The Beta Cube

Álvaro García
IMDEA Software and Babel Research Group, Universidad Politécnica de Madrid, Spain
Pablo Nogueira, Emilio Jesús Gallego Arias
Babel Research Group, Universidad Politécnica de Madrid, Spain
{agarcia,pablo,egallego}@babel.ls.fi.upm.es *

Abstract

We define a big-step-style template for reduction strategies that can be instantiated to the foremost (and more) reduction strategies of the pure lambda calculus. We implement the template in Haskell as a parametric monadic reducer whose fixed points are reduction strategies. The resulting code is clean and abstracts away from the machinery required to guarantee semantics preservation for all strategies in lazy Haskell. By interpreting some parameters as boolean switches we obtain a reduction strategy lattice or beta cube which captures the strategy space neatly and systematically. We define a hybridisation function that generates hybrid strategies by composing a base and a subsidiary strategy from the cube. We prove an absorption theorem which states that subsidiaries are left-identities of their hybrids. More properties from the cube remain to be explored.

1 Introduction

Sestoft [Ses02] defines the big-step operational semantics of various reduction strategies for the pure lambda calculus, including call-by-value (cbv), call-by-name (cbn), applicative order (aor), normal order (nor), hybrid applicative order (ha), hybrid normal order (hn), and head spine (he), the latter identical to headNF in [Pau96] but different from head reduction (hr) in [Bar84]. One of his motivations is to clarify the meaning in the pure lambda calculus of strategies used in programming languages, where there are no free variables nor evaluation under lambda. He finds, for example, varying and innacurate definitions of cbn by several authors, including [Plo75]. He implements each strategy as a reducer function in ML using a deep embedding of lambda terms. He does not discuss the paramount implementation issue, first noted by Reynolds [Rey98] in the context of interpretation, of semantics preservation or independence from the evaluation strategy of the implementation language: implementing nor in an eager language, aor in a lazy language, etc. Sestoft employs standard tricks (thunks, etc) to defer evaluation in strict ML. Reynolds showed that continuation-passing style is enough for semantics preservation, but it has a cost in code readability.

We take Sestoft's programme much further.

First, we define a big-step-style template for reduction strategies that can be instantiated to all the aforementioned strategies and more (including Barendregt's hr). We implement the template in Haskell as a higher-order monadic function whose fixed points are reduction strategies. We like to think of this function as a generic reducer although technically it is a functional. The resulting code is readable, clean, and abstracts away from the machinery required to guarantee semantics preservation for all strategies in lazy Haskell.

Second, by interpreting some parameters of the generic reducer as boolean switches, we obtain a reduction strategy lattice we call the β-cube (after Barendregt's λ-cube). The β-cube captures neatly and systematically many reduction strategies of the pure lambda calculus. It contains eight uniform strategies: aor, cbv, cbn, he, and four new ones. We define a hybridisation function that generates up to

*Authors supported by CAM grants CPI/3060/2006, CPI/0622/2008, and P2009/TIC-1465 (PROMETIDOS), and by grants MEC TIN2006-15660-C02-02 (DESAFIOS) and MICINN TIN2009-14599-C03-00 (DESAFIOS10).

ten hybrid strategies (including nor, ha, hn, and hr) by appropriately composing a base and a subsidiary strategy taken from the cube. We prove an absorption theorem which states that subsidiaries are left-identities of their hybrids. More properties from the cube remain to be explored.

2 Technical preliminaries

We consider the pure (untyped or $\lambda\mathbf{K}\beta$) lambda calculus [Bar84]. We write Λ for the set of lambda terms over a set of variables V. We use lowercase letters $(x,y,z,\ldots)$ as meta-variables for variables in V and uppercase letters $(B,M,N,M',N',\ldots)$ as meta-variables for terms in Λ. The syntax of lambda terms is given by $M ::= x \mid (\lambda x.M) \mid (M\,M)$ but we use parenthesis-dropping conventions. We also assume capture-avoiding substitution $M[N/x]$. A reduction strategy (or reduction order) is a partial function $r : \Lambda \to \Lambda$ such that $r\,x = y \implies x \to_\beta^\star y$. We borrow this definition and relations $\equiv$, $\to_\beta$, $\to_\beta^\star$ and $=_\beta$ (syntactic identity, β-reduction, its transitive closure, β-equality) from [Bar84]. Following [Ses02, Plo81] we write $M \xrightarrow{r} N$ for the big-step reduction of M to N using r. We also follow their style in the definition of r's big-step operational semantics. We use 'reducer' in the spirit of [Ses02]. Some readers prefer 'normaliser' when a normal form is always reached but others (e.g. the normalisation by evaluation community) are untroubled. We rather avoid sterile controversy.

3 Rule Template and Generic Reducer

We define a big-step-style template for reduction strategies $\xrightarrow{p}$ that is parametric on strategies $\xrightarrow{la}$, $\xrightarrow{op_1}$, $\xrightarrow{ar_1}$, $\xrightarrow{su}$, $\xrightarrow{op_2}$, and $\xrightarrow{ar_2}$:

$$\textsc{var}\ \frac{}{x \xrightarrow{p} x} \qquad \textsc{abs}\ \frac{B \xrightarrow{la} B'}{\lambda x.B \xrightarrow{p} \lambda x.B'} \qquad \textsc{red}\ \frac{M \xrightarrow{op_1} M' \equiv \lambda x.B \qquad N \xrightarrow{ar_1} N' \qquad B[N'/x] \xrightarrow{su} E}{M\,N \xrightarrow{p} E}$$

$$\textsc{app}\ \frac{M \xrightarrow{op_1} M' \not\equiv \lambda x.B \qquad M \xrightarrow{op_2} M'' \qquad N \xrightarrow{ar_2} N''}{M\,N \xrightarrow{p} M''\,N''}$$

Rule ABS leaves to *la* reduction under lambda. Rule RED relies on op_1 to find the redex's abstraction, on ar_1 to reduce the operand, and on *su* to reduce after substitution. Rule APP describes what to do when op_1 delivers a variable or a non-op_1-reducible application. The result is the application with subterms reduced by op_2 and ar_2. The shape of terms M',N',E, etc, depends on what sort of normal form the parameter strategies deliver (if they terminate). For example, nor is p with $la,su,op_2 = p$, $op_1 =$ cbn, and $ar_1,op_2 =$ id, whereas cbv is p with $op_1,ar_1,su,op_2,ar_2 = p$ and $la =$ id.

If the left-hand-sides of conclusions are non-overlapping then the rules are deterministic [BN98]. This is the case after the computation of the leftmost premise in the APP and RED rules. The template can therefore be interpreted as a syntax-directed partial function in which a term matching the left-hand-side of the conclusion is recursively reduced by strategies in the premises from left to right. Infinite derivation accounts for non-termination. We implement this function in Haskell as a monadic higher-order function shown in Figure 1 (colours explained in Section 4). The monad constraint `m` must be instantiated to a monad that guarantees semantics preservation (e.g., CPS or strict monad). Specific reducers are fixed points of the function. In the monadic code `return` corresponds to the identity strategy.

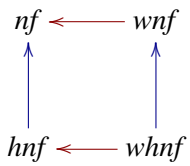
```
type Red = Monad m => Term -> m Term
genred :: Red -> Red -> Red -> Red -> Red -> Red -> Red
genred la op1 ar1 su op2 ar2 t =
    case t of v@(Var _) -> return v
              (Lam v b) -> do b' <- la b
                              return (Lam v b')
              (App m n) -> do m' <- op1 m
                              case m' of
                                (Lam v b) -> do n' <- ar1 n
                                                su (subst b n' x)
                                _         -> do m'' <- op2 m
                                                n'' <- ar2 n
                                                return (App m'' n'')

aor = genred aor    aor aor    aor aor aor
cbv = genred return cbv cbv    cbv cbv cbv
cbn = genred return cbn return cbn cbn return
he  = genred he     he  return he  he  return
...
```

Figure 1: Generic reducer in Haskell. Colours explained in Section 4.

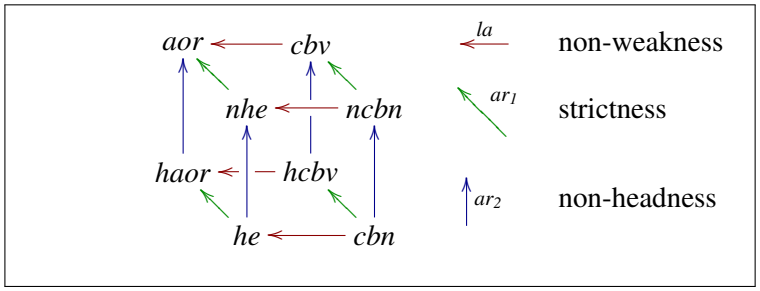
4 The β-cube

The generic reducer `genred` has six parameters. We decrease the number of parameters by focusing on uniform strategies, which are those where `op1`, `op2`, and `su` are recursive calls. So-called hybrid strategies rely on other strategies for `op1`. For example, nor and hr rely on cbn, hn relies on he, and ha relies on cbv [Ses02]. Uniform strategies differ on whether `la`, `ar1` and `ar2` are either recursive calls or `return`. We can encode this variability using the cartesian product of three booleans. The obvious partial order relation on them induces a lattice we call the β-cube (Figure 2). Function `cube2red` delivers a uniform reducer from a vertex in the cube. Some vertices correspond to novel strategies we call head-aor (haor), head-cbv (hcbv), non-head spine (nhe) and non-head cbn (ncbn). Indeed, the boolean parameters (`la`, `ar1` and `ar2`) respectively specify non-weakness (whether abstraction bodies are reduced), strictness (whether arguments are reduced), and non-headness (whether operands with non-reducible applications as operators are reduced). Unsurprisingly, the front and back faces of the cube describe the informal inclusion relation between normal forms ('less reducible form') along the non-headness and non-weakness axes.



5 Hybridisation

Recall from Section 4 that hybrid strategies rely on a uniform strategy (let us call it *subsidiary*) for the `op1` argument (the op_1 strategy in the template). Interestingly, hybrid strategies can be obtained by composing their subsidiary with another uniform strategy from the cube (let us call it *base*) that specifies




```
data BetaCube = BC Bool Bool Bool

sel :: Bool -> Red -> Red
sel b r = if b then r else return

cube2red :: BetaCube -> Red
cube2red (BC la ar1 ar2) = let r = genred (sel la r) r (sel ar1 r) r r (sel ar2 r)
                           in r
cbn = cube2red (BC False False False)
cbv = cube2red (BC False True  True )
...
```

Figure 2: The β-cube and `cube2red`.

the behaviour for `la`, `ar1`, and `ar2`. The subsidiary is in general expected to perform less reduction than the base because the former is only used to locate the redex. The following hybridisation function delivers a hybrid strategy from a subsidiary and a base:

```
hybridise :: (BetaCube, BetaCube) -> Red
hybridise (sub, (BC lab ar1b ar2b)) =
    let s = cube2red sub
        h = genred (sel lab h) s (sel ar1b s) h (s >=> h) (sel ar2b h)
    in h
```

Notice that the Kleisli composition `s >=> h` is the monadic implementation of the relational composition $h \circ s$ and therefore `op2` reduces at least as much as `op1`. For illustration, we show nor as a fixed point of `genred`, using Kleisli composition for the `op2` argument, and as a hybrid of cbn (subsidiary) and nhe (base):

```
nor  = genred nor cbn return nor (cbn >=> nor) nor
nor  = hybridise (BC False False False) (BC True False True)
```

The other cases are: hr is a hybrid of cbn and he, hn of he and nhe, and ha of cbv and aor. Our ha differs from the one in [Ses02] because in the back face of the cube (strictness) the choice is between `return` or the subsidiary (not the hybrid) for `ar1`. This has consequences for our absorption theorem.

6 The Absorption Theorem

Theorem 1 (Absorption Theorem)**.** Let `s` and `b` be respectively a subsidiary and a base strategy that have the same `ar1` argument and that considered as points in the cube satisfy `s` $\sqsubseteq$ `b`. Let `h = hybridise s b` be the hybrid strategy obtained from them. Then `s` is a left identity of `h`, that is, `s >=> h = h`. [Proof omitted for the extended abstract]

7 Conclusions and Future Work

The β-cube captures neatly and systematically the foremost reduction strategies of the pure lambda calculus by means of its uniform strategies and of a hybridisation function that completes the space. The cube helps uncover properties of strategies. Our absorption theorem is one example, but more remain to be explored. The reduction-strategy template suggests itself naturally as a generalisation of the rules of all the well- and less-well-known strategies collected by [Ses02]. The need for op_1 and op_2 in rule APP to accommodate hybrids is perhaps the only subtlety. We are surprised that generic reduction for the pure lambda calculus has, to our knowledge, not been considered before nor its consequences been investigated (e.g., hybrid strategies can be defined in terms of two uniform strategies). The Haskell implementation is deceivingly straightforward. It requires careful attention to semantics preservation and deployment of some advanced Haskell programming. The beta cube is one way of focusing on a subspace of the generic reducer, that of uniform strategies, from which we can obtain more (even new) strategies and state properties.

It is possible to construct versions of our generic reducer for other calculi (simply typed, System F, etc.) and for other representations (de Bruijn indices, nominal terms, explicit substitutions, etc). It is also possible to carry the idea to evaluators (interpreters as in [Rey98] or normalisation by evaluation [Dan96, FR04]). In the typed case we think there is a relation between hybrid strategies and going under lambda during on-the-fly evaluation of abstractions (e.g., aren't evaluators using ha instead of cbv?). We also wish to formalise the cube and prove properties like absorption in terms of reduction strategies as mathematical functions on the set of lambda terms. A first-order inductive representation (which alleviates the pain of α-equivalence) will surely help simplify the number of lemmas and proofs.

Acknowledgements

Many thanks to Jeremy Gibbons, Bruno Oliveira, Richard Bird, Geraint Jones, and Ralf Hinze for their hospitality and invaluable feedback and suggestions. Thanks to Ángel Herranz for improving our presentation of these ideas and to the anonymous referees for their useful comments.

References

[Bar84] Henk P. Barendregt. *The Lambda Calculus, Its Syntax and Semantics*. North Holland, 1984.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[Dan96] Olivier Danvy. Type-directed partial evaluation. In *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 242–257, St. Petersbur (FL), USA, 1996.

[FR04] Andrzej Filinski and Henning Korsholm Rohde. A denotational account of untyped normalization by evaluation. In *7th International Conference on Foundations of Software Science and Computation Structures*, volume 2987 of *Lecture Notes in Computer Science*, pages 167–181, Barcelona, Spain, March 2004.

[Pau96] Lawrence C. Paulson. *ML for the working programmer*. Cambridge University Press, 1996.

[Plo75] Gordon Plotkin. Call-by-name, call-by-value and the lambda calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[Plo81] Gordon Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Department of Computer Science, Aarhus University, Denmark, 1981.

[Rey98] John C. Reynolds. Definitional interpreters for higher-order programming languages. *Higher-Order and Symbolic Computation*, 11(4):363–397, 1998.

[Ses02] Peter Sestoft. Demonstrating lambda calculus reduction. In *The Essence of Computation, Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, volume 2566 of *Lecture Notes in Computer Science*, pages 420–435. Springer, 2002.