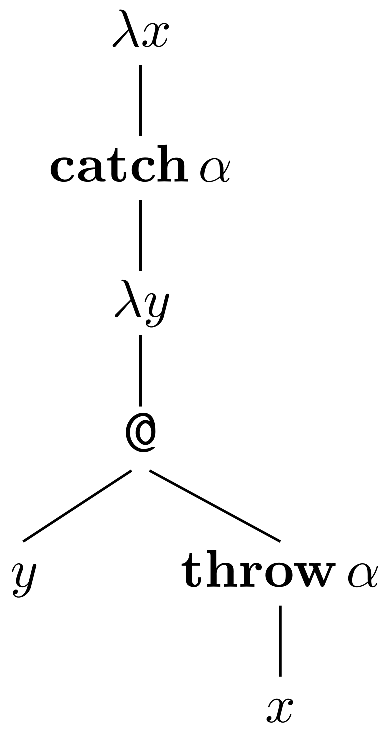
Note. This definition assumes that variables are distincts.

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

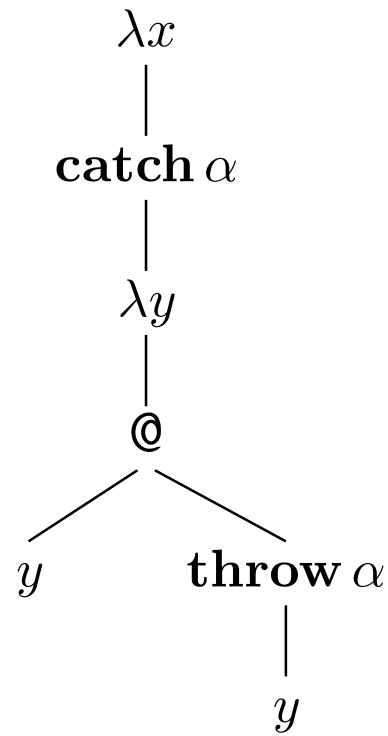
- either build the set of free variables and check that it is empty
- or define a function which takes as argument the set of bound variables

Example

Safe:



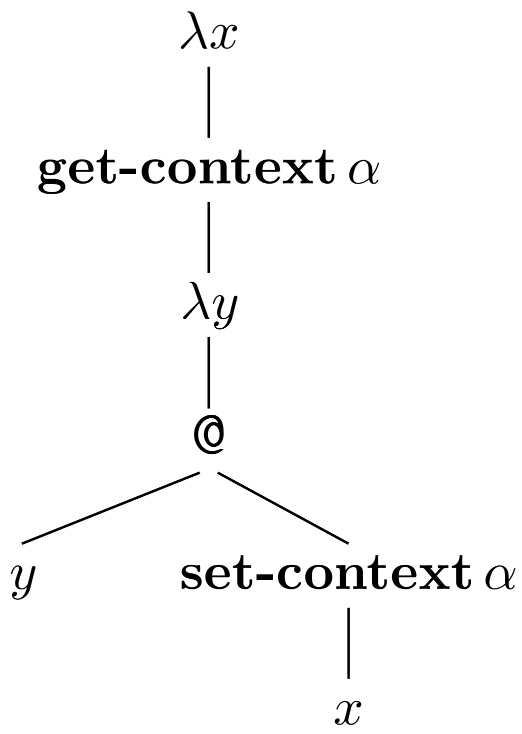
Not Safe:



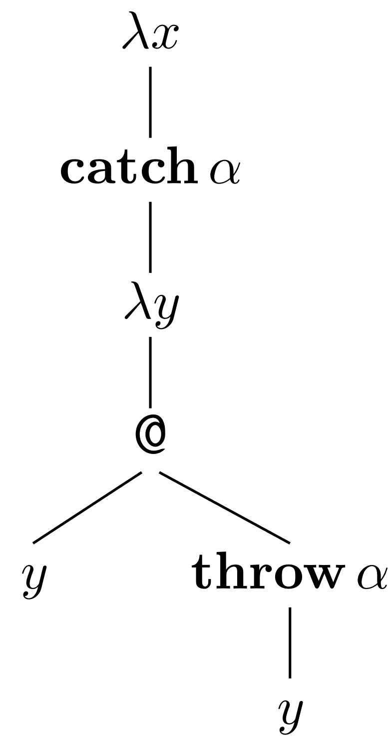
Note. In safe terms, **catch/throw** are renamed **get-context/set-context**.

Example

Safe:



Not Safe:



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 $(t, \mathcal{E}, \mathcal{E}_\mu)$ where:

- $\mathcal{E}$ maps variables onto closures
- $\mathcal{E}_\mu$ maps μ-variables onto stacks of closures

– A state is a tuple $\langle t, \mathcal{E}, \mathcal{E}_\mu, \mathcal{S}\rangle$ where $(t, \mathcal{E}, \mathcal{E}_\mu)$ is a closure and $\mathcal{S}$ is a stack.

– Evaluation rules:

$$
\begin{array}{lcll}
\langle x, \mathcal{E}, \mathcal{E}_\mu, \mathcal{S}\rangle & \leadsto & \langle t, \mathcal{E}', \mathcal{E}'_\mu, \mathcal{S}\rangle & \textit{when } \mathcal{E}(x) = (t, \mathcal{E}', \mathcal{E}'_\mu) \\
\langle t\ u, \mathcal{E}, \mathcal{E}_\mu, \mathcal{S}\rangle & \leadsto & \langle t, \mathcal{E}, \mathcal{E}_\mu, (u, \mathcal{E}, \mathcal{E}_\mu) :: \mathcal{S}\rangle & \\
\langle \lambda x.t, \mathcal{E}, \mathcal{E}_\mu, c :: \mathcal{S}\rangle & \leadsto & \langle t, (x \mapsto c; \mathcal{E}), \mathcal{E}_\mu, \mathcal{S}\rangle & \\
\langle \textbf{catch}\ \alpha\ t, \mathcal{E}, \mathcal{E}_\mu, \mathcal{S}\rangle & \leadsto & \langle t, \mathcal{E}, (\alpha \mapsto \mathcal{S}; \mathcal{E}_\mu), \mathcal{S}\rangle & \\
\langle \textbf{throw}\ \alpha\ t, \mathcal{E}, \mathcal{E}_\mu, \mathcal{S}\rangle & \leadsto & \langle t, \mathcal{E}, \mathcal{E}_\mu, \mathcal{S}'\rangle & \textit{when } \mathcal{E}_\mu(\alpha) = \mathcal{S}'
\end{array}
$$

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 enviroment (maps variables onto closures)
- $\mathcal{L}_\mu$ maps μ-variables onto local environments
- $\mathcal{E}_\mu$ maps μ-variables onto stacks of closures

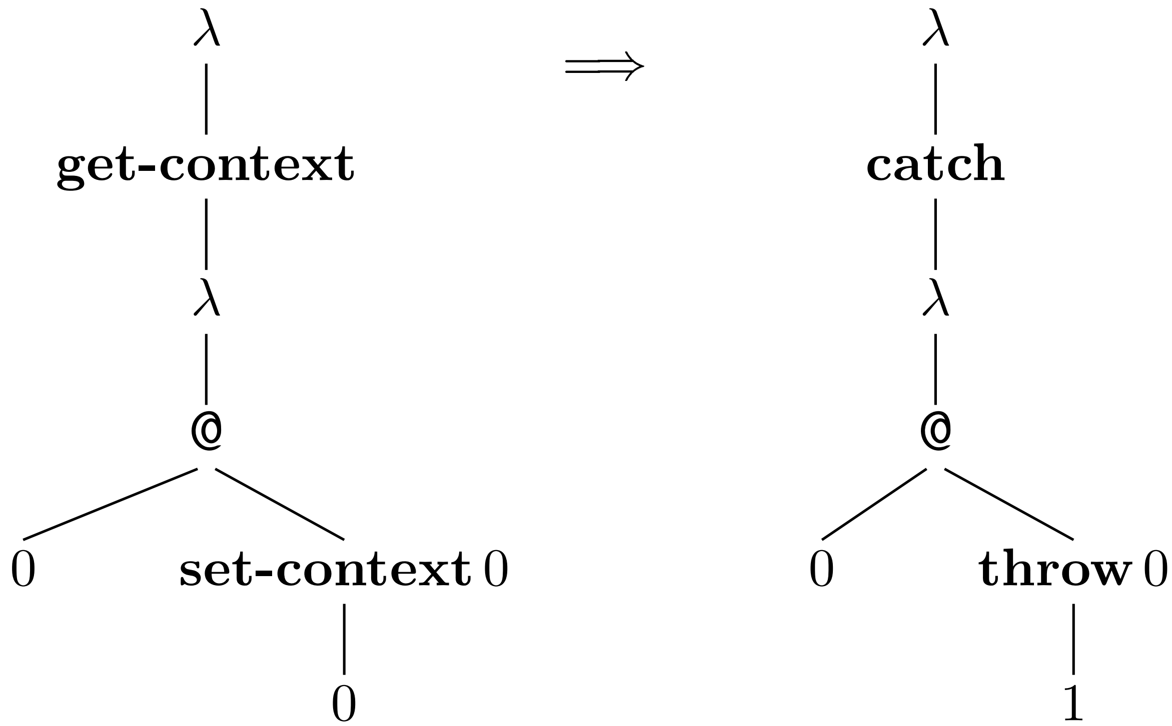
– A state is a tuple $\langle t, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu, \mathcal{S}\rangle$ if $(t, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu)$ is a closure and $\mathcal{S}$ is a stack.

– Evaluation rules:

$$\begin{array}{lcl}
\langle x, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu, \mathcal{S}\rangle & \leadsto & \langle t, \mathcal{L}', \mathcal{L}'_\mu, \mathcal{E}'_\mu, \mathcal{S}\rangle \;\; \textit{when } \mathcal{L}(x) = (t, \mathcal{L}', \mathcal{L}'_\mu, \mathcal{E}'_\mu) \\
\langle t\ u, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu, \mathcal{S}\rangle & \leadsto & \langle t, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu, (u, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu) :: \mathcal{S}\rangle \\
\langle \lambda x.t, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu, c :: \mathcal{S}\rangle & \leadsto & \langle t, (x \mapsto c; \mathcal{L}), \mathcal{L}_\mu, \mathcal{E}_\mu, \mathcal{S}\rangle \\
\langle \textbf{get-context}\ \alpha\ t, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu, \mathcal{S}\rangle & \leadsto & \langle t, \mathcal{L}, (\alpha \mapsto \mathcal{L}; \mathcal{L}_\mu), (\alpha \mapsto \mathcal{S}; \mathcal{E}_\mu), \mathcal{S}\rangle \\
\langle \textbf{set-context}\ \alpha\ t, \mathcal{L}, \mathcal{L}_\mu, \mathcal{E}_\mu, \mathcal{S}\rangle & \leadsto & \langle t, \mathcal{L}', \mathcal{L}_\mu, \mathcal{E}_\mu, \mathcal{S}'\rangle \;\; \textit{when } \mathcal{L}_\mu(\alpha) = \mathcal{L}',\ \mathcal{E}_\mu(\alpha) = \mathcal{S}'
\end{array}$$

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.



Bisimulation

- Define an intermediate machine with local indices, global environment and indirection tables.
- Define two functional bi-simulations $(\text{-})^\star$ and $(\text{-})^\diamond$ showing that this intermediate machine:
  - bi-simulates the regular machine with global indices
  - bi-simulates the modified machine with local environments

$$
\begin{array}{lccccccccc}
\textit{Regular machine} & \tilde{\sigma}_0^\star & \leadsto & \cdots & \leadsto & \tilde{\sigma}_n^\star & \leadsto & \tilde{\sigma}_{n+1}^\star & \leadsto & \cdots \\
 & \big\uparrow\star & & & & \big\uparrow\star & & \big\uparrow\star & & \\
\textit{Intermediate machine} & \tilde{\sigma}_0 & \leadsto & \cdots & \leadsto & \tilde{\sigma}_n & \leadsto & \tilde{\sigma}_{n+1} & \leadsto & \cdots \\
 & \big\downarrow\diamond & & & & \big\downarrow\diamond & & \big\downarrow\diamond & & \\
\textit{Modified machine} & \tilde{\sigma}_0^\diamond & \leadsto & \cdots & \leadsto & \tilde{\sigma}_n^\diamond & \leadsto & \tilde{\sigma}_{n+1}^\diamond & \leadsto & \cdots
\end{array}
$$

- Proof completely formalized in Twelf.

First-class coroutines

Introduction rule (I_-)

$$\frac{t\colon \Gamma \vdash \Delta; A}{\textbf{make-coroutine}\, t\ \beta\colon \Gamma \vdash \Delta, B^\beta; A - B}$$

Elimination rule (E_-)

$$\frac{t\colon \Gamma \vdash \Delta; A - B \qquad u\colon \Gamma, A^x \vdash \Delta; B}{\textbf{resume}\ t\ \textbf{with}\ x \mapsto u\colon \Gamma \vdash \Delta; C}$$

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

Defining $A - B$ as $A \wedge \neg B$

$$\begin{aligned} \textbf{make-coroutine}\, t\, \beta &\equiv (t, \lambda x.\textbf{set-context}\ \beta\ x) \\ \textbf{resume}\ t\ \textbf{with}\ x \mapsto u &\equiv \textbf{match}\ t\ \textbf{with}\ (x,k) \mapsto \textbf{abort}\ (k\ u) \end{aligned}$$

Derivation of the introduction rule

$$\dfrac{t{:}\,\Gamma \vdash \Delta; A \qquad \dfrac{\dfrac{x{:}\,\Gamma, B^x \vdash \Delta; B}{\textbf{set-context}\ \beta\ x{:}\,\Gamma \vdash \Delta, B^\beta; \bot}}{\lambda x.\textbf{set-context}\ \beta\ x{:}\,\Gamma \vdash \Delta, B^\beta; \neg D}}{(t, \lambda x.\textbf{set-context}\ \beta\ x){:}\,\Gamma \vdash \Delta, B^\beta; A - B}$$

Derivation of the elimination rule

$$\dfrac{t{:}\,\Gamma \vdash \Delta; A \wedge \neg B \qquad \dfrac{\dfrac{k{:}\,\neg B^k \vdash \neg B \qquad u{:}\,\Gamma, A^x \vdash \Delta; B}{(k\ u){:}\,\Gamma, A^x, \neg B^k \vdash \Delta; \bot}}{\textbf{abort}\ (k\ u){:}\,\Gamma, A^x, \neg B^k \vdash \Delta; C}}{\textbf{match}\ t\ \textbf{with}\ (x,k) \mapsto \textbf{abort}\ (k\ u){:}\,\Gamma \vdash \Delta; C}$$

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 :

$$\nu X.F(X) \equiv \exists X.X - (X - F(X))$$

Remark. The ususal 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 = ∃X.X − (X − F(X))

(*  unfold = fold⊥  *)
let unfold : ∀X.(X → F(X)) → X → generator =
   fun X k s →
      get-context α
         resume (swap-context β α (X, make-coroutine (s, β)))
         with k

(*  out = in⊥  *)
let out : generator → F(generator) =
   fun g →
      match g with (s, κ) →
         let k s = swap-context β κ (make-coroutine (s, β))
         in F (unfold k) (k s)
```

Bibliography

[1] N. Brede. $\lambda\ \mu$PRL - A Proof Refinement Calculus for Classical Reasoning in Computational Type Theory. Master's thesis, University of Potsdam, 2009. URL `http://www.cs.uni-potsdam.de/`.

[2] T. Crolard. Extension de l'Isomorphisme de Curry-Howard au Traitement des Exceptions (application d'une étude de la dualité en logique intuitionniste). Thèse de Doctorat. Université Paris 7, 1996.

[3] H. B. Curry and R. Feys. Combinatory Logic. North-Holland, 1958.

[4] P. de Groote. An environment machine for the lambda-mu-calculus. Mathematical Structure in Computer Science, 8: 637–669, 1998.

[5] J.-Y. Girard. Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur. PhD thesis, Thèse de doctorat d'état, Université Paris VII, 1972.

[6] T. G. Griffin. A formulæ-as-types notion of control. In Conference Record of the 17th Annual ACM Symposium on Principles of Programming Langages, pages 47–58, 1990.

[7] W. A. Howard. The Formulæ-as-types Notion of Constructions. In To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculs and Formalism, pages 479–490. Academic Press, 1969.

[8] M. Hyland and V. de Paiva. Full Intuitionistic Linear Logic (extended abstract). Annals of Pure and Applied Logic, 64 (3): 273–291, 1993.

[9] R. Kashima. Cut-Elimination for the intermediate logic CD. Research Report on Information Sciences C100, Institute of Technology, Tokyo, 1991.

[10] J.-L. Krivine and M. Parigot. Programming with proofs. J. Inf. Process. Cybern. EIK, 26 (3): 149–167, 1990.

[11] D. Leivant. Contracting proofs to programs. In Odifreddi, editor, Logic and Computer Science, pages 279–327. Academic Press, 1990.

[12] J. C. Mitchell and G. D. Plotkin. Abstract types have existential type. In 12th Annual ACM symposium on Principles of Programming Languages, 1985.

[13] C. R. Murthy. Extracting Constructive Content from Classical proofs. PhD thesis, Cornell University, Department of Computer Science, 1990.

[14] M. Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In Proc. Logic Prog. and Autom. Reasoning, volume 624 of LNCS, pages 190–201, 1992.

[15] D. Pym and E. Ritter. Reductive logic and proof-search: proof theory, semantics, and control, volume 45. Oxford University Press, USA, 2004.

[16] J. C. Reynolds. Towards a theory of type structure. In Symposium on Programming, volume 19 of Lecture Notes in Computer Science, pages 408–423. Springer, 1974. ISBN 3-540-06859-7.

[17] T. Streicher and B. Reus. Classical Logic, Continuation Semantics and Abstract Machines. Journal of Functional Programming, 8 (6): 543–572, 1998.

[18] A. S. Troelstra. Metamathematical Investigation of Intuitionistic Arithmetic and Analysis, volume 344 of Lecture Notes in Mathematics. Springer-Verlag, Berlin, 1973.