PLN CDR draft: Issue 2

Gleefre

July 6, 2024

1 Issue 2 (PRINT-READ consistency)

1.1 Description

The Lisp reader uses `find-package` when reading a symbol, which is affected by the *local nicknames* of the *current package*. That means that to maintain **print-read** consistency when printing a symbol, a good *package prefix* must be used - such that calling `find-package` on it in the *current package* returns the *home package* of the symbol.

There are several situations to consider:

1. The symbol is *apparently uninterned*.

   In this case it is printed without the package prefix, using the uninterned symbol syntax `#:`.

2. The symbol is accessible in the *current package*.

   In this case it is printed without any package prefix.

3. The *name* or one of the *global nicknames* of the *home package* of the symbol is not shadowed by any *local nickname* defined in the *current package*.

   In this case that name or global nickname might be used as the package prefix.

4. There exists a *local nickname* defined in the *current package* for the *home package* of the symbol.

   In this case that local nickname might be used as the package prefix.

5. Both the *name* and all *global nicknames* of the *home package* of the symbol are shadowed by *local nicknames* of the *current package*, and there is no *local nickname* defined in the *current package* for the *home package*.

   It is not clear how the symbol is printed, see PROPOSALS.

1.2 Examples

```
(defpackage #:foo
  (:use)
  (:export #:+))

(defpackage #:bar
  (:use #:cl)
  (:local-nicknames (#:foo #:cl)))
```

```
(let ((*package* (find-package '#:bar)))
  (print 'foo:+))
; >> FOO:+  (sbcl, ccl, ecl, acl, abcl, clasp, lispworks)

;; In the package #:BAR symbol FOO:+ refers to CL:+

(defpackage #:foo-a (:use) (:export #:quux))
(defpackage #:foo-b (:use) (:export #:quux))

(defpackage #:bar
  (:use)
  (:local-nicknames (#:foo-a #:foo-b)
                    (#:foo-b #:foo-a)))

(let ((*package* (find-package '#:bar)))
  (print 'foo-a:quux))
; >> FOO-B:QUUX  (sbcl, ccl, abcl, lispworks)
; >> FOO-A:QUUX  (ecl, acl, clasp)

;; In the package #:BAR symbol FOO-A:QUUX refers to FOO-B:QUUX
```

1.3 Current behavior

sbcl, ccl, abcl, lispworks: When possible, a *local nickname* is used as a package prefix.

ecl, acl, clasp: *Local nicknames* are never used as a package prefix.

1.4 Proposal SHARPSIGN-DOT

In the 5th case the symbol is printed using the `#.` syntax:

```
#.(cl:let ((cl:*package* (cl:find-package "KEYWORD")))
    (cl:find-symbol "BAR" "FOO"))
;; or
#.(cl:let ((cl:*package* (cl:find-package "KEYWORD")))
    (cl:intern "BAR" "FOO"))
```

If `*read-eval*` is *false* and `*print-readably*` is *true*, an error of type `print-not-readable` is signaled.

1.4.1 Note

Since `#:KEYWORD` cannot be used as a *local nickname*, and no *local nicknames* can be defined in the `#:KEYWORD` package, this expression is guaranteed to evaluate to the symbol in the correct package.

1.5 Proposal SHARPSIGN-COLON

In the 5th case the symbol is printed using the *extended `#:` syntax*:

```
#:(package name)
#::(package name)
```

Shinmera's idea.

1.6 Proposal SHARPSIGN-BACKQUOTE

In the 5th case the symbol is printed using the new `#‘` syntax for reading an expression ignoring *local nicknames* in the *current package*:

```
#‘foo:bar
#‘foo::bar
```

It can be implemented roughly as follows:

```
(defun |#‘-reader| (stream subchar arg)
  (declare (ignore subchar arg))
  (let* ((current-package *package*)
         (local-nicknames (package-local-nicknames current-package)))
    (loop for (nick . package) in local-nicknames
          do (remove-package-local-nickname nick current-package))
    (unwind-protect
      (read stream t nil t)
      (loop for (nick . package) in local-nicknames
            do (add-package-local-nickname nick package current-package)))))

(set-dispatch-macro-character #\# #\‘ #’|#‘-reader|)
```

It is *implementation-dependent* whether *local nicknames* are actually removed from the *current package* or not.

1.7 Proposal PRINT-UNREADABLY

In the 5th case the symbol is printed unreadably using the `#<` syntax:

```
#<SYMBOL IN THE SHADOWED PACKAGE FOO:BAR>
#<SYMBOL IN THE SHADOWED PACKAGE FOO::BAR>
```

(Specifics are *implementation-dependent.*)
If `*print-readably*` is *true*, an error of type `print-not-readable` is signaled.

1.8 Proposal THREE-FOUR-PACKAGE-MARKERS

In the 5th case the symbol is printed using the extended symbol token syntax:

```
foo:::bar   ; same as (cl:find-symbol "BAR" "FOO") in the #:KEYWORD package
foo::::bar  ; same as (cl:intern "BAR" "FOO") in #:KEYWORD package
```

1.9 Links

See CLHS 22.1.3.3.1 Package Prefixes for Symbols.