Exploiting Tierney's new shared library mechanism

B. Narasimhan
Department of Statistics
Stanford University
Stanford, CA 94305

Draft of July 1, 1998

Contents

Abstract

A very flexible shared library mechanism has been introduced by Tierney in the new release of `Lisp-Stat`. This is a document in progress that records some of my experiments in getting the best of both the `Lisp-Stat` and C worlds.

1 Introduction

There are times when one wishes to avoid the overhead associated with lisp functions and lisp data types. In some programs I have written, for example, there is a dire need for speed in dynamic graphic computations. Recent releases of `Lisp-Stat` have introduced many features, among them are the new shared library mechanism and support for `C-LONG` and `C-DOUBLE` arrays. In this document, I record some of my experiments in exploiting these features. I must confess there might be pitfalls that I am not aware of. Let me know if you spot any.

2 Avoiding some overhead with calling C functions

In [1] and [2] Tierney describes how `call-cfun` actually works. To glue the C program and `Lisp-Stat` together the arguments passed to the C routine are actually coerced into either C `long` or C `double` sequences. Then copies of the sequences are passed to the actual routine. In some cases, it might be desirable to get rid of this coercing and copying overhead *provided the programmer takes care to ensure that all requirements are met.* Such a routine could then be used to great effect when these arrays are stored in data slots in an object. Of course, it is assumed then that such an object will not be garbage-collected.

In my applications, I have need for manipulating large arrays both in C and `Lisp-Stat`. In order to have both C and `Lisp-Stat` share the same data, I previously used C programs based on `Xlisp` internals to manage this. There were some drawbacks to this approach.

1. The header file `xlisp.h` had to be included and the poor user had to know where to find it.
2. There could conceivably have been some flags used in compiling `Lisp-Stat` that would have to be replicated in compiling the C programs to be safe.
3. If any of the `Lisp-Stat` internals changed, the programs might bomb.
4. Platform specific issues might intrude in a big way.

With the modern mechanism, it appears that all these problems can be elegantly avoided. I present an illustrative example below.

2.1 An example

Consider the following C function which basically prints the contents of an array.

2a ⟨*C Routine FOO* 2a⟩≡

```
#include <stdio.h>

void foo(n, x)
     int *n;
     double *x;
{
  int i;

  for (i = 0; i < *n; i++) {
    printf("x[%d] is %f\n", i, x[i]);
  }
}
```

Now suppose I have an array in `Lisp-Stat` of type `C-DOUBLE` that I wish to pass to this function *without the usual copying overhead* associated with `call-cfun`, how could I do it? In other words, how could I "share" the same data in C and `Lisp-Stat`?

The following modified function of Tierney helps.

2b ⟨*Call by reference oldcfun.lsp* 2b⟩≡ (3b)

```
(defun call-by-reference-oldcfun (name lib &rest args)
  "Applies function NAME from shared library handle LIB with arguments.
All compound data are passed by reference but simple ones are not."
   (let* ((fun-addr (shlib::shlib-symaddr lib name))
          (argvecs (mapcar #'lisp-to-arg-if-not-compound-data args))
          (arg-addrs (mapcar #'array-data-address argvecs)))
     (apply #'shlib::call-by-address fun-addr arg-addrs)))
```

Defines:
`call-by-reference-oldcfun`, used in chunk 3.

The helper routine `lisp-to-arg-if-not-compound-data` is needed.

3a ⟨*Helper routine* 3a⟩≡ (3b)

```
(defun lisp-to-arg-if-not-compound-data (x)
   (if (compound-data-p x)
       x
      (if (integerp x)
          (coerce (list x) '(vector c-long))
        (coerce (list x) '(vector c-double)))))
```

Defines:
`list-to-arg-if-not-compound-data`, never used.

For extraction here is a package.

3b ⟨*Call by Reference Package* 3b⟩≡

```
(defpackage "CALL-BY-REFERENCE" (:use "XLISP"))
(in-package "CALL-BY-REFERENCE")
⟨Helper routine 3a⟩
⟨Call by reference oldcfun.lsp 2b⟩
(export
        '(call-by-reference-oldcfun))
```

Uses `call-by-reference-oldcfun` 2b.

Here then is an example session after creating the shared library `libfoo.so`.

3c ⟨*Example* 3c⟩≡

```
(require "call-by-reference")
(use-package "CALL-BY-REFERENCE")
(def lib (shlib::shlib-open "./libfoo.so"))
(def n 24)
(def x (make-array '(2 3 4) :initial-contents (iseq 23 0)
                   :element-type 'c-double))
(call-by-reference-oldcfun "foo" lib n x)
```

Uses `call-by-reference-oldcfun` 2b.

List of code chunks

This list is generated automatically. The numeral is that of the first definition of the chunk.

⟨*C Routine FOO* 2a⟩
⟨*Call by reference oldcfun.lsp* 2b⟩
⟨*Call by Reference Package* 3b⟩
⟨*Example* 3c⟩
⟨*Helper routine* 3a⟩

Index

Here is a list of the identifiers used, and where they appear. Underlined entries indicate the place of definition. This index is generated automatically.

`call-by-reference-oldcfun`: 2b, 3b, 3c
`list-to-arg-if-not-compound-data`: 3a

References

[1] Luke Tierney. *LISP-STAT: An Object-oriented Environment for Statistical Computing and Dynamic Graphics.* John Wiley & Sons (New York, Chichester), 1990.

[2] Luke J. Tierney. Shared libraries for xlisp-stat. URL `http://www.stat.umn.edu/~luke/xls/projects/`, 1998.