Towards an Atomic Abstract Machine

David R. Sherratt

The 5th Southern-Region English Programming Language Seminar (S-REPLS)
University of Oxford

January 12, 2017

Outline

What do we want?

Lambda Calculus → Functional Programming

Efficient evaluation mechanism → Efficient functional programs

Full Laziness

```
f x = x + sqrt 16
```

=

```
SQRT16 = sqrt 16
f x = x + SQRT16
```

[7] Wadsworth, C. Semantics and Pragmatics of the Lambda-Calculus. Diss. University of Oxford, 1971.

What do we have?

The Atomic Lambda-Calculus

Atomic duplication. Almost fully local.

The Lambda-Calculus with Director Strings

Makes variable information local.

Atomic lambda-calculus: The basic calculus

Grammar of Basic Calculus

$$t ::= x \mid \lambda x.t \mid (t)t \mid t[x_1, ...x_n \leftarrow t]$$

Atomic duplication

$$\ldots x_1 \ldots x_2 \ldots [x_1, x_2 \leftarrow (t)u]$$

$$\ldots (y_1) z_1 \ldots (y_2) z_2 \ldots [y_1, y_2 \leftarrow t][z_1, z_2 \leftarrow u]$$

Atomic duplication

$$\ldots x_1 \ldots x_2 \ldots [x_1, x_2 \leftarrow \lambda y.t]$$

$$\ldots \lambda y_1.z_1 \ldots \lambda y_2.z_2 \ldots [z_1, z_2 \leftarrow t]$$

Atomic duplication

$$\ldots(x_1)w\ldots(x_2)y\ldots[x_1, x_2 \leftarrow \lambda y.\ldots y\ldots][y \leftarrow s]$$

$$\ldots(\lambda y_1.z_1)w\ldots(\lambda y_2.z_2)y\ldots[z_1, z_2 \leftarrow \ldots y\ldots][y \leftarrow s]$$

Atomic duplication

$$\ldots x_1 \ldots x_2 \ldots [x_1, x_2 \leftarrow \lambda y.t]$$

$$\ldots \lambda y_1.z_1 \ldots \lambda y_2.z_2 \ldots [z_1, z_2 \leftarrow t]$$

$$\ldots x_1 \ldots x_2 \ldots [x_1, x_2 \twoheadleftarrow \lambda y.\langle z_1, z_2 \rangle [z_1, z_2 \leftarrow t]]$$

Atomic lambda-calculus: Distributor

Grammar of Atomic lambda-calculus

$$t ::= x \mid \lambda x.t \mid (t)t \mid t[x_1, ...x_n \leftarrow t] \mid t[x_1, ...x_n \twoheadleftarrow \lambda y.t^n]$$

$$t^n ::= \langle t_1, ...t_n \rangle \mid t^n[x_1, ...x_m \leftarrow u] \mid t^n[x_1, ...x_m \twoheadleftarrow \lambda y.t^m]$$

Atomic lambda-calculus: Fully Lazy Sharing

$$\ldots z_1 \ldots z_2 \ldots [z_1, z_2 \leftarrow \lambda x.t[\leftarrow x]]$$

$$\rightsquigarrow \ldots z_1 \ldots z_2 \ldots [z_1, z_2 \twoheadleftarrow \lambda x.\langle y_1, y_2 \rangle [y_1, y_2 \leftarrow t[\leftarrow x]]]$$

$$\rightsquigarrow \ldots z_1 \ldots z_2 \ldots [z_1, z_2 \twoheadleftarrow \lambda x.\langle y_1, y_2 \rangle [y_1, y_2 \leftarrow t][\leftarrow x]]$$

$$\rightsquigarrow \ldots z_1 \ldots z_2 \ldots [z_1, z_2 \twoheadleftarrow \lambda x.\langle y_1, y_2 \rangle [\leftarrow x]][y_1, y_2 \leftarrow t]$$

$$\rightsquigarrow \ldots \lambda x_1.y_1[\leftarrow x_1] \ldots \lambda x_2.y_2[\leftarrow x_2] \ldots [y_1, y_2 \leftarrow t]$$

The idea

Implement an atomic abstract machine for atomic lambda-terms that performs full laziness

Atomic lambda-calculus is not *local*.

$$\lambda x.t[\gamma] \rightsquigarrow (\lambda x.t)[\gamma]$$
$$\text{iff } x \notin \mathsf{FV}(\gamma)$$

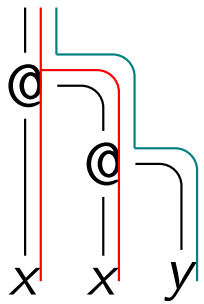
[6] Y. Lafont. (1989). Interaction nets.

Directed lambda-calculus

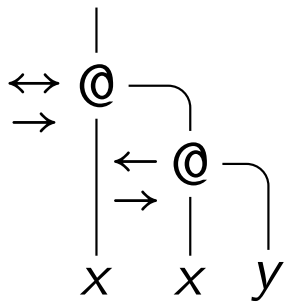
$$\mathcal{A} = (\downarrow \mid \text{-})^* \qquad \mathcal{B} = (\leftarrow \mid \leftrightarrow \mid \rightarrow \mid \text{-})^*$$

$$t ::= \square^{\alpha} \mid (\lambda t)^{\alpha} \mid (t)t^{\beta} \mid t[k \leftarrow t]^{\beta}$$
$$k \geq 1$$

$$(x)((x)y)[x \leftarrow X][y \leftarrow Y]$$



Paths



Annotated term

[1] M. Fernández, I. Mackie, and F-R. Sinot. (2005). Lambda-calculus with director strings.

Directed **atomic** lambda-calculus

$$\mathcal{D} = (0 \mid 1)^*$$

$$t_n ::= \square_{n=1} \mid (\lambda t_{n+1}) \mid (t_k)t_m{}^{\sigma}_{\substack{|\sigma| = n \\ |\sigma|_0 = k \\ |\sigma|_1 = m}} \mid t_{k+i}[\gamma]_{i,m}{}^{\sigma}_{\substack{|\sigma| = n = k+m \\ |\sigma|_0 = k \\ |\sigma|_1 = m}}$$

$$[\gamma]_{k,n} ::= [a_1, \ldots, a_k \leftarrow t_n] \mid [a_1, \ldots, a_k \twoheadleftarrow \lambda.d_{k,n}]$$

$$d_{k,n} ::= u_{k,n} \mid d_{k,j+l}[\gamma]_{l,h}{}^{\sigma}_{\substack{|\sigma| = n = j+h \\ |\sigma|_0 = j \\ |\sigma|_1 = h}}$$

$$u_{k,n} ::= \langle\rangle_{k=n=0} \mid \langle t_n \rangle_{k=1} \mid \langle t_j, u_{k-1,l} \rangle^{\sigma}_{\substack{|\sigma| = n = j+l \\ |\sigma|_0 = j \\ |\sigma|_1 = l}}$$

$$\lambda x.t[\gamma] \rightsquigarrow (\lambda x.t)[\gamma] \quad \text{iff } x \notin \mathsf{FV}(\gamma) \qquad \lambda t[\gamma]^{\sigma 0} \rightsquigarrow (\lambda t)[\gamma]^{\sigma} \quad \text{iff last director in string is 0}$$

Directed atomic lambda-calculus: Substitution

```
sub :: Term -> Int -> Term -> Scope -> Term
sub (V x) 1 v ys = v
sub (L x) n v ys        | m >= n        = (L (sub x n v (addZero ys)))
                        | otherwise     = (L x)
                              where (Just m) = checkScopes (L x)
sub (A s t a) n v ys    | (getith a n) == 0     = (A (sub s (noOfZero (headSubScope a n)) v (phiL ys (removeScope a n)))
                          t (psiOne ys (removeScope a n)))
                        | (getith a n) == 1     = (A s (sub t (noOfOne (headSubScope a n)) v (phiR ys (removeScope a n)))
                          (psiTwo ys (removeScope a n)))
                        | (getith a n) == (-1)  = (A s t a)
sub (C t c a) n v ys    | (getith a n) == 0     = (C (sub t l v bs) (setSharedVar c js) zs)
                        | (getith a n) == 1     = (C t (subC c (noOfOne (headSubScope a n)) v (phiR ys (removeScope a n)))
                          (psiTwo ys (removeScope a n)))
                        | (getith a n) == (-1)  = (C t c a)
                                      where   (xs, zs) = swapShar (removeScope a n) ys
                                              k = noOfZero (headSubScope a n)
                                              l = nextInList (getSharedVar c) k
                                              ks = allNewPos (getSharedVar c) [l]
                                              js = kappaL xs ks
                                              bs = thetaL xs ks
                                              --l = k + (noOfLess (getSharedVar c) k)
subC :: Closure -> Int -> Term -> Scope -> Closure
subC (S xs t) n v ys    = (S xs (sub t n v ys))
subC (D xs t) n v ys    = (D xs (subD t n v ys))
subD :: DTerm -> Int -> Term -> Scope -> DTerm
subD (T xs) n v ys       = (T (subT xs n v ys))
subD (D2 d c s) n v ys  | (getith s n) == 0     = (D2 (subD d k v bs) (setSharedVar c js) zs)
                        | (getith s n) == 1     = (D2 d (subC c (noOfOne (headSubScope s n)) v (phiR ys (removeScope s n)))
                          (psiTwo ys (removeScope s n)))
                        | (getith s n) == (-1)  = (D2 d c s)
                                      where   (xs, zs) = swapShar (removeScope s n) (addZero ys)
                                              k = noOfZero (headSubScope s n)
                                              l = nextInList (getSharedVar c) k
                                              ks = allNewPos (getSharedVar c) [l]
                                              js = kappaL xs ks
                                              bs = thetaL xs ks
```

Directed atomic lambda-calculus: Substitution

$$(u)t^{\alpha}[k \leftarrow s]^{\beta} \rightsquigarrow (u[j \leftarrow s]^{\sigma_2})t^{\sigma_1}$$

$\alpha \setminus \alpha_k$	β		σ_2
0	0	$\rightsquigarrow$	0
1	0	$\rightsquigarrow$	-
-	1	$\rightsquigarrow$	1

$\alpha \setminus \alpha_k$	β		σ_1
0	0	$\rightsquigarrow$	0
1	0	$\rightsquigarrow$	1
-	1	$\rightsquigarrow$	0

$$j = |\alpha_1..\alpha_k|_0$$

$$(u)t^{00101}[4 \leftarrow s]^{010001} \rightsquigarrow (u[3 \leftarrow s]^{0101})t^{000110}$$

Summarize

The Atomic Lambda-Calculus

Atomic duplication. Almost fully local.

The Lambda-Calculus with Director Strings

Makes variable information local.

The Directed Atomic Lambda-Calculus

Fully Local.

Bibliography

[1] M. Fernández, I. Mackie, and F-R. Sinot. (2005). Lambda-calculus with director strings.
[2] Guglielmi, Gundersen, & Parigot. (2010). A proof calculus which reduces syntactic bureaucracy.
[3] Gundersen, Heijltjes, & Parigot. (2013). Logic in Computer Science
[4] Hendriks and van Oostrom. (2003). λ.
[5] J.-L. Krivine. (2007). A call-by-name lambda-calculus machine. Higher-Order and Symbolic Computation.
[6] Y. Lafont. (1989). Interaction nets.
[7] Wadsworth, C. Semantics and Pragmatics of the Lambda-Calculus. Diss. University of Oxford, 1971.