Reducibility and ⊤⊤-lifting for Computation Types

Ian Stark and Sam Lindley

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

TLCA 2005, Nara, Japan
Friday 22 April 2005

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 in 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 $\mathrm{T}A$ of programs that compute a value of type A

Sample computational effects:

Non-termination, exceptions, I/O, state, non-deterministic choice, jumps, ...

Types and terms of λ_{ml}

Types $\quad A, B \quad ::= \quad \iota \mid A \to B \mid A \times B \mid TA$

Terms $\quad L, M, N, P \quad ::= \quad x^A \mid \lambda x^A.M \mid MN$

$$\mid \quad \langle M, N \rangle \mid \mathit{fst}(M) \mid \mathit{snd}(M)$$

$$\mid \quad [M] \mid \mathrm{let}\, x^A \Leftarrow M \,\mathrm{in}\, N$$

Typing

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

Type constructor T acts as a categorical strong monad

Types and terms of λ_{ml}

Types $\quad A, B \quad ::= \quad \iota \mid A \to B \mid A \times B \mid TA$

Terms $\quad L, M, N, P \quad ::= \quad x^A \mid \lambda x^A.M \mid MN$

$\quad\quad \mid \quad \langle M, N\rangle \mid \mathtt{fst}(M) \mid \mathtt{snd}(M)$

$\quad\quad \mid \quad [M] \mid \mathtt{let}\, x^A \Leftarrow M \,\mathtt{in}\, N$

Typing

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

Type constructor T acts as a categorical strong monad

Types and terms of λ_{ml}

Types $\quad A, B ::= \iota \mid A \to B \mid A \times B \mid TA$

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

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

Type constructor T acts as a categorical strong monad

Applications of λ_{ml}

For example...

- Denotational semantics: extend pure models (domains, categories) 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:

$$\text{T.}\beta \qquad \mathsf{let}\, x \Leftarrow [N]\, \mathsf{in}\, M \longrightarrow M[x := N]$$

$$\text{T.}\eta \qquad \mathsf{let}\, x \Leftarrow M\, \mathsf{in}\, [x] \longrightarrow M$$

$$\text{T.assoc} \qquad \mathsf{let}\, y \Leftarrow (\mathsf{let}\, x \Leftarrow L\, \mathsf{in}\, M)\, \mathsf{in}\, N \longrightarrow \mathsf{let}\, x \Leftarrow L\, \mathsf{in}\, (\mathsf{let}\, y \Leftarrow M\, \mathsf{in}\, N)$$

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)

$$red_\iota = \{ M : \iota \mid M\downarrow \}$$

$$red_{A\to B} = \{ F : A \to B \mid \forall M \in red_A \,.\, FM \in red_B \}$$

$$red_{A\times B} = \{ P : A \times B \mid fst(P) \in red_A \;\&\; snd(P) \in red_B \}$$

... 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:

$$K : TA \multimap TB \text{ and } M : TA \implies K @ M : TB$$

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

$$(K \circ (x)N) @ M = K @ (\mathit{let}\, x \Leftarrow M \,\mathit{in}\, N)$$

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

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

Reducibility for computations

Definition (Reducibility, completed)

$$red_\iota = \{ M : \iota \mid M\downarrow \}$$

$$red_{A \to B} = \{ F : A \to B \mid \forall M \in red_A \,.\, FM \in red_B \}$$

$$red_{A \times B} = \{ P : A \times B \mid \mathtt{fst}(P) \in red_A \;\&\; \mathtt{snd}(P) \in red_B \}$$

$$red_{TA} = \{ M : TA \mid \forall K \in red_A^\top \,.\, (K \mathbin{@} M)\downarrow \}$$

$$red_A^\top = \{ K : TA \multimap TB \mid \forall N \in red_A \,.\, (K \mathbin{@} [N])\downarrow \}$$

Structured continuations help with the inductive proofs that $[-]$ and $\mathtt{let}$ 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

Jump over continuations to lift properties from values to computations:

General $\top\top$-lifting

Predicate $\phi \subseteq A$ $\qquad (K \top M \overset{def}{\iff} (K @ M)\downarrow)$

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

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

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

Extension to λ_{ml} + sums

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

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

Sum continuations

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

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

$$\mathit{red}_{A+B} = \{\, L : A + B \mid \forall S \in \mathit{red}^{\top}_{A+B} \,.\, (S \mathbin{@} L)\!\downarrow \,\}$$

Enough to show SN for λ_{ml} + sums, including commuting conversions

Further: use frame stacks for leap-frog definitions of reducibility at sums, products and function types, even in the plain lambda-calculus.

Extension to λ_{ml} + exceptions

Enhance let with exceptional syntax [Benton, Kennedy '01; also Erlang '05]

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

Continuations with handlers

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

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

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

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]

Evident similarities between leap-frog and continuation-passing style; also the continuation monad itself $TA = R^{(R^A)}$.

Summary and further work

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

$$\phi \subseteq A \quad \longrightarrow \quad \phi^{\top} \subseteq A^{\top} \quad \longrightarrow \quad \phi^{\top\top} \subseteq TA$$

values — continuations — computations

"best observable approximation to ϕ"

- Continuations as frame stacks are good for proof by induction
- Example: type-directed reducibility $\Longrightarrow$ 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]