Efficient Reduction Functions in Coq

Bruno Barras

INRIA-Rocquencourt

June, 18^{th}, 1998

Plan

1. Motivations
2. Definitions
3. Naive Attempt
4. Abstract Machine (KAM)
5. Substitution Calculus

1 Coq and Reduction

Programs: $\beta\delta\iota$-reduction

Type system: polymorphism, dependent types

- Dependent types:

$$\begin{array}{rcl} \text{Array} & : & \text{nat} \to \text{Set} \\ \begin{array}{|ccc|} 17 & 42 & 5 \end{array} & : & \text{Array}(3) \ : \ \text{Set} \\ T_1 & : & \text{Array}(2+1) \\ T_2 & : & \text{Array}(\sqrt{9}) \end{array}$$

$$T_1 = T_2 \text{ well-typed ?}$$
$$\text{Array}(2+1) \stackrel{?}{\equiv} \text{Array}(\sqrt{9})$$

- Proof by reflection:

$$\begin{aligned}(a+b)(a-b) &= a^2-b^2\\ a(a-b)+b(a-b) &= a^2-b^2\\ a^2-ab+(ba-b^2) &= a^2-b^2\\ a^2-b^2 &= a^2-b^2\end{aligned}$$

  Define F such that

  - $F((X+Y)(X-Y)) =_{\beta\delta\iota} X^2-Y^2$
  - $\forall a,b,P.\ F(P)(a,b) = P(a,b)$

2 Definitions

Pure λ-calculus

De Bruijn notation:

$$t := n \ (n \in \mathbb{N}) \mid \lambda t \mid (t \ t')$$

Example:

$$\lambda u.(\lambda x.(x \ \lambda y.x) \ (\lambda z.z \ u))$$

$$\lambda_u(\lambda_x(0_x \ \lambda_y 1_x) \ (\lambda_z 0_z \ 0_u))$$

β-reduction:

$$(\lambda m \ n) \to \{n/0\}m$$

Goal: compute β (head) normal form

Coq's encoding of terms

- λ is used as a binder
- ( ) bears a tag.

Typed λ-calculus:

$$\begin{aligned} \lambda x\!:\!T.\ M &\equiv \mathrm{Lambda}_x(T\ \lambda M) \\ (M\ N) &\equiv \mathrm{AppL}(M\ N) \end{aligned}$$

New β rule:

$$\mathrm{AppL}(\mathrm{Lambda}_x(T\ \lambda M)\ N) \to \{N/0\}M$$

Strategies

Call by value (ML)

Weak normal form of argument first

$$
\begin{array}{ll}
 & (\lambda x.(x+x) \;\; (\hat{1}+\hat{1})) \\
\to_\iota & (\lambda x.(x+x) \;\; \hat{2}) \\
\to_\beta & \hat{2}+\hat{2} \\
\to_\iota & \hat{4}
\end{array}
$$

+ Avoid duplication of redexes

− Useless computations

⇒ Faster on functional programs

Call by name

Leftmost outermost first

$$
\begin{array}{ll}
 & (\lambda x.(x+x)\ \ (\hat{1}+\hat{1})) \\
\to_\beta & (\hat{1}+\hat{1})+(\hat{1}+\hat{1}) \\
\to_\iota & \hat{2}+(\hat{1}+\hat{1}) \\
\to_\iota & \hat{2}+\hat{2} \\
\to_\iota & \hat{4}
\end{array}
$$

+ Laziness

− Redundant computations

⇒ Better for proof normalisation

Lazy (LML)

Call by name with sharing

$$\begin{array}{ll} & (\lambda x.(x+x) \;\; (\hat{1}+\hat{1})) \\ \rightarrow_{\beta} & (\underline{\hat{1}+\hat{1}}) + (\underline{\hat{1}+\hat{1}}) \\ \rightarrow_{\iota} & \hat{2}+\hat{2} \\ \rightarrow_{\iota} & \hat{4} \end{array}$$

+ Number of reduction steps $<$ CBV, CBN

− Overhead (updates)

$\Rightarrow$ Good trade-off

3 First implementation

```
let rec norm = function
  Rel i     -> Rel i
| Lam t     -> Lam (norm t)
| App(a,b) -> (match (norm a) with
                  Lam f -> norm (subst b f)
                | na -> App(na, norm b))

let rec nrm_whd = function
  (App(a,b), s)    -> nrm_whd (a, (b::s))
| (Lam f, (b::s)) -> nrm_whd (subst b f, s)
| (Lam t, [])      -> Lam (whdn t)
| (Rel i, s)       -> app_list (Rel i) (List.map whdn s)
and whdn t = nrm_whd (t,[])
```

Quadratic factor

$O(n)$ substitutions in terms of size $O(n)$

Memory allocation

$$(\lambda_x\lambda_y(\overbrace{\ldots 1_x \ldots \lambda\lambda \ldots 2_y \ldots 3_x \ldots}^{M})\ T_1\ T_2)$$

$$(\lambda_y(\ldots \uparrow^1 T_1 \ldots \lambda\lambda \ldots 2_y \ldots \uparrow^3 T_1 \ldots)\ T_2)$$

$$(\ldots T_1 \ldots \lambda\lambda \ldots \uparrow^2 T_2 \ldots \uparrow^2 T_1 \ldots)$$

2 substitutions:

- M re-allocated twice
- T_1 re-allocated 4 times
- T_2 re-allocated once

Parallel substitution

$$(\lambda_x \lambda_y (\overbrace{\ldots 1_x \ldots \lambda\lambda \ldots 2_y \ldots 3_x \ldots}^{M})\ T_1\ T_2)$$
$$(\{T_1/0_x\}\lambda_y M\ T_2)$$
$$\{T_1/1_x;\ T_2/0_y\}M$$
$$(\ldots T_1 \ldots \{T_1/1_x;\ T_2/0_y\}\lambda\lambda \ldots 2_y \ldots 3_x \ldots)$$
$$(\ldots T_1 \ldots \lambda\lambda \ldots \uparrow^2 T_2 \ldots \uparrow^2 T_1 \ldots)$$

- M, T_1, T_2 re-allocated once

Closure = delayed substitution

Environment = parallel substitution

$$C := \langle E, t \rangle \qquad E := \vec{C}$$

- In CBN, arguments are frozen
- In CBV, partial application

4 Krivine's Abstract Machine

State: $(e, t, s) \in E \times t \times \vec{C}$

Initial: $([\,], t, [\,])$

Transitions:

$$
\begin{array}{rcl}
(e, (u\ v), s) & \rightarrow & (e, u, \langle e, v\rangle :: s) \\
(e, \lambda t, c :: s) & \rightarrow & (c :: e, t, s) \\
(\langle e', v\rangle :: e, 0, s) & \rightarrow & (e', v, s) \\
(c :: e, n+1, s) & \rightarrow & (e, n, s)
\end{array}
$$

Computes weak head normal form (CBN)

KAM under abstractions (KN, Crégut 91)

```
type clos = Clos of env * lambda
 and env  = clos list


let rec kn k = function
   (e, App(a,b), s)       -> kn k (e, a, (Clos(e,b)::s))
 | (e, Lam f, (arg::s))  -> kn k ((arg::e), f, s)
 | (((Clos(e',u))::_), Rel 0, s) -> kn k (e', u, s)
 | ((_::e), Rel n, s)    -> kn k (e, Rel (n-1), s)
 | (e, Lam f, [])         ->
     Lam (kn (k+1) (((Clos([], Rel (-k-1)))::e), f, []))
 | ([], Rel n, s)         ->
     app_list (Rel (k+n))
       (List.map (fun (Clos(e,t)) -> kn k (e,t,[])) s)
```

$$\text{succ} = \lambda\lambda\lambda(1\ (2\ 1\ 0))$$

$$(A, B) = \lambda(0\ \uparrow^1 A\ \uparrow^1 B)$$

$$\text{pred} = \lambda(0\ \lambda(\text{succ}\ (0\ \lambda\lambda 1), 0\ \lambda\lambda 1)\ (\hat{0}, \hat{0})\ \lambda\lambda 0)$$

(pred $\hat{n}$)	$T_{\texttt{norm}}$	$T_{\texttt{whdn}}$	$T_{\texttt{kn}}$	$T_{\texttt{OCaml}}$
10	0.3	0.02		
14	5			
15	9			
16	19			
50		0.6	0.005	
100		3.3	0.01	0.0004
200		19	0.021	
1000			0.14	0.005
	$O(2^n)$	$O(n^2)$	$O(n)$	$O(n)$

5 Explicit Substitutions

Why ?

- Closer to the implementation
- Correctness proof

Closure Calculus

Mutable terms with shifts:

$$T := \bar{n}\ (n \in I\!N) \mid \lambda_{\langle S,t\rangle}T \mid (T\ T') \mid \uparrow^n T \mid [S]t$$

Substitution (cf $\lambda\phi$, in Lescanne 93)

$$S := \mathrm{id} \mid S.T \mid \uparrow^n S \mid \Uparrow^n S$$

Sharing

$$\begin{array}{rl} & \lambda u.(\underline{\lambda x}.(x\ \lambda y.x)\ (\underline{\lambda z.z\ u})) \\ \to & \lambda u.(\underline{\lambda z.z\ u}\ \lambda y.(\underline{\lambda z.z\ u})) \\ \to & \lambda u.(\underline{u}\ \lambda y.\underline{u}) \end{array}$$

De Bruijn:

$$\begin{array}{rl} & \lambda_u(\underline{\lambda_x}(0_x\ \lambda_y 1_x)\ (\underline{\lambda_z 0_z\ 0_u})) \\ \to & \lambda_u(\underline{\lambda_z 0_z\ 0_u}\ \lambda_y(\underline{\lambda_z 0_z\ 1_u}))) \end{array}$$

Explicit shift:

$$\begin{array}{rl} \to & \lambda_u(\underline{\lambda_z 0_z\ 0_u}\ \lambda_y {\uparrow}^1(\underline{\lambda_z 0_z\ 0_u})) \\ \to & \lambda_u(\underline{0_u}\ \lambda_y {\uparrow}^1\underline{0_u}) \end{array}$$

Weak reduction or not ?

```
let rec pow2 = function
  0 -> 1
| n -> let x = pow2 (n-1) in x+x

let rec pow2 = function
  0 -> 1
| n -> (pow2 (n-1))+(pow2 (n-1))
```

$$\begin{array}{ll} & (\lambda f.\lambda x.(x\ f\ f\ (f\ x))\ \lambda y.M) \\ \to & \lambda x.(x\ \lambda y.\mathsf{nf}(M)\ \lambda y.\mathsf{nf}(M)\ \ \{x/y\}\,M) \end{array}$$

$\Rightarrow$ 2 usages for f

Reduction Rules

Substitution:

$$
\begin{array}{rcl}
[S](u\ v) & \to & ([S]u\ \ [S]v) \\
[S]\lambda t & \to & \lambda_{\langle S,t\rangle}\, [\Uparrow^1 S]\, t \\
[\mathrm{id}]n & \to & \bar{n} \\
[S.\,T]0 & \to & T \\
[S.\,T](n+1) & \to & [S]n \\
[\uparrow^k S]\, n & \to & \uparrow^k [S]n \\
[\Uparrow^k S]\, n & \to & \bar{n}\ \ \text{if } n < k \\
[\Uparrow^k S]\, n & \to & \uparrow^k [S](n-k)\ \ \text{if } n \geq k
\end{array}
$$

Confluent and terminates (no composition)

Beta:

$$
\begin{array}{rcl}
(\lambda_{\langle S,m\rangle} T\ N) & \to & [S.\,N]m \\
(\uparrow^k \lambda_{\langle S,m\rangle} T\ N) & \to & [\uparrow^k S.\,N]\, m
\end{array}
$$

$$
\begin{array}{rcl}
(\uparrow^n T, s) & \to & (T, \uparrow^n :: s) \\
((M\ N), s) & \to & (M, N :: s) \\
(\lambda_{\langle S,t\rangle} T, M :: s) & \to & (S.\, M, t, s) \\
(\lambda_{\langle S,t\rangle} T, \uparrow^n :: M :: s) & \to & (\uparrow^n S.\, M, t, s) \\
(\# a, s) & \to & (!a, \mathrm{U}(a) :: s) \\
(M = \lambda_{\langle S,t\rangle} T, \mathrm{U}(a) :: s) & \to & (M, s) \ \ a := M \\
(\bar{n}, \mathrm{U}(a) :: s) & \to & (\bar{n}, s) \ \ a := \bar{n} \\
([S]t, s) & \to & (S, t, s) \\
(S, \lambda t, s) & \to & (\lambda_{\langle S,t\rangle} \left[\Uparrow^1 S\right] t, s) \\
(S, (m\ n), s) & \to & (S, m, [S]n :: s) \\
(S, n, s) & \to & (S(n), s)
\end{array}
$$

	CBN	Lazy
Recursive function	Coq V5.10	Coq V6.1
Abstract machine	KN	KNP, Coq V6.3?