Reducibility and $\top\top$-lifting for Computation Types

Ian Stark and Sam Lindley

Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh

Séminaire PPS
Mardi 17 mai

http://www.ed.ac.uk/~stark/reducibility.html

Overview

Summary

We present $\top\top$-lifting: an operational technique to define and prove properties of terms of Moggi's monadic computation types.

Demonstrate application to Girard-Tait reducibility, with a proof of strong normalisation for the computational metalanguage.

Talk outline

- The computational metalanguage λ_{ml}
- $\top\top$-lifting for reducibility $\implies$ proof of strong normalisation
- Robustness: extension to sum types and exceptions

The computational metalanguage λ_{ml}

Moggi's computational metalanguage λ_{ml}: how to capture effectful computation within a pure typed lambda-calculus.

Computation types

For each type A of values there is a type TA of programs that compute a value of type A

Sample computational effects:

Non-termination, exceptions, input/output, state, non-deterministic choice, jumps, ...

The computational metalanguage λ_{ml}

Moggi's computational metalanguage λ_{ml}: how to capture effectful computation within a pure typed lambda-calculus.

Computation types

For each type A of values there is a type TA of programs that compute a value of type A

Sample computational effects:

Non-termination $TA = A_\perp$, exceptions $TA = A+E$, input/output $TA = \mu X.(A+O\times X+X^I)$, state $TA = (S\times A)^S$, non-deterministic choice $TA = \mathcal{P}(A)$, jumps $TA = R^{R^A}$, ...

The computational metalanguage λ_{ml}

Moggi's computational metalanguage λ_{ml}: how to capture effectful computation within a pure typed lambda-calculus.

Computation types

For each type A of values there is a type $\mathsf{T}A$ of programs that compute a value of type A

Sample computational effects:

Non-termination, exceptions, input/output, state, non-deterministic choice, jumps, ...

Types and terms of λ_{ml}

Types	A, B	$::=$	ι	Ground values
		$\mid$	$A \to B$	Functions
		$\mid$	$A \times B$	Products
		$\mid$	TA	Computations

$$L, M, N, P \quad ::= \quad x^A \mid \lambda x^A.M \mid MN \mid \langle M, N \rangle \mid fst(M) \mid snd(M) \mid [M] \mid \mathrm{let}\, x^A \Leftarrow M \,\mathrm{in}\, N$$

Terms

Typing

$$\frac{M : A}{[M] : TA} \qquad \frac{M : TA \quad N : TB}{\mathrm{let}\, x^A \Leftarrow M \,\mathrm{in}\, N : TB}$$

Lift value to computation

Compute M, bind result to x, compute N

Applications of λ_{ml}

For example. . .

- Denotational semantics: extend pure models uniformly to handle computational effects.
- Haskell: monads for mixing functional and effectful code, programming interactions with the real world.
- Compilers: MLj and SML.NET use a monadic intermediate language to carry out type-preserving compilation.

Generic vs. concrete

Different applications may use λ_{ml} *generically* (any T), or *concretely* (fixed T for specific computational features).

We look at strong normalisation for generic λ_{ml}.

Reductions for λ_{ml}

Standard $\beta\eta$ for functions and products, and for computations:

$$\begin{array}{lr} \mathsf{T}.\beta & \mathsf{let}\, x \Leftarrow [N]\, \mathsf{in}\, M \longrightarrow M[x := N] \\ \mathsf{T}.\eta & \mathsf{let}\, x \Leftarrow M\, \mathsf{in}\, [x] \longrightarrow M \\ \mathsf{T}.assoc & \mathsf{let}\, y \Leftarrow (\mathsf{let}\, x \Leftarrow L\, \mathsf{in}\, M)\, \mathsf{in}\, N \\ & \longrightarrow \mathsf{let}\, x \Leftarrow L\, \mathsf{in}\, (\mathsf{let}\, y \Leftarrow M\, \mathsf{in}\, N) \end{array}$$

Theorem (To prove)

λ_{ml} is strongly normalising: no term $M \in \lambda_{ml}$ has an infinite reduction sequence $M \to M_1 \to \cdots$

Reducibility

Straightforward induction on term structure fails to prove strong normalisation. Standard step: use an auxiliary reducibility predicate.

- Define $\mathit{red}_A \subseteq A$ by induction on structure of type A.
- Show useful properties of red_A by induction on A; in particular that all elements are strongly normalising: $\forall M \in \mathit{red}_A \,.\, M\!\downarrow$
- Show all M are in red_A, by induction on structure of term M.

Roughly, reducibility will be the logical predicate induced by SN at ground type

Reducibility for $\lambda_{\beta\eta}$

Standard reducibility for ground, function and product types:

Definition (Reducibility, begun)

$$\begin{aligned}
\mathit{red}_\iota &= \{\, M : \iota \mid M{\downarrow} \,\} \\
\mathit{red}_{A \to B} &= \{\, F : A \to B \mid \forall M \in \mathit{red}_A \,.\, FM \in \mathit{red}_B \,\} \\
\mathit{red}_{A \times B} &= \{\, P : A \times B \mid \mathtt{fst}(P) \in \mathit{red}_A \;\&\; \mathtt{snd}(P) \in \mathit{red}_B \,\}
\end{aligned}$$

. . . but how to define this "semantic" predicate at TA, when T has no fixed semantics?

Structured continuations

- A term abstraction $(x)N$ is a computation term N with a distinguished free variable x.
- A typed continuation K is a finite list of term abstractions:

$$K ::= \mathrm{Id} \mid K \circ (x)N$$

- Apply continuations to computations with nested let:

$$\begin{array}{ll} K : TA \multimap TB \text{ and } M : TA & \mathrm{Id} \mathbin{@} M = M \\ \quad \implies K \mathbin{@} M : TB & (K \circ (x)N) \mathbin{@} M = K \mathbin{@} (\mathit{let}\, x \Leftarrow M \,\mathit{in}\, N) \end{array}$$

  Stack depth of K tracks the $T.\mathit{assoc}$ commuting conversions.

- Continuations reduce: $K \to K'$ iff $\forall M \,.\, K \mathbin{@} M \to K' \mathbin{@} M$.

Reducibility for computations

Definition (Reducibility, completed)

$$
\begin{aligned}
red_\iota &= \{\, M : \iota \mid M\!\downarrow \,\} \\
red_{A\to B} &= \{\, F : A \to B \mid \forall M \in red_A \,.\, FM \in red_B \,\} \\
red_{A\times B} &= \{\, P : A \times B \mid \mathtt{fst}(P) \in red_A \;\&\; \mathtt{snd}(P) \in red_B \,\} \\
red_{TA} &= \{\, M : TA \mid \forall K \in red_A^\top \,.\, (K \mathbin{@} M)\!\downarrow \,\} \\
red_A^\top &= \{\, K : TA \multimap TB \mid \forall N \in red_A \,.\, (K \mathbin{@} [N])\!\downarrow \,\}
\end{aligned}
$$

Structured continuations — specifically $|K|$ — are vital for inductive proofs that $\mathtt{let}$-terms preserve reducibility.

Result

Fundamental Theorem

If $N_1 \in \mathit{red}_{A_1}, \ldots, N_k \in \mathit{red}_{A_k}$ *and* $M : B$ *then*

$$M[x_1 := N_1, \ldots, x_k := N_k] \in \mathit{red}_B \ .$$

(Proof by induction on the structure of term M*)*

Corollary

Each λ_{ml} *term* $M : A$ *is in* red_A*, and hence strongly normalising*

Leap-frog[1]

Jump over continuations to lift properties from values to computations:

General $\top\top$-lifting

$(K \top M \overset{def}{\iff} (K \mathbin{@} M){\downarrow})$

$$\text{Predicate } \phi \subseteq A$$

$$\phi^{\top} = \{\, K \mid K \top [N] \text{ for all } N \in \phi \,\}$$

$$\phi^{\top\top} = \{\, M \mid K \top M \text{ for all } K \in \phi^{\top} \,\} \subseteq TA$$

Continuation K — “observation”
Lifting $\phi^{\top\top}$ — “best observable approximation to ϕ on computations”

[1]Français *saute-mouton*

λ_{ml} + exceptions

Exceptional syntax

[Benton, Kennedy '01; also Erlang '05]

An enhanced let that strictly extends the standard $\mathit{try} \ldots \mathit{catch}$:

$$\mathit{try}\, x \Leftarrow M \,\mathit{in}\, N \,\mathit{unless}\, \{E_1 \mapsto P_1, \ldots\}$$

Evaluate M, bind result to x and evaluate N; unless exception raised in M, in which case engage handler $H = \{E_1 \mapsto P_1, \ldots\}$.

Types and terms

$$\frac{E \in \mathit{Exn}}{\mathit{raise}(E) : TA} \qquad \frac{M : TA \quad N : TB \quad E_i \in \mathit{Exn} \quad P_i : TB}{\mathit{try}\, x^A \Leftarrow M \,\mathit{in}\, N \,\mathit{unless}\, \{E_1 \mapsto P_1, \ldots\} : TB}$$

$$\mathit{let}\, x \Leftarrow M \,\mathit{in}\, N \stackrel{def}{=} \mathit{try}\, x \Leftarrow M \,\mathit{in}\, N \,\mathit{unless}\, \{\}$$

$\top\top$-lifting with exceptions

Put a handler within each continuation frame:

$$K \ ::= \ \mathrm{Id} \ | \ K \circ \langle (x)N, H \rangle \qquad H = \{E_1 \mapsto P_1, \ldots\}$$

$$\mathrm{Id} \mathbin{@} M = M$$

$$(K \circ \langle (x)N, H \rangle) \mathbin{@} M = K \mathbin{@} (\mathrm{try}\, x \Leftarrow M \,\mathrm{in}\, N \,\mathrm{unless}\, H)$$

Reducibility with exceptions and handlers

$$\vdots$$

$$\begin{aligned} \mathit{red}_A^\top &= \{\, K : TA \multimap TB \mid \forall N \in \mathit{red}_A \,.\, (K \mathbin{@} [N])\!\downarrow \\ &\qquad\qquad\qquad\qquad \& \ \forall E \in \mathit{Exn} \,.\, (K \mathbin{@} \mathrm{raise}(E))\!\downarrow \} \\ \mathit{red}_{TA} &= \{\, M : TA \mid \forall K \in \mathit{red}_A^\top \,.\, (K \mathbin{@} M)\!\downarrow \} \end{aligned}$$

Sufficient to prove strong normalisation for λ_{ml} + exceptions

λ_{ml} + sums

Types and terms

Sum type $A + B$, with constructors $\mathit{inl}(M)$, $\mathit{inr}(N)$ and destructor

$$\mathit{case}\,L\,\mathsf{of}\,(\mathit{inl}(x) \Rightarrow M \mid \mathit{inr}(y) \Rightarrow N) : TC$$

Reductions

$$+.\beta_l \qquad \mathit{case}\,\mathit{inl}(M)\,\mathsf{of}\,(\mathit{inl}(x) \Rightarrow P \mid \mathit{inr}(y) \Rightarrow Q) \longrightarrow P[x := M]$$

$$+.\beta_r \qquad \mathit{case}\,\mathit{inr}(N)\,\mathsf{of}\,(\mathit{inl}(x) \Rightarrow P \mid \mathit{inr}(y) \Rightarrow Q) \longrightarrow Q[x := N]$$

$$+.\eta \qquad \mathit{case}\,L\,\mathsf{of}\,(\mathit{inl}(x) \Rightarrow \mathit{inl}(x) \mid \mathit{inr}(y) \Rightarrow \mathit{inr}(y)) \longrightarrow L$$

$+.T$

$$\mathit{let}\,z \Leftarrow (\mathit{case}\,L\,\mathsf{of}\,(\mathit{inl}(x) \Rightarrow M \mid \mathit{inr}(y) \Rightarrow N))\,\mathit{in}\,P$$
$$\longrightarrow \mathit{case}\,L\,\mathsf{of}\,(\mathit{inl}(x) \Rightarrow (\mathit{let}\,z \Leftarrow M\,\mathit{in}\,P) \mid \mathit{inr}(y) \Rightarrow (\mathit{let}\,z \Leftarrow N\,\mathit{in}\,P))$$

$\top\top$-lifting for sum types

Introduce continuations especially for sums:

$$S \ ::= \ K \circ \langle (x)M, (y)N \rangle$$

$$(K \circ \langle (x)M, (y)N \rangle) \mathbin{@} L = K \mathbin{@} (\mathtt{case}\, L\, \mathtt{of}\, (\mathtt{inl}(x) \Rightarrow M \mid \mathtt{inr}(y) \Rightarrow M))$$

Reducibility for sums

$$\vdots$$

$$\begin{aligned} \mathit{red}^{\top}_{A+B} &= \{ S : (A+B) \multimap TC \mid \forall M \in \mathit{red}_A \,.\, (S \mathbin{@} \mathtt{inl}(M))\!\downarrow \\ &\qquad\qquad\qquad\qquad\quad \& \ \forall N \in \mathit{red}_B \,.\, (S \mathbin{@} \mathtt{inr}(N))\!\downarrow \} \\ \mathit{red}_{A+B} &= \{ L : A+B \mid \forall S \in \mathit{red}^{\top}_{A+B} \,.\, (S \mathbin{@} L)\!\downarrow \} \end{aligned}$$

Enough to show strong normalisation for λ_{ml} + sums

Further: use frame stacks for leap-frog definitions of reducibility at sums, products and function types, as well as computations.

Related work

Various closure operators on predicates or relations:

- $\top\top$-closure of [Pitts 2000, Abadi 2000] for defining an operational analogue of admissibility
- Saturation and saturated sets in reducibility proofs: for example, [Girard 1987] for linear logic, [Parigot 1997] for $\lambda\mu$
- Biorthogonality in operational models for recursive types [Melliès, Vouillon 2004]

Other precursors:

- Proof techniques for dynamically-allocated store [Pitts, Stark 1998]
- Evident similarities between leap-frog and continuation-passing style; also the continuation monad itself $TA = R^{(R^A)}$ [qv. Filinski 1994]
- Normalisation for λ_{ml} by translation into $\lambda +$ sums [Benton et al., 1998]

Summary

- $\top\top$-lifting raises operational predicates in λ_{ml} from A to TA:

$$\begin{tikzcd}
\phi \subseteq A \arrow[r, bend left] & \phi^{\top} \subseteq A^{\top} \arrow[r, bend left] & \phi^{\top\top} \subseteq TA \\
\text{values} & \text{continuations} & \text{computations}
\end{tikzcd}$$

"best observable approximation to ϕ"

- Continuations as frame stacks are good for proof by induction
- Example: type-directed reducibility $\implies$ strong normalisation of λ_{ml}
- Extends to treat sums, exceptions

Basis for a normalisation by evaluation algorithm for λ_{ml}; implementation for the monadic intermediate language of the SML.NET compiler

[Lindley PhD 2005]

λ_{ml} in type-preserving compilation

MLj and SML.NET

These compilers use a monadic intermediate language to manage the translation from a higher-order functional language (ML) into an imperative object-oriented bytecode (JVM/.NET).

Typed ML source code
$\downarrow$
Complex MIL
$\downarrow$
Simplified MIL
$\downarrow$
Verifiable bytecode

Monadic Intermediate Language

MIL is λ_{ml} extended with datatypes, exceptions, effects, *etc.*

This *type-preserving* compilation takes types right through to guide optimisation and help generate verifiable code.

Strong normalisation by translation

Map types and terms of λ_{ml} into plain lambda-calculus

$\phi : \lambda_{ml} \longrightarrow \lambda_{\beta\eta}$

$$\phi(0) = 0$$
$$\phi(TA) = \phi(A)$$
$$\phi(A \to B) = \phi(A) \to \phi(B)$$

$$\phi(x) = x$$
$$\phi(MN) = \phi(M)\phi(N)$$
$$\phi(\lambda x.M) = \lambda x.\phi(M)$$
$$\phi(\,[M]\,) = \phi(M)$$
$$\phi(\mathrm{let}\,x \Leftarrow M\,\mathrm{in}\,N) = (\lambda x.\phi(N))\phi(M)$$

Interpret T as the identity type constructor, with no computational effects

Reductions translated

Standard $\lambda_{\beta\eta}$ rewrites go through unchanged; while for computations:

$$\begin{array}{lrcll}
\phi(T.\beta) & (\lambda x.N)M & \longrightarrow & N[M/x] & \\
\phi(T.\eta) & (\lambda x.x)M & \longrightarrow & M & \\
\phi(T.assoc) & (\lambda x.P)((\lambda y.N)M) & \longrightarrow & (\lambda y.(\lambda x.P)N))M & y \notin fv(P)
\end{array}$$

The last rule is a strict extension of $\lambda_{\beta\eta}$, although it is consistent and a known "administrative" reduction from work on continuation-passing style.

Theorem (SN by translation)

λ_{assoc} *is strongly normalising, and hence so is* λ_{ml}.

Proof is combinatorial: a manipulation of rewrite sequences to show that $\mathsf{SN}(\lambda_{\beta\eta}) \implies \mathsf{SN}(\lambda_{assoc})$.

Basic properties of reducibility

The standard useful properties of reducibility:

Theorem (Reducibility)

For every λ_{ml} term M of type A, the following hold:

(i) *If $M \in red_A$, then M is strongly normalising.*

(ii) *If $M \in red_A$ and $M \to M'$, then $M' \in red_A$.*

(iii) *If M is neutral, and whenever $M \to M'$ then $M' \in red_A$, then $M \in red_A$.*

(iv) *If M is neutral and normal (has no reductions) then $M \in red_A$.*

Proof by induction over types. A term is *neutral* if it is of the form x, MN, $fst(M)$ or $snd(M)$.