Parigot's Second Order $\lambda\mu$-Calculus and Inductive Types

Ralph Matthes, Computer Science Institute, University of Munich

presented on May 4, 2001 at TLCA '01
Jagiellonian University, Kraków, Poland

matthes@informatik.uni-muenchen.de

Abstract

A new proof of strong normalization of Parigot's (second order) $\lambda\mu$-calculus is given by a reduction-preserving embedding into system F (second order polymorphic λ-calculus). The main idea is to use the least stable supertype for any type. These non-strictly positive inductive types and their associated iteration principle are available in system F, and allow to give a translation vaguely related to CPS translations (corresponding to the Kolmogorov embedding of classical logic into intuitionistic logic). However, they simulate Parigot's μ-reductions whereas CPS translations hide them.

As a major advantage, this embedding does not use the idea of reducing stability ($\neg\neg\phi \to \phi$) to that for atomic formulae. Therefore, it even extends to non-interleaving positive fixed-point types. As a non-trivial application, strong normalization of $\lambda\mu$-calculus, extended by primitive recursion on monotone inductive types, is established.

System F as a basis for Parigot's $\lambda\mu$-calculus

System F is second-order polymorphic λ-calculus.

Types of F:

- type variables, denoted by α, β, . . .
- function types $\rho \to \sigma$
- universal types $\forall\alpha\,\rho$

Typed terms of F:

- typed variables, denoted e. g. by x^ρ
- lambda abstraction $(\lambda x^\rho r^\sigma)^{\rho\to\sigma}$ for terms
- term application $(r^{\rho\to\sigma} s^\rho)^\sigma$
- lambda abstraction $(\Lambda\alpha r^\rho)^{\forall\alpha\rho}$ for types (under the usual proviso)
- type application $(r^{\forall\alpha\rho}\sigma)^{\rho[\alpha:=\sigma]}$

β-reduction

for system F is given as usual by

$$
\begin{array}{ll}
(\beta_{\rightarrow}) & (\lambda x^{\rho} r)s \vartriangleright r[x^{\rho} := s] \\
(\beta_{\forall}) & (\Lambda \alpha r)\sigma \vartriangleright r[\alpha := \sigma]
\end{array}
$$

The reduction relation $\rightarrow$ is defined as the term closure of $\vartriangleright$.
Well-known: $\rightarrow$ is strongly normalizing, i. e., there are no infinite reduction sequences $r_1 \rightarrow r_2 \rightarrow r_3 \rightarrow \cdots$.

Parigot's $\lambda\mu$-calculus

Extension of system F by "reductio ad absurdum" and by corresponding rewrite rules.

The new term rule:

$$\mu a^\sigma.[b^\rho]r^\rho : \sigma$$

with μ-variables a and b (the "normal" variables are called λ-variables).

Interpretation: the μ-variable b^ρ assumes $\neg\rho$, $[b^\rho]r^\rho$ is an application not counted as a term but as a "named term". Morally, it has type $\perp$. But $\perp$ is no type in the system. The μ-variable a^σ (possible appearing free in [b]r) is bound by μ, hence representing a derivation of $\perp$ from $\neg\sigma$. The principle of *reductio ad absurdum*—to be expressed by μ-abstraction—yields σ.

Example — Peirce's Law

$((\alpha \to \beta) \to \alpha) \to \alpha$ is inhabited (even) for $\alpha \neq \beta$.

Proof:

$$\lambda y^{(\alpha\to\beta)\to\alpha}\mu a^{\alpha}.[a]\,\underbrace{(y(\lambda x^{\alpha}\,\underbrace{\mu b^{\beta}.[a]x}_{:\beta}))}_{:\alpha} : ((\alpha \to \beta) \to \alpha) \to \alpha$$

Note: The μ-variables have atomic types in this example. Composite types in the use of *reductio ad absurdum* may lead to detours in proofs and are removed by μ-reductions to be defined next.

Reduction rules of $\lambda\mu$

$\triangleright$ of system F is extended by μ-reductions:

$$
\begin{array}{ll}
(\mu_\to) & (\mu a^{\rho\to\sigma}.r)s \triangleright \mu b^\sigma.r\Big[x^{\rho\to\sigma}.[a^{\rho\to\sigma}]x := [b](xs)\Big] \\
(\mu_\forall) & (\mu a^{\forall\alpha\rho}.r)\sigma \triangleright \mu b^{\rho[\alpha:=\sigma]}.r\Big[x^{\forall\alpha\rho}.[a^{\forall\alpha\rho}]x := [b](x\sigma)\Big]
\end{array}
$$

with r always a named term.

$r\Big[x^\rho.[a^\rho]x := [b^\sigma]t^\sigma\Big]$ denotes the result of replacing inductively (bottom-up) in r each subexpression of the form $[a^\rho]u^\rho$ by $[b^\sigma](t[x^\rho := u])$ for any term u of type ρ.

Most important case:

$$([a^\rho]r^\rho)\Big[x^\rho.[a^\rho]x := [b^\sigma]t^\sigma\Big] := [b^\sigma]t\Big[x^\rho := r\Big[x^\rho.[a^\rho]x := [b^\sigma]t^\sigma\Big]\Big]$$

Aim: embed $\lambda\mu$ into F

A type-respecting reduction-preserving embedding (embedding for short) of a typed term rewrite system $\mathcal{S}$ into a typed term rewrite system $\mathcal{S}'$ is a function $-'$ (the $-$ sign represents the indefinite argument of the function $'$) which assigns to every type ρ of $\mathcal{S}$ a type ρ' of $\mathcal{S}'$ and to every term r of type ρ of $\mathcal{S}$ a term r' of the (image) type ρ' of $\mathcal{S}'$ such that the following implication holds:

If $r \to s$ in $\mathcal{S}$, then $r' \to^+ s'$ in $\mathcal{S}'$.

($\to^+$ denotes the transitive closure of $\to$.)

Obviously, if there is an embedding of $\mathcal{S}$ into $\mathcal{S}'$, then strong normalization of $\mathcal{S}'$ is inherited by $\mathcal{S}$.

Method: Introduce a system $\mathsf{F}^\sharp$ and embed $\lambda\mu$ into $\mathsf{F}^\sharp$ and $\mathsf{F}^\sharp$ into $\mathsf{F}^\perp$ (which is F with the type constant $\perp$).

$\mathsf{F}^\sharp$—$\mathsf{F}^\perp$ with iteration on stabilization

If $\neg\neg\rho \to \rho$ is provable, then ρ is called stable. We now add a least stable supertype $\sharp\rho$ for any type ρ: Assume constants

$$I_{\sharp\rho} : \rho \to \sharp\rho \quad \text{and} \quad S_{\sharp\rho} : \neg\neg\sharp\rho \to \sharp\rho$$

and add an elimination rule:

$$\frac{r : \sharp\rho \qquad s_1 : \rho \to \tau \qquad s_2 : \neg\neg\tau \to \tau}{r E_\tau s_1 s_2 : \tau}$$

Extend $\triangleright$ of system F by

$$\begin{array}{ll} (\sharp I) & (I_{\sharp\rho} r) E_\tau s_1 s_2 \triangleright s_1 r \\ (\sharp S) & (S_{\sharp\rho} r) E_\tau s_1 s_2 \triangleright s_2 \Big(\lambda y^{\neg\tau}.r(\lambda z^{\sharp\rho}.y(z E_\tau s_1 s_2)) \Big) \end{array}$$

Recall that one may lift function types $\sigma \to \tau$ up through $\lambda\alpha.\neg\neg\alpha$ via

$$\lambda f^{\sigma\to\tau} \lambda x^{\neg\neg\sigma} \lambda y^{\neg\tau}.x(\lambda z^\sigma.y(f z)) : (\sigma \to \tau) \to \neg\neg\sigma \to \neg\neg\tau$$

$\mathsf{F}^\sharp$ embeds into $\mathsf{F}^\perp$

By the very description, $\sharp\rho$ is nothing but the non-strictly positive inductive type $\mu\alpha.\rho + \neg\neg\alpha$ with $\alpha \notin \mathrm{FV}(\rho)$, formulated without the sum type. Its canonical polymorphic encoding would be

$$\forall\alpha.((\rho + \neg\neg\alpha) \to \alpha) \to \alpha.$$

We simplify this to

$$\forall\alpha.(\rho \to \alpha) \to (\neg\neg\alpha \to \alpha) \to \alpha.$$

This gives the only non-trivial clause for the sought translation of types:

$$(\sharp\rho)' := \forall\alpha.(\rho' \to \alpha) \to (\neg\neg\alpha \to \alpha) \to \alpha$$

$\mathsf{F}^\sharp$ embeds into $\mathsf{F}^\perp$ (cont'd)

The term translation:

$$\begin{aligned}
(I_{\sharp\rho})' &:= \lambda x^{\rho'} \Lambda\alpha \lambda x_1^{\rho' \to \alpha} \lambda x_2^{\neg\neg\alpha\to\alpha}.x_1 x \\
(S_{\sharp\rho})' &:= \lambda x^{\neg\neg(\sharp\rho)'} \Lambda\alpha \lambda x_1^{\rho' \to \alpha} \lambda x_2^{\neg\neg\alpha\to\alpha}.x_2 \Big(\lambda y^{\neg\alpha}.x(\lambda z^{(\sharp\rho)'}.y(z\alpha x_1 x_2))\Big) \\
(rE_\tau s_1 s_2)' &:= r'\tau' s_1' s_2'
\end{aligned}$$

Easy calculation shows

$$((I_{\sharp\rho} r)E_\tau s_1 s_2)' \to^4 (s_1 r)' \qquad \text{and}$$

$$\Big((S_{\sharp\rho} r)E_\tau s_1 s_2\Big)' \to^4 \Big(s_2\Big(\lambda y^{\neg\tau}.r(\lambda z^{\sharp\rho}.y(zE_\tau s_1 s_2))\Big)\Big)'.$$

$\lambda\mu$ embeds into $\mathsf{F}^\sharp$

Define for ρ in $\lambda\mu$ a type ρ^* in $\mathsf{F}^\sharp$:

$$\begin{aligned}\alpha^* &:= \alpha\\ (\rho\to\sigma)^* &:= \sharp\rho^*\to\sharp\sigma^*\\ (\forall\alpha\rho)^* &:= \forall\alpha\,\sharp\rho^*\end{aligned}$$

Clearly, $(\rho[\alpha:=\sigma])^* = \rho^*[\alpha:=\sigma^*]$.

The type translation of the embedding is given by $\rho' := \sharp\rho^*$. Therefore,

$$\begin{aligned}\alpha' &= \sharp\alpha\\ (\rho\to\sigma)' &= \sharp(\rho'\to\sigma')\\ (\forall\alpha\rho)' &= \sharp(\forall\alpha\rho')\end{aligned}$$

and $(\rho[\alpha:=\sigma])' = \rho'[\alpha:=\sigma^*]$.

Note that a Kolmogorov translation would have had $\neg\neg\tau$ instead of $\sharp\tau$ in the clauses.

$\lambda\mu$ embeds into $\mathsf{F}^\sharp$ (cont'd)

Term translation:

$$
\begin{aligned}
(x^\rho)' &:= x^{\rho'}\\
(\lambda x^\rho r^\sigma)' &:= I_{(\rho\to\sigma)'}(\lambda x^{\rho'} r')\\
(r^{\rho\to\sigma} s^\rho)' &:= r' E_{\sigma'}(\lambda z^{\rho'\to\sigma'}.z s') S_{\sigma'}\\
(\Lambda\alpha r^\rho)' &:= I_{(\forall\alpha\rho)'}(\Lambda\alpha r')\\
(r^{\forall\alpha\rho}\sigma)' &:= r' E_{(\rho[\alpha:=\sigma])'}(\lambda z^{\forall\alpha\rho'}.z\sigma^*) S_{(\rho[\alpha:=\sigma])'}\\
(\mu a^\sigma.[b^\rho]r^\rho)' &:= S_{\sigma'}(\lambda a^{\neg\sigma'}.b^{\neg\rho'} r')
\end{aligned}
$$

Notice that free μ-variables a^σ are translated into $a^{\neg\sigma'}$. Hence, we assume that a is a variable name in $\mathsf{F}^\sharp$.

We see that the λ-abstractions are always "injected" into the I part of the stabilization types and that the μ-abstraction is "injected" into the S part of it.

$\lambda\mu$ embeds into $\mathsf{F}^\sharp$ (final part)

Lemma: $(r[x^\rho := s])' = r'[x^{\rho'} := s']$ and $(r[\alpha := \rho])' = r'[\alpha := \rho^*]$.

Let $\to^*$ denote the reflexive transitive closure of $\to$.

Main Lemma: $r'[a^{\neg\rho'} := \lambda x^{\rho'}.b^{\neg\sigma'}t'] \to^* \big(r[x^\rho.[a^\rho]x := [b^\sigma]t]\big)'$ for t of type σ.

Theorem: $-'$ is an embedding of $\lambda\mu$ into $\mathsf{F}^\sharp$.

Main tool for the proof is a uniform notation for eliminations: Write R for a term or a type. Define the term or type $\hat{R}$ as follows: $\hat{r} := r'$ and $\hat{\rho} := \rho^*$.

The μ-reduction rules both become (if types are omitted):

$$(\mu a.r)R \rhd \mu b.r[x.[a]x := [b](xR)].$$

We also have uniformly $(rR)' = r'E(\lambda z.z\hat{R})S$.

Extension to Fixed-Point Types

We extend every system by non-interleaving positive fixed-point types and can extend the embeddings as well.

Fixed-point types $f\alpha\rho$ are supposed to describe arbitrary fixed-points of $\lambda\alpha\rho$, i. e., of the operation $\sigma \mapsto \rho[\alpha := \sigma]$. We confine ourselves to types ρ which are (non-strictly) positively dependent on α, i. e., $f\alpha\rho$ may only be formed when every occurrence of α in ρ is "to the left of an even number of $\rightarrow$". We also forbid fixed-point formation w. r. t. type variables which are free parameters of already formed fixed-point types. Hence, we allow $f\alpha.1 + \neg\neg\alpha$ and $f\alpha.1 + ((f\beta.1 + \beta) \rightarrow \alpha)$, but we disallow $f\alpha f\beta.1 + \alpha \times \beta$ (with $+$ and $\times$ encoded impredicatively). For the allowed $f\alpha\rho$, we assume term rules and a new reduction rule:

$$\frac{t : \rho[\alpha := f\alpha\rho]}{C_{f\alpha\rho}t : f\alpha\rho} \qquad \frac{r : f\alpha\rho}{rE_f : \rho[\alpha := f\alpha\rho]}$$

$$(\beta_f) \qquad (C_{f\alpha\rho}t)E_f \triangleright t$$

$\lambda\mu$ with fixed-point types

We now also consider E_f as a possible value of R. Following our standard pattern

$$(\mu a.r)R \triangleright \mu b.r[x.[a]x := [b](xR)]$$

for μ-reductions, we consider

$$(\mu_f) \qquad (\mu a^{f\alpha\rho}.r)E_f \triangleright \mu b^{\rho[\alpha:=f\alpha\rho]}.r\Big[x^{f\alpha\rho}.[a^{f\alpha\rho}]x := [b](xE_f)\Big]$$

$\lambda\mu$ with fixed-point types ($\lambda\mu^f$) can be embedded into $\mathsf{F}^\sharp$ with fixed-point types ($\mathsf{F}^{\sharp f}$) by a straightforward extension of $-^*$: $(f\alpha\rho)^* := f\alpha\sharp\rho^*$. The term rules

$$\begin{aligned}(C_{f\alpha\rho}t^{\rho[\alpha:=f\alpha\rho]})' &:= I_{(f\alpha\rho)'}(C_{f\alpha\rho'}t') \\ (r^{f\alpha\rho}E_f)' &:= r'E_{(\rho[\alpha:=f\alpha\rho])'}(\lambda z^{f\alpha\rho'}.zE_f)S_{(\rho[\alpha:=f\alpha\rho])'}\end{aligned}$$

respect types since $(\rho[\alpha := f\alpha\rho])' = \rho'[\alpha := f\alpha\rho']$ fortunately holds.

$\lambda\mu$ with monotone inductive types

Inductive types are a syntactic representation of least pre-fixed-points of operations $\sigma \mapsto \rho[\alpha := \sigma]$. Here, we do not impose any positivity restriction on α's occurrences in ρ but assume the existence of a monotonicity witness, i. e., a term of type

$$\forall\alpha\forall\beta.(\alpha \to \beta) \to \rho \to \rho[\alpha := \beta]$$

in the introduction rule. Hence, add arbitrary types $\mu\alpha\rho$ and, as term rules:

$$\frac{m : \forall\alpha\forall\beta.(\alpha \to \beta) \to \rho \to \rho[\alpha := \beta] \qquad t : \rho[\alpha := \mu\alpha\rho]}{C_{\mu\alpha\rho} m t : \mu\alpha\rho}$$

$$\frac{r : \mu\alpha\rho \qquad s : \rho[\alpha := \mu\alpha\rho \times \sigma] \to \sigma}{r E_\mu \sigma s : \sigma}$$

Primitive recursion is added as rewrite rule

$$(\beta_\mu) \qquad (C_{\mu\alpha\rho} m t) E_\mu \sigma s \rhd s\Big(m(\mu\alpha\rho)(\mu\alpha\rho \times \sigma)\Big(\lambda x^{\mu\alpha\rho}.\langle x, (\lambda x^{\mu\alpha\rho}.x E_\mu \sigma s) x\rangle\Big) t\Big)$$

$\lambda\mu$ with monotone inductive types (cont'd)

The μ-reduction rule (μ in the sense of Parigot) follows the standard pattern if we now even allow R to be $E_\mu \sigma s$:

$$(\mu_\mu) \qquad (\mu a^{\mu\alpha\rho}.r)E_\mu\sigma s \rhd \mu b^\sigma.r\Big[x^{\mu\alpha\rho}.[a^{\mu\alpha\rho}]x := [b](xE_\mu\sigma s)\Big]$$

Theorem: There is an embedding of this system into $\lambda\mu^f$, hence $\lambda\mu$ with primitive recursion on monotone inductive types is strongly normalizing.

Idea: Produce an embedding of F with monotone inductive types into F with fixed-point types and show that it is compatible with μ-reductions.

Remark: The concept of stabilization types is completely useless for this purpose since we cannot get $(\rho[\alpha := \mu\alpha\rho \times \sigma])' = \rho'[\alpha := \mu\alpha\rho' \times \sigma']$ with a reasonable extension of the stabilizing translation of types of $\lambda\mu$.

Monotone inductive types into positive fixed-point types

The crucial clause for the types is

$$(\mu\alpha\rho)' := f\beta\forall\gamma.\Big(\Big(\forall\alpha.(\beta\times\gamma\to\alpha)\to\rho'\Big)\to\gamma\Big)\to\gamma\ .$$

The interesting term clauses:

$$(C_{\mu\alpha\rho}mt)' := C_{(\mu\alpha\rho)'}\Big(\Lambda\gamma\lambda z^{(\forall\alpha.((\mu\alpha\rho)'\times\gamma\to\alpha)\to\rho')\to\gamma}.z\Big(\Lambda\alpha\lambda u^{(\mu\alpha\rho)'\times\gamma\to\alpha}.$$

$$m'(\mu\alpha\rho)'\alpha\Big(\lambda x^{(\mu\alpha\rho)'}.u\langle x,(\lambda x^{(\mu\alpha\rho)'}.xE_f\gamma z)x\rangle\Big)t'\Big)\Big)$$

and

$$(rE_\mu\sigma s)' := r'E_f\sigma'\Big(\lambda z^{\forall\alpha.((\mu\alpha\rho)'\times\sigma'\to\alpha)\to\rho'}.s'\Big(z((\mu\alpha\rho)'\times\sigma')(\lambda x^{(\mu\alpha\rho)'\times\sigma'}x)\Big)\Big)$$

Conclusions, future work

Showed an alternative to the Kolmogorov translation of classical logic into minimal logic which simplifies proofs of normalization for a classical version of λ-calculus (Parigot's $\lambda\mu$).

Stabilization illustrates what can be achieved with non-strictly positive inductive types.

The logical reading of the main theorem gives consistency for classical second-order propositional logic with "extended induction" on monotone inductive propositions (where extended induction is given by reading the typing rule for E_μ as an inference rule of natural deduction).

Operational behaviour of the embedding should be studied and compared with CPS translation stemming from Kolmogorov translation.

How can sums with permutative conversions be treated in an analogous fashion (cf. de Groote's contribution just before lunch)?

Finis