Classical logic, control calculi and data types

Robbert Krebbers

November 2, 2010 @ The Brouwer seminar

The Curry-Howard correspondence

Computation		**Logic**
Types	$\Longleftrightarrow$	Formulas
Programs	$\Longleftrightarrow$	Proofs
Reduction	$\Longleftrightarrow$	Proof normalization

Classical logic

- People originally believed:

  "Curry-Howard is limited to constructive logic"

- Classical logic contains constructive content [Kreisel, Friedman]

$$\frac{\mathsf{PA} \vdash \Pi^0_2}{\mathsf{HA} \vdash \Pi^0_2} \quad \text{Proof translation}$$

- But this is an indirect result

Griffin's discovery

- Felleisen's control operator $\mathcal{C}$ can be typed with DN

$$\frac{\Gamma \vdash t : \neg\neg\rho}{\Gamma \vdash \mathcal{C}_\rho t : \rho}$$

- Curry-Howard correspondence for classical logic!
- This is a direct result
- Continuation passing style (CPS) translation
  - Not only a simulation of control in a system without
  - Also an embedding of classical logic into constructive logic

Problem

- Many *control calculi* present in the literature
  - Which is the best?
- Most of these calculi are quite simple
  - No ‘real’ data types (natural numbers, lists, . . .)
  - No polymorphism
  - No dependent types
  - . . .
- No *program extraction* à la Paulin/Letouzey

My work

- Compared the $\lambda_{\mathcal{C}}$-, λ_{Δ}- and λ_{μ}-calculus
  - Main theoretical properties
  - Simulation of `catch` and `throw`
  - Possibility of extension with data types
- Developed the $\lambda_{\mu}^{\mathsf{T}}$-calculus
  - λ_{μ} with natural numbers and primitive recursion
- Proven its main theoretical properties
- Embedding of $\lambda_{\mu}^{\mathsf{T}}$ into other systems

Control mechanisms

Allow to separate the unusual case from the normal case

- Domain failures: precondition fails
- Range failures: postcondition cannot be satisfied

For example, Lisp's `catch` and `throw`

- To evaluate `catch` α t we evaluate t
- If evaluation of t yields an actual result v
  $\Longrightarrow$ normal return: `catch` α t yields v
- If we encounter `throw` α s during the evaluation of t
  $\Longrightarrow$ exceptional return: `catch` α t yields s

Control mechanisms

Example

The ordinary list product function:

```
let rec listmult l = match l with
  | nil    -> 1
  | x :: k -> x * (listmult k)
```

With control:

```
let listmult l = catch α (listmult2 l)

let rec listmult2 l = match l with
  | nil    -> 1
  | 0 :: k -> throw α 0
  | x :: k -> x * (listmult2 k)
```

Simple type theory

Types:

$$\rho, \delta ::= \alpha \mid \rho \to \delta$$

Terms:

$$t, r, s ::= x \mid \lambda x : \rho.t \mid ts$$

Typing rules:

$$\frac{x : \rho \in \Gamma}{\Gamma \vdash x : \rho} \quad \frac{\Gamma, x : \rho \vdash t : \delta}{\Gamma \vdash \lambda x : \rho.t : \rho \to \delta} \quad \frac{\Gamma \vdash t : \rho \to \delta \qquad \Gamma \vdash s : \rho}{\Gamma \vdash ts : \delta}$$

Curry-Howard correspondence:

Simple type theory $\iff$ Minimal logic

Reduction:

$$(\lambda x : \rho.t)r \quad \to_\beta \quad t[x := r]$$

The $\lambda_{\mathcal{C}}$-calculus [Felleisen, Griffin]

- Simple type theory extended with:

$$\frac{\Gamma \vdash t : \bot}{\Gamma \vdash \mathcal{A}_\rho t : \rho} \qquad \frac{\Gamma \vdash t : \neg\neg\rho}{\Gamma \vdash \mathcal{C}_\rho t : \rho}$$

- Does originally not satisfy subject reduction

$$\dfrac{\dfrac{t : \bot}{\mathcal{A}_\rho t : \rho}}{E[\mathcal{A}_\rho t] : \delta} \quad \rhd_{\mathcal{A}} \quad t : \bot$$

- Some *ad hoc* modifications needed.
- Evaluation instead of reduction theory
  - Inconvenient for equational reasoning
  - If $t_1 \Rightarrow t_2$, then not necessarily $E[t_1] \Rightarrow E[t_2]$
  - Known reduction theories are unsatisfactory

The λ_Δ-calculus [Rehof, Sørensen]

- Simple type theory extended with:

$$\frac{\Gamma, x : \rho \to \bot \vdash t : \bot}{\Gamma \vdash \Delta x : \rho \to \bot . t : \rho}$$

- Reduction instead of evaluation
- Satisfies subject reduction, confluence, strong normalization
- Not able to simulate `throw` β (`throw` α s) $\to$ `throw` α s

The λ_μ-calculus [Parigot]

Terms and commands:

$$\begin{aligned} t, r, s &::= x \mid \lambda x : \rho . r \mid ts \mid \mu\alpha : \rho . c \\ c, d &::= [\alpha] t \end{aligned}$$

Two kinds of judgments:

$$\Gamma; \Delta \vdash c : \perp\!\!\!\perp \qquad \text{and} \qquad \Gamma; \Delta \vdash t : \rho$$

Two contexts:

$$\Gamma : \text{ assumptions} \qquad \text{and} \qquad \Delta : \text{ passivated goals}$$

The typing rules of simple type theory and the following rules:

$$\frac{\Gamma; \Delta, \alpha : \rho \vdash c : \perp\!\!\!\perp}{\Gamma; \Delta \vdash \mu\alpha : \rho . c : \rho} \qquad \frac{\Gamma; \Delta \vdash t : \rho \qquad \alpha : \rho \in \Delta}{\Gamma; \Delta \vdash [\alpha] t : \perp\!\!\!\perp}$$

The λ_μ-calculus

Minimal classical logic

Theorem

$\Gamma \vdash A$ in minimal classical logic $\implies$ $;\Gamma \vdash t : A$ in λ_μ.

Proof.

Peirce's law is inhabited:

$$\dfrac{\dfrac{t : (\rho \to \delta) \to \rho \qquad \dfrac{\dfrac{\dfrac{x : \rho}{[\alpha]x : \bot\!\!\!\bot}}{\mu\beta.[\alpha]x : \delta}}{\lambda x.\mu\beta.[\alpha]x : \rho \to \delta}}{\dfrac{t(\lambda x.\mu\beta.[\alpha]x) : \rho}{[\alpha]t(\lambda x.\mu\beta.[\alpha]x) : \bot\!\!\!\bot}}}{\mu\alpha.[\alpha]t(\lambda x.\mu\beta.[\alpha]x) : \rho}$$

□

The λ_μ-calculus

Classical logic

Theorem

$\Gamma; \Delta \vdash t : \rho$ in λ_μ $\implies$ $\Gamma, \neg\Delta \vdash \rho$ in classical logic.

$$\frac{\Gamma; \Delta, \alpha : \rho \vdash c : \perp\!\!\!\perp}{\Gamma; \Delta \vdash \mu\alpha : \rho.c : \rho} \qquad \frac{\Gamma; \Delta \vdash t : \rho \qquad \alpha : \rho \in \Delta}{\Gamma; \Delta \vdash [\alpha]t : \perp\!\!\!\perp}$$

$$\frac{\Gamma, \neg\Delta, \neg\rho \vdash \bot}{\Gamma, \neg\Delta \vdash \rho} \qquad \frac{\Gamma, \neg\Delta \vdash \rho \qquad \Gamma, \neg\Delta \vdash \neg\rho}{\Gamma, \neg\Delta \vdash \bot}$$

Theorem

$\Gamma; \vdash t : \rho$ in λ_μ $\implies$ $\Gamma \vdash \rho$ in minimal *classical logic.*

The λ_μ-calculus

Contexts:

$$E ::= \Box \mid Et$$

For example:

$$E = \Box\ z\ (\lambda xy.x) \quad \text{then} \quad E[t] \equiv t\ z\ (\lambda xy.x)$$

Structural substitution:

$$\begin{aligned}
x[\alpha := \beta E] &:= x \\
(\lambda x.r)[\alpha := \beta E] &:= \lambda x.r[\alpha := \beta E] \\
(ts)[\alpha := \beta E] &:= t[\alpha := \beta E]s[\alpha := \beta E] \\
(\mu\gamma.c)[\alpha := \beta E] &:= \mu\gamma.c[\alpha := \beta E] \\
([\gamma]t)[\alpha := \beta E] &:= [\gamma]t[\alpha := \beta E] \qquad \text{provided that } \gamma \neq \alpha \\
([\alpha]t)[\alpha := \beta E] &:= [\beta]E[t[\alpha := \beta E]]
\end{aligned}$$

For example:

$$([\alpha]\, x\, (\mu\beta.[\alpha]\, y))[\alpha := \gamma\ (\Box s)] \equiv [\gamma]\, x\, (\mu\beta.[\gamma]\, y\ s)\ s$$

The λ_μ-calculus

Reduction

Intuition:

$$\mu\alpha.[\beta]t \quad \sim \quad \text{catch } \alpha \text{ in } t \text{ and throw the result to } \beta$$

Another intuition:

$$(\mu\alpha.c)t_1 \ldots t_n \quad \sim \quad \text{apply } t_1 \ldots t_n \text{ to all subterms labeled } \alpha$$

Reduction:

$$\begin{array}{rcll}
(\lambda x.t)r & \to_\beta & t[x := r] & \\
(\mu\alpha.c)s & \to_{\mu R} & \mu\alpha.c[\alpha := \alpha\ (\Box s)] & \\
\mu\alpha.[\alpha]t & \to_{\mu\eta} & t & \text{provided that } \alpha \notin \mathrm{FCV}(t) \\
[\alpha]\mu\beta.c & \to_{\mu i} & c[\beta := \alpha\ \Box] &
\end{array}$$

We do not have $s(\mu\alpha.c) \quad \to \quad \mu\alpha.c[\alpha := \alpha\ (s\Box)]$

The λ_μ-calculus

Catch and throw

Define:

$$\texttt{catch}\ \alpha\ t := \mu\alpha.[\alpha]t$$
$$\texttt{throw}\ \beta\ s := \mu\gamma.[\beta]s \quad \text{provided that } \gamma \notin \text{FCV}(s) \cup \{\beta\}$$

We have the following reductions for `catch` and `throw`:

1. $E[\texttt{throw}\ \alpha\ t] \twoheadrightarrow \texttt{throw}\ \alpha\ t$
   $E[\mu\gamma.[\beta]s] \to \mu\gamma.[\beta]s[\gamma := \gamma\ E] \equiv \mu\gamma.[\beta]s$
2. $\texttt{catch}\ \alpha\ (\texttt{throw}\ \alpha\ t) \twoheadrightarrow \texttt{catch}\ \alpha\ t$
   Not $\texttt{catch}\ \alpha\ (\texttt{throw}\ \alpha\ t) \twoheadrightarrow t$
   $\mu\alpha.[\alpha]\mu\gamma.[\alpha]t \to \mu\alpha.[\alpha]t[\gamma := \alpha] \equiv \mu\alpha.[\alpha]t$
3. $\texttt{catch}\ \alpha\ t \twoheadrightarrow t$ provided that $\alpha \notin \text{FCV}(t)$
   $\mu\alpha.[\alpha]t \to t$
4. $\texttt{throw}\ \beta\ (\texttt{throw}\ \alpha\ s) \to \texttt{throw}\ \alpha\ s$
   $\mu\gamma.[\beta]\mu\delta.[\alpha]s \to \mu\gamma.[\alpha]s[\delta := \beta] \equiv \mu\gamma.[\alpha]s$

The λ_μ-calculus

Meta theoretical properties

- λ_μ satisfies subject reduction

$$\dfrac{\dfrac{c : \perp\!\!\!\perp}{\mu\alpha.c : \rho \to \delta} \qquad s : \rho}{(\mu\alpha.c)s : \delta} \quad \to_{\mu R} \quad \dfrac{c[\alpha := \alpha\ (\Box s)] : \perp\!\!\!\perp}{\mu\alpha.c[\alpha := \alpha\ (\Box s)] : \delta}$$

$$\dfrac{\dfrac{t : \rho}{[\alpha]t : \perp\!\!\!\perp}}{\mu\alpha.[\alpha]t : \rho} \quad \to_{\mu\eta} \quad t : \rho$$

$$\dfrac{\dfrac{c : \perp\!\!\!\perp}{\mu\beta.c : \rho}}{[\alpha]\mu\beta.c : \perp\!\!\!\perp} \quad \to_{\mu i} \quad c[\beta := \alpha\ \Box] : \perp\!\!\!\perp$$

- λ_μ is confluent
- λ_μ is strongly normalizing

Gödel's $\mathbf{T}$

Types:

$$\rho, \delta ::= \mathtt{N} \mid \rho \to \delta$$

Terms:

$$\begin{aligned} t, r, s ::= {} & x \mid \lambda x : \rho . r \mid ts \\ & \mid 0 \mid \mathtt{S}t \mid \mathtt{nrec}_\rho \; r \; s \; t \end{aligned}$$

The typing rules of simple type theory and the following rules:

$$\Gamma \vdash 0 : \mathtt{N} \qquad \frac{\Gamma \vdash t : \mathtt{N}}{\Gamma \vdash \mathtt{S}t : \mathtt{N}}$$

$$\frac{\Gamma \vdash r : \rho \qquad \Gamma \vdash s : \mathtt{N} \to \rho \to \rho \qquad \Gamma \vdash t : \mathtt{N}}{\Gamma \vdash \mathtt{nrec}_\rho \; r \; s \; t : \rho}$$

Gödel's **T**

Reduction:

$$\begin{array}{rcl} (\lambda x.t)r & \to_\beta & t[x := r] \\ \mathtt{nrec}\ r\ s\ 0 & \to_0 & r \\ \mathtt{nrec}\ r\ s\ (\mathtt{S}t) & \to_\mathtt{S} & s\ t\ (\mathtt{nrec}\ r\ s\ t) \end{array}$$

Expressive power:

- Simple type theory: extended polynomials
- λ_μ-calculus: extended polynomials (by CPS)
- Gödel's **T**: provably recursive functions
  For example:

$$\mathtt{pred} := \lambda z.\mathtt{nrec}\ 0\ (\lambda xy.x)\ z$$

The $\lambda_\mu^{\mathbf{T}}$-calculus

Terms:

$$
\begin{aligned}
t, r, s &::= x \mid \lambda x : \rho . r \mid ts \mid \mu\alpha : \rho . c \\
&\quad \mid 0 \mid \mathtt{S}t \mid \mathtt{nrec}_\rho \; r \; s \; t \\
c, d &::= [\alpha] t
\end{aligned}
$$

The typing rules of simple type theory and the following rules:

$$
\Gamma; \Delta \vdash 0 : \mathtt{N} \qquad \frac{\Gamma; \Delta \vdash t : \mathtt{N}}{\Gamma; \Delta \vdash \mathtt{S}t : \mathtt{N}}
$$

$$
\frac{\Gamma; \Delta \vdash r : \rho \qquad \Gamma; \Delta \vdash s : \mathtt{N} \to \rho \to \rho \qquad \Gamma; \Delta \vdash t : \mathtt{N}}{\Gamma; \Delta \vdash \mathtt{nrec}_\rho \; r \; s \; t : \rho}
$$

$$
\frac{\Gamma; \Delta, \alpha : \rho \vdash c : \underline{\perp\!\!\!\perp}}{\Gamma; \Delta \vdash \mu\alpha : \rho . c : \rho} \qquad \frac{\Gamma; \Delta \vdash t : \rho \qquad \alpha : \rho \in \Delta}{\Gamma; \Delta \vdash [\alpha] t : \underline{\perp\!\!\!\perp}}
$$

Reduction

Contexts:

$$E ::= \Box \mid Et \mid \mathtt{S}E \mid \mathtt{nrec}\ r\ s\ E$$

Reduction:

$$\begin{array}{rcll}
(\lambda x.t)r & \to_\beta & t[x := r] & \\
(\mu\alpha.c)s & \to_{\mu R} & \mu\alpha.c[\alpha := \alpha\ (\Box s)] & \\
\mu\alpha.[\alpha]t & \to_{\mu\eta} & t & \text{provided that } \alpha \notin \mathrm{FCV}(t) \\
[\alpha]\mu\beta.c & \to_{\mu i} & c[\beta := \alpha\ \Box] & \\
\mathtt{nrec}\ r\ s\ 0 & \to_0 & r & \\
\mathtt{nrec}\ r\ s\ (\mathtt{S}t) & \to_{\mathtt{S}} & s\ t\ (\mathtt{nrec}\ r\ s\ t) & \\
\mathtt{nrec}\ r\ s\ (\mu\alpha.c) & \to_{\mu\mathtt{N}} & \mu\alpha.c[\alpha := \alpha\ (\mathtt{nrec}\ r\ s\ \Box)] &
\end{array}$$

Closed normal forms of type $\mathtt{N}$ are not necessarily numerals:

$$\mathtt{catch}\ \alpha\ \mathsf{S}(\mathtt{throw}\ \alpha\ 0) \equiv \mu\alpha.[\alpha]\,\mathtt{S}(\mu\beta.[\alpha]0)$$

Reduction

$$
\begin{array}{rcl}
(\lambda x.t)r & \rightarrow_{\beta} & t[x := r] \\
\mathtt{S}(\mu\alpha.c) & \rightarrow_{\mu\mathtt{S}} & \mu\alpha.c[\alpha := \alpha\ (\mathtt{S}\Box)] \\
(\mu\alpha.c)s & \rightarrow_{\mu R} & \mu\alpha.c[\alpha := \alpha\ (\Box s)] \\
\mu\alpha.[\alpha]t & \rightarrow_{\mu\eta} & t \qquad \text{provided that } \alpha \notin \mathrm{FCV}(t) \\
[\alpha]\mu\beta.c & \rightarrow_{\mu i} & c[\beta := \alpha\ \Box] \\
\mathtt{nrec}\ r\ s\ 0 & \rightarrow_{0} & r \\
\mathtt{nrec}\ r\ s\ (\mathtt{S}t) & \rightarrow_{\mathtt{S}} & s\ t\ (\mathtt{nrec}\ r\ s\ t) \\
\mathtt{nrec}\ r\ s\ (\mu\alpha.c) & \rightarrow_{\mu\mathtt{N}} & \mu\alpha.c[\alpha := \alpha\ (\mathtt{nrec}\ r\ s\ \Box)]
\end{array}
$$

Not confluent. For example $\mu\alpha.[\alpha]\mathtt{nrec}\ 0\ (\lambda xh.K)\ (\mathtt{S}\mu\gamma.[\alpha]L)$ reduces to:

$$\mu\alpha.[\alpha](\lambda xh.K)\ (\mu\gamma.[\alpha]L)\ (\mathtt{nrec}\ \ldots) \rightarrow \mu\alpha.[\alpha]K \rightarrow K$$

$$\mu\alpha.[\alpha]\mathtt{nrec}\ 0\ (\lambda xh.K)\ (\mu\gamma.[\alpha]L) \rightarrow \mu\alpha.[\alpha]\mu\gamma.[\alpha]L \rightarrow \mu\alpha.[\alpha]L \rightarrow L$$

Reduction

$$
\begin{array}{rcl}
(\lambda x.t)r & \to_\beta & t[x := r] \\
\mathtt{S}(\mu\alpha.c) & \to_{\mu\mathtt{S}} & \mu\alpha.c[\alpha := \alpha\ (\mathtt{S}\Box)] \\
(\mu\alpha.c)s & \to_{\mu R} & \mu\alpha.c[\alpha := \alpha\ (\Box s)] \\
\mu\alpha.[\alpha]t & \to_{\mu\eta} & t \qquad \text{provided that } \alpha \notin \mathrm{FCV}(t) \\
[\alpha]\mu\beta.c & \to_{\mu i} & c[\beta := \alpha\ \Box] \\
\mathtt{nrec}\ r\ s\ 0 & \to_0 & r \\
\mathtt{nrec}\ r\ s\ (\mathtt{S}\underline{n}) & \to_{\mathtt{S}} & s\ \underline{n}\ (\mathtt{nrec}\ r\ s\ \underline{n}) \\
\mathtt{nrec}\ r\ s\ (\mu\alpha.c) & \to_{\mu\mathtt{N}} & \mu\alpha.c[\alpha := \alpha\ (\mathtt{nrec}\ r\ s\ \Box)]
\end{array}
$$

However, now the $\to_{\mathtt{S}}$-rule looks more like call-by-value?

Meta theoretical properties

Theorem

1. λ_μ^{T} *satisfies a normal form theorem*
   If $;\vdash t : \mathtt{N}$ *and* t *in normal form, then* $t \equiv \underline{n}$.
2. λ_μ^{T} *satisfies subject reduction*
   If $\Gamma;\Delta \vdash t_1 : \rho$ *and* $t_1 \twoheadrightarrow t_2$*, then* $\Gamma;\Delta \vdash t_2 : \rho$.
3. λ_μ^{T} *is confluent*
   If $t_1 \twoheadrightarrow t_2$ *and* $t_1 \twoheadrightarrow t_3$*, then* $t_2 \twoheadrightarrow t_4$ *and* $t_3 \twoheadrightarrow t_4$.
4. λ_μ^{T} *is strongly normalizing*
   If $\Gamma;\Delta \vdash t : \rho$*, then each reduction sequence starting from* t *is finite.*
5. *The functions definable in* λ_μ^{T} *are exactly the ones that are provably recursive.*

Confluence

Usual approach [Tait, Martin-Löf]:

1. Define a parallel reduction $\Rightarrow$
2. Prove that $\Rightarrow$ is confluent
3. Prove that $t_1 \to t_2$ implies $t_1 \Rightarrow t_2$
4. Prove that $t_1 \Rightarrow t_2$ implies $t_1 \twoheadrightarrow t_2$

For the ordinary λ-calculus:

1. $x \Rightarrow x$
2. If $t \Rightarrow t'$, then $\lambda x.t \Rightarrow \lambda x.t'$
3. If $t \Rightarrow t'$ and $r \Rightarrow r'$, then $tr \Rightarrow t'r'$
4. If $t \Rightarrow t'$ and $r \Rightarrow r'$, then $(\lambda x.t)r \Rightarrow t'[x := r']$

For a more streamlined proof [Takahashi]:

- Define $t^\diamond$ such that if $t_1 \Rightarrow t_2$, then $t_2 \Rightarrow t_1^\diamond$

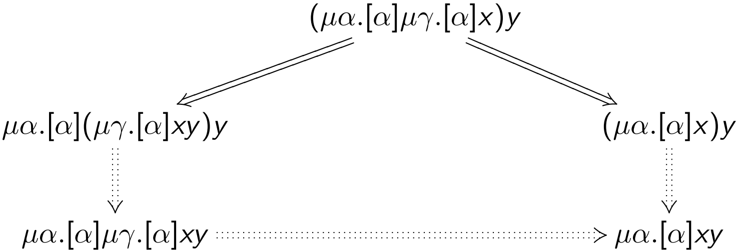
Confluence

The straightforward extension to λ_μ:

1. $x \Rightarrow x$
2. If $t \Rightarrow t'$, then $\lambda x.t \Rightarrow \lambda x.t'$
3. If $t \Rightarrow t'$ and $r \Rightarrow r'$, then $tr \Rightarrow t'r'$
4. If $t \Rightarrow t'$ and $r \Rightarrow r'$, then $(\lambda x.t)r \Rightarrow t'[x := r']$
5. If $c \Rightarrow c'$, then $\mu\alpha.c \Rightarrow \mu\alpha.c'$
6. If $c \Rightarrow c'$ and $s \Rightarrow s'$, then $(\mu\alpha.c)s \Rightarrow \mu\alpha.c'[\alpha := \alpha\ (\Box s')]$
7. If $t \Rightarrow t'$ and $\alpha \notin \mathrm{FCV}(t)$, then $\mu\alpha.[\alpha]t \Rightarrow t'$
8. If $t \Rightarrow t'$, then $[\alpha]t \Rightarrow [\alpha]t'$.
9. If $c \Rightarrow c'$, then $[\alpha]\mu\beta.c \Rightarrow c'[\beta := \alpha\Box]$

Confluence

This is not confluent. For example:



Here:

$$[\alpha]\mu\gamma.[\alpha]x \Rightarrow [\alpha]x$$

Confluence

Baba, Hirokawa and Fujita's approach:

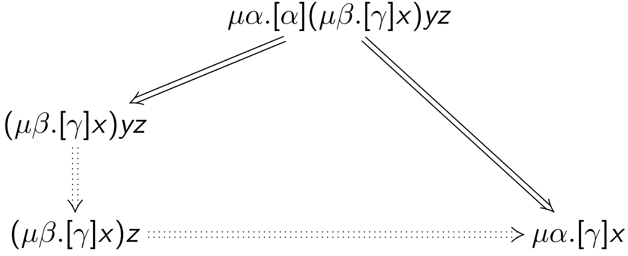
1. $x \Rightarrow x$
2. If $t \Rightarrow t'$, then $\lambda x.t \Rightarrow \lambda x.t'$
3. If $t \Rightarrow t'$ and $r \Rightarrow r'$, then $tr \Rightarrow t'r'$
4. If $t \Rightarrow t'$ and $r \Rightarrow r'$, then $(\lambda x.t)r \Rightarrow t'[x := r']$
5. If $c \Rightarrow c'$, then $\mu\alpha.c \Rightarrow \mu\alpha.c'$
6. If $c \Rightarrow c'$ and $s \Rightarrow s'$, then $(\mu\alpha.c)s \Rightarrow \mu\alpha.c'[\alpha := \alpha\ (\Box s')]$
7. If $t \Rightarrow t'$ and $\alpha \notin \mathrm{FCV}(t)$, then $\mu\alpha.[\alpha]t \Rightarrow t'$
8. If $t \Rightarrow t'$, then $[\alpha]t \Rightarrow [\alpha]t'$
9. If $c \Rightarrow c'$ and $E \Rightarrow E'$, then $[\alpha]E[\mu\beta.c] \Rightarrow c'[\beta := \alpha E']$

Confluence

This is not confluent if we include the rule:

7. If $t \Rightarrow t'$ and $\alpha \notin \mathsf{FCV}(t)$, then $\mu\alpha.[\alpha]t \Rightarrow t'$

For example:



Here:

$$[\alpha](\mu\beta.[\gamma]x)yz \Rightarrow [\gamma]x$$

Confluence

1. $x \Rightarrow x$
2. If $t \Rightarrow t'$, then $\lambda x.t \Rightarrow \lambda x.t'$
3. If $t \Rightarrow t'$ and $r \Rightarrow r'$, then $tr \Rightarrow t'r'$
4. If $t \Rightarrow t'$ and $r \Rightarrow r'$, then $(\lambda x.t)r \Rightarrow t'[x := r']$
5. If $c \Rightarrow c'$, then $\mu\alpha.c \Rightarrow \mu\alpha.c'$
6. If $c \Rightarrow c'$ and $E \Rightarrow E'$, then $E[\mu\alpha.c] \Rightarrow \mu\alpha.c'[\alpha := \alpha E']$
7. If $t \Rightarrow t'$ and $\alpha \notin \mathsf{FCV}(t)$, then $\mu\alpha.[\alpha]t \Rightarrow t'$
8. If $t \Rightarrow t'$, then $[\alpha]t \Rightarrow [\alpha]t'$
9. If $c \Rightarrow c'$ and $E \Rightarrow E'$, then $[\alpha]E[\mu]\beta.c \Rightarrow c'[\beta := \alpha E']$

Confluence

- This notion of reduction is very strong

$$E_n[\mu\alpha_n.[\alpha_n] \dots E_1[\mu\alpha_1.[\alpha_1] E_0[\mu\alpha_0.c]] \dots] \Rightarrow c'$$

- Non-trivial definition of complete development
- Non-trivial confluence proof
- Proven for $\lambda_\mu^{\mathbf{T}}$ in my thesis

Conclusions - Further research

- Other data types (lists, ...)
- Call-by-value instead of call-by-name
- Primitive `catch` and `throw` [Herbelin 2010, for example]
- Closer to implementations?
- Program extraction