Higher-Order Symb Comput (2007) 20: 231–235
DOI 10.1007/s10990-007-9019-8

On the correctness of the Krivine machine

Mitchell Wand

Published online: 24 October 2007

Abstract We provide a short proof of the correctness of the Krivine machine by showing how it simulates weak head reduction.

Keywords Krivine machine · Lambda calculus · Weak head reduction

1 Correctness

Recall that every closed λ-term is either in weak head normal form (that is, an abstraction), or is of the form

$$(\lambda x.M)\ N_1\ \dots\ N_k,$$

where as usual concatenation denotes application and associates to the left.

So weak head reduction is simply the single reduction

$$(\lambda x.M)\ N_1\ \dots\ N_k \to M[N_1/x]\ N_2\ \dots\ N_k.$$

If we represent the λ-terms $(\lambda x.M), N_1, \dots, N_k$ as closures, we can avoid doing the substitution. To do this, let us define some syntax:

M	$::=$	$x \mid M\ N \mid \lambda x.M$	Terms
C	$::=$	(M, ρ)	Closures
ρ	$::=$	$[] \mid \rho[x = C]$	Environments

Note that ρ is a syntactic structure, not a function. Nevertheless, we interpret ρ as a partial function in the usual way: $(\rho[x = C])(x) = C$ and $(\rho[x = C])(y) = \rho(y)$ for $y \neq x$.

M. Wand (✉)
College of Computer and Information Science, Northeastern University, Boston, MA 02115, USA
e-mail: wand@ccs.neu.edu

We define the size of a closure to be the size of its syntax tree according to the grammar above. This implies that the size of (M, ρ) is always greater than the size of $\rho(x)$ for any free variable x of M.

We define a translation U from closures to λ-terms. The translation of a closure is obtained by substitution

$$\begin{array}{lll}
U(M, []) & = M & \\
\text{otherwise:} & & \\
U(x, \rho) & = U(\rho(x)) & \text{if } \rho(x) \text{ is defined} \\
 & = x & \text{if } \rho(x) \text{ is undefined} \\
U(MN, \rho) & = U(M, \rho)\, U(N, \rho) & \\
U(\lambda x.M, \rho) & = \lambda x.U(M, \rho - x) &
\end{array}$$

where $\rho - x$ denotes ρ with all bindings for x removed. In the line for abstractions, x should be α-renamed in the usual way if necessary to avoid variable capture.

We say that a closure C is *closed* iff $U(C)$ is closed. It is easy to see that if (M, ρ) is closed, then $\rho(x)$ is defined and is closed for every free variable of M. This implies that in $U(C)$, it will never be necessary to α-rename to avoid variable capture.

Lemma 1 $U(M, \rho[x = C]) = (U(M, \rho - x))[U(C)/x]$

Proof Easy induction on M. □

We define a *configuration* to be a sequence $C_0 \ldots C_n$ of closed closures. We use the metavariables L and R to range over configurations. In the configuration $C_0 \ldots C_n$, we call C_0 the *head closure* of the configuration. We extend the translation U to configurations by translating the configuration to the left-associative application of its elements, analogous to concatenation of terms:

$$U(C_0 \ldots C_n) = U(C_0 \ldots C_{n-1})\, U(C_n) \qquad n > 0$$

So if we represent $(\lambda x.M)\, N_1 \ldots N_k$ by $C_0\, C_1 \ldots C_k$, we can simulate the behavior of a weak head reduction by the three rules

$$\begin{array}{llll}
(x, \rho)\, C_1 \ldots C_k & \to \rho(x)\, C_1 \ldots C_k & & \textbf{var} \\
((M\, N), \rho)\, C_1 \ldots C_k & \to (M, \rho)\, (N, \rho)\, C_1 \ldots C_k & & \textbf{app} \\
((\lambda x.M), \rho)\, C_1 \ldots C_k & \to (M, \rho[x = C_1])\, C_2 \ldots C_k & (k \geq 1) & \textbf{abs}
\end{array}$$

which is our version of the Krivine Machine. The first two rules simulate the behavior of U, and the third rule simulates head reduction. In the first rule, $\rho(x)$ is always defined, because in a configuration all the C_i are closed. We can make this observation precise as follows:

Lemma 2 *If L is a configuration, and $L \rightarrow R$, then R is a configuration.*

Proof If L is a configuration, then it is closed. We proceed by cases on the term of the head closure of L. If the term in the head closure is a variable x, then $\rho(x)$ must be defined, since L is configuration, and is therefore closed. The other two rules are easily seen to preserve closedness. □

The key lemma is:

Lemma 3 *For each reduction step $L \rightarrow R$ of the Krivine machine, either:*

1. *$U(L)$ and $U(R)$ are identical λ-terms, and the head closure of R is smaller than the head closure of L, or*
2. *$U(L) \rightarrow U(R)$ by a weak head reduction.*

Furthermore, if L is a terminal configuration, then $U(L)$ is an abstraction.

Proof The **var** and **app** rules satisfy case 1; the **abs** rule satisfies case 2. Since, by the previous lemma, the **var** rule can never stick, L is in a terminal configuration iff L consists of a single closure whose term is an abstraction. Hence $U(L)$ is an abstraction. □

Therefore, we have:

Theorem 1 *If M is a closed term, and L is a configuration such that $U(L) = M$, then the Krivine machine reduces L to a terminal configuration R iff M reduces to a weak head normal form N. Furthermore, $U(R) = N$.*

Proof If the Krivine machine terminates in configuration R, then by the preceding lemma, M head-reduces to $U(R)$. Since both weak head reduction and the Krivine machine halt as soon as they reach a single abstraction, we have $U(R) = N$.

For the converse, assume that the Krivine machine, started at L, takes infinitely many steps. The machine can take only finitely many **var** and **app** steps before it takes an **abs** step, which simulates a weak head reduction. Hence, if the Krivine machine runs infinitely, so does the weak head reduction of $U(L)$. □

2 A modification

The machine as presented has a well-known space leak: recursive invocation of an application in which the argument is a variable can lead to nested thunks. Consider the reduction of $(\lambda x.xx)\ (\lambda x.xx)$. Let $\Delta = (\lambda x.xx)$, let $\rho_1 = [x = (\Delta, [])]$, $\rho_2 = [x = (x, \rho_1)]$, etc. Then the machine reduces as follows:

$$\begin{aligned}
&(\Delta\ \Delta, []) \rightarrow (\Delta, [])\ (\Delta, []) \\
&\rightarrow (x\ x, \rho_1) \rightarrow (x, \rho_1)\ (x, \rho_1) \rightarrow (\Delta, [])\ (x, \rho_1) \\
&\rightarrow (x\ x, \rho_2) \rightarrow (x, \rho_2)\ (x, \rho_2) \rightarrow (x, \rho_1)\ (x, \rho_2) \rightarrow (\Delta, [])\ (x, \rho_2) \\
&\rightarrow (x\ x, \rho_3) \ldots
\end{aligned}$$

The machine loops, as required, and uses only a bounded number of stack positions, but the environments grow unboundedly, even though the head reduction of $\Delta\ \Delta$ should require only constant space.

This problem can be remedied by introducing a special case into the **app** rule as follows:

$$((M\,x),\rho)\,C_1\,\ldots\,C_k \to (M,\rho))\,\rho(x)\,C_1\,\ldots\,C_k \qquad \textbf{app-var}$$

if N is not a variable:

$$((M\,N),\rho)\,C_1\,\ldots\,C_k \to (M,\rho)\,(N,\rho)\,C_1\,\ldots\,C_k \qquad \textbf{app}$$

The new machine satisfies the additional invariant that if the binding $[x = (M, \rho')]$ appears in an environment, then M is not a variable, thus eliminating the trivial bindings that caused the problem in the reduction of $\Delta\ \Delta$. The new machine reduces $\Delta\ \Delta$ as follows:

$$(\Delta\,\Delta,[]) \to (\Delta,[])\,(\Delta,[]) \to (x\,x,\rho_1) \to (x,\rho_1)\,(\Delta,[]) \to (\Delta,[])\,(\Delta,[])\ldots$$

Thus it uses only constant space, as required.

The lemma and theorem above clearly go through *mutatis mutandum* for the revised machine.

3 Related work

The standard presentation of the Krivine machine, as in [3, 4, 6], models configurations as triples (c, e, s), where c is a λ-term, e is an environment, and s is a stack of closures. Furthermore, the standard presentation uses de Bruijn indices rather than explicit variable names.

Krivine's original presentation [5] differs from the standard presentation in one important way. He writes:

> Execution of $\lambda x_1 \ldots \lambda x_n.t$, where t does not begin with a $\lambda\ldots$: Create an environment $(e,\ x_1,\ldots,x_n)$ [where] e is the address of E, and pop $x_1,\ldots,x_n$ from the stack. Place in E the address of this new environment and execute t (make T point to t).[1]

In combination with this rule, Krivine's presentation uses de Bruijn indices that are pairs of integers (ν, k), where ν corresponds to the number of λ-blocks above and k is the position of the variable within such a block.

According to this rule, $((\lambda x\lambda y.x)\ (\lambda z.z))$ would take a single step and then become stuck, since the application of the first abstraction requires the machine to unstack two arguments from the stack and it will have only one. Since this term has a weak head-normal form, but the machine does not find it, the rule as stated does not quite achieve completeness. Luckily, this case arises only when the next set of beta-reductions would reach a whnf, so completeness is easily restored by making the interpreting the stuck configuration appropriately.

Krivine's original presentation offers no hints about correctness. Leroy [6] proves the soundness, but not the completeness, of the machine relative to a calculus with explicit substitution. Abadi et al. [1] prove both soundness and completeness for weak head normal forms in their calculus $\lambda\sigma$ of explicit substitutions; the standard Krivine machine can be

[1] "Execution de $\lambda x_1 \ldots \lambda x_n.t$ où t ne commence pas par $\lambda\ldots$: On crée un environnement $(e,\ x_1,\ldots,x_n)$ [où] e est l'adresse de E, et $x_1,\ldots,x_n$ sont dépilées. On met dans E l'adresse de ce nouvel environnement et on va exécuter t (on fait pointer T sur t)."

seen to be the restriction of their machine to the case without composition (their $s \circ s'$). Eliminating composition makes possible a much simpler proof than the one in [1].

A number of other derivations or explanations for the Krivine machine have been published. Hannan and Miller [4] derive the Krivine machine from a big-step operational semantics for the λ-calculus under call-by-name. Crégut [3] extends the machine to produce full normal forms. Streicher and Reus [8] derive the Krivine machine from a denotational continuation semantics. Ager et al. [2] derive the Krivine machine by taking a compositional call-by-name interpreter, defunctionalizing it, and then converting to continuation-passing style, and then defunctionalizing the continuations.

The problem with nested thunks for operands that are variables is classic, as is the solution discussed here. For example, Randell and Russell [7] observed that variables should be dereferenced before putting them on the operand stack.

4 Historical note and acknowledgements

My notes indicate that I figured out this explanation for a semantics class in the Winter of 1992.

Thanks to Olivier Danvy for urging us to submit this note, and to the anonymous referees for their helpful reports.

Thanks also to Dan Friedman, who collaborated on a preliminary version of this paper, and to Dave Herman, for his close reading.

References

1. Abadi, M., Cardelli, L., Curien, P.-L., Lévy, J.-J.: Explicit substitutions. J. Funct. Program. **1**(4), 375–416 (1991) Summary in ACM Symposium on Principles of Programming Languages (POPL), San Francisco, California, 1990
2. Ager, M.S., Biernacki, D., Danvy, O., Midtgaard, J.: A functional correspondence between evaluators and abstract machines. In: Proceedings of the Fifth ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming, pp. 8–19 (2003)
3. Crégut, P.: An abstract machine for the normalization of λ-terms. In: Proc. 1990 ACM Symposium on Lisp and Functional Programming, pp. 333–340 (1990)
4. Hannan, J., Miller, D.: From operational semantics to abstract machines: preliminary results. In: Proc. 1990 ACM Symposium on Lisp and Functional Programming, pp. 323–332 (1990)
5. Krivine, J.-L.: Un interpréteur du lambda-calcul. Unpublished note, at http://www.pps.jussieu.fr/~krivine/articles/interprt.pdf (1985)
6. Leroy, X.: The ZINC experiment: an economical implementation of the ML language. Technical Report 117, INRIA (1991)
7. Randell, B., Russell, L.J.: Algol 60 Implementation. Academic, New York (1964)
8. Streicher, T., Reus, B.: Classical logic, continutation semantics and abstract machines. J. Funct. Program. **8**(6), 543–572 (1998)