Advanced macros and their implementation

Matthew Might
University of Utah
matt.might.net

Agenda

- Syntax-rules
- Explicit-renaming
- Implementation

Why hygiene is good

- Two kinds of capture problem
- Expansion captured by context
- Context captured by expansion

Context captures

```
(let ((v 3)) (+ v 1))
=>
((lambda (v) (+ v 1)) 3)
```

Context captures

```
(let ((v 3)) (+ v 1))
=>
((lambda (v) (+ v 1)) 3)
```

```
(lambda (lambda)
(let ((v 3)) (+ v 1)))
```

Expansion captures

```
(or #f $tmp)
=>
(let (($tmp #f))
  (if $tmp $tmp $tmp))
```

Why hygiene is bad

Sometimes you **want** capture.

Example

```
(loop
  ...
  (exit)
  ...)
```

Example

```
(let ((else #f))
  (cond
    (else (begin (display "didn't get here!")
                 (newline)))))

(cond
  (else (begin (display "but got here!")
               (newline))))
```

Syntax-rules

Syntax-rules

- High-level pattern language
- Auto-hygienic -- no capture!

Syntax-rules

```
(define-syntax name
  (syntax-rules (keyword ...)
   ((_ pat ...)  template)
   ...))
```

Patterns

```
<pattern> ::=
    <name>
  | <constant>
  | (<pattern> ...)
  | (<pattern> <pattern> ... . <pattern>)
  | (<pattern> ... <pattern> <ellipsis>)
```

Templates

```
<template> ::=
    <name>
  | <constant>
  | (<element> ...)
  | (<element> <element> ... . <template>)

<element> ::=
    <template>
  | <template> <ellipsis>
```

Example:
Records/Structs

Low-level mixed hygiene: Explicit renaming

Explicit-renaming

```
(define-syntax name
  (explicit-renamer
   (lambda (exp rename compare)
    ...)))
```

Meaning

- exp -- the expression to transform
- `rename` -- hygienic renaming procedure
- `compare` -- hygienic symbol comparison

rename

- Takes a name, yields fresh name
- Behaves like a pure function (mostly)

`compare`

Purpose: Checks for macro-defined keywords.

Takes two names, and yields true iff
both names have the same meaning
in *the environment in which*
the output of the macro is expanded.

Names (or identifiers)

- A symbol is a name, but
- there are non-symbol names
- generated during expansion.

Example

```
(rename ‘foo) = [foo 17]
```

Example

```
(rename ‘[foo 17]) = [foo 21]
```

Example

```
(quote [foo 17]) = (quote foo)
```

Example: `let`

```
(lambda (exp rename compare)
  (let ((vars (map car (cadr exp)))
        (inits (map cadr (cadr exp)))
        (body (cddr exp)))
    `((lambda ,vars ,@body)
      ,@inits)))
```

Example: `let`

```
(lambda (exp rename compare)
  (let ((vars (map car (cadr exp)))
        (inits (map cadr (cadr exp)))
        (body (cddr exp)))
    `((,(rename 'lambda) ,vars ,@body)
      ,@inits)))
```

Classic example

```
(define-syntax loop
  (transformer
    (lambda (x r c)
      (let ((body (cdr x)))
        `(,(r 'call-with-current-continuation)
          (,(r 'lambda) (exit)
           (,(r 'let) ,(r 'f) () ,@body (,(r 'f)))))))))
```

Example: cond

```
(lambda (exp rename compare)
  (let ((clauses (cdr exp)))
    (if (null? clauses)
        `(,(rename 'quote) unspecified)
        (let* ((first (car clauses))
               (rest (cdr clauses))
               (test (car first)))
          (cond ((and (identifier? test)
                      (compare test (rename 'else)))
                 `(,(rename 'begin) ,@(cdr first)))
                (else `(,(rename 'if)
                        ,test
                        (,(rename 'begin) ,@(cdr first))
                        (cond ,@rest)))))))))
```

Implementation

Expansion

- `eval   : exp   env  -> s-exp`
- `expand : s-exp senv -> exp`

Environments

- `env  = name -> s-exp`
- `senv = name -> denotation`

Denotations

```
denotation = name + syntax-primitive + macro
```

Syntax primitives

All binding forms, e.g., `lambda` and `let-syntax`.

Mixed-hygiene macros

A mixed-hygiene macro is a **transcriber**.

Transcribers

A **transcriber** takes a
(1) syntactic form to rewrite;
(2) a local syntactic environment;
to produce:
(1) an expanded form
(2) an syntactic environment delta.

Transcribers

(macro s-exp senv) = (s-exp' Δsenv)

Expansion core

```
; expand : s-exp senv -> exp
(define (expand s-exp senv)
  (cond
    ((boolean? s-exp) s-exp)
    ((number?  s-exp) s-exp)
    ((string? s-exp)  s-exp)
    ((name? s-exp)    (senv-lookup senv s-exp))
    ((app? s-exp)     (apply-denotation (expand (app->fun s-exp) senv)
                                        s-exp senv))
    (else             (error "unknown expression type: " s-exp))))
```

Applying denotations

```
; apply-denotation : denotation app-exp senv -> senv
(define (apply-denotation denotation app-exp senv)
  (cond
    ((syntax-primitive? denotation)
     ((syntax-primitive->expander denotation) app-exp senv))

   ((syntax-transformer? denotation)
    (let* ((exp-senv   (expand-transformer denotation app-exp senv))
           ($exp       (car exp-senv))
           (senv-delta (cadr exp-senv)))
      (expand $exp (senv-divert senv senv-delta))))

   (else
    (cons denotation (map (expand-with senv) (app->args app-exp))))))
```

Syntax-rules
by example

Example: `test` macro

```
(define-syntax test
  (syntax-rules ()
    ((test ?thing ?proc ?else)
     (let ((temp ?thing))
       (if temp (?proc temp) ?else)))))
```

Example: `test` macro

```
(test (read if)
      (lambda (form)
        (write (list form temp) of))
      (newline of))
```

Example: `test` macro

```
(let ((temp (read if)))
  (if temp
      ((lambda (form) (write (list form temp) of))
       temp)
      (newline of))).
```

Example: `test` macro

```
([let 13] (([temp 13] (read if)))
  ([if 13] [temp 13]
           ((lambda (form) (write (list form temp) of))
            [temp 13])
           (newline of)))
```

Further reading

- Kohlbecker et al. “Hygienic macro expansion.”
- Kohlbecker et al. “Macro-by-example.”
- Rees. “Implementing lexically scoped macros.”
- Clinger & Rees. “Macros that work.”