Hygienic macro on Stak Scheme

@raviqqe

September 30, 2023

Contents

Hygienic macro

Overview

- A macro transcribes a source code into another source code.
- Hygienic macros must not:
  - Insert a binding that captures a reference not introduced by the macro itself.
  - Insert a reference that is captured by a binding not introduced by the macro itself.

Scheme

- You can capture free variables in macros.
  - Just like functions

Examples

Inserting a reference

Definition

```
(define x 42)

(define-syntax foo
  (syntax-rules
    ((foo)
      x)))
```

Use

```
(let ((x 13))
  (foo)) ; -> x, 42 but not 13
```

Examples

Inserting a binding

Definition

```
(define-syntax foo
  (syntax-rules
    ((foo x)
      ((lambda (y) x) 13))))
```

Use

```
(define y 42)

(foo y) ; -> ((lambda (y) y) 13), 42 but not 13
```

Implementation

- Based on "Macros That Work" by William Clinger
- With modifications for:
  - Global variables
  - Destructive update of syntactic environment

What to do?

- Track syntactic environment
  - What do variables denote on definitions and uses of macros?
- Expanding macros while preserving the hygienic invariants
  - Renaming variables introduced by macros

Implementation

Representation of syntactic environment

- The `environment` field is an association list from symbols to their denotations.

```
(define-record-type expansion-context
  (make-expansion-context environment)
  expansion-context?
  (environment expansion-context-environment expansion-context-set-environment!))
```

Implementation

Macro transformers

Definition

```
; (define-syntax foo (syntax-rules ...))
(define transformer
  (make-transformer definition-context macro-transformer-definition))

(define new-environment
  (environment-push environment name transformer))
```

Use

```
; (foo ...)
(transformer use-context expression)
```

Implementation

Expanding macros

- Rename free variables introduced by macros.
- Keep denotations on the use of macros.

```
(define (fill-template definition-context use-context matches template)
  (cond
    ((symbol? template)
      (let ((pair (assv template matches)))
        (if pair
          (cdr pair)
          (let (
              (name (rename-variable use-context template))
              (denotation (resolve-denotation definition-context template)))
            (when (denotation? denotation)
              (expansion-context-set! use-context name (denotation-value denotation)))
            name))))

    ; ...
```

Stak Scheme

- It had only the "poisonous" `syntax-rules` macro.
- Now, it's hygienic!
  - ~300 lines in total
    - `syntax-rules` pattern match
    - Hygienic macro definition and expansion
- Supports most of macro constructs from R7RS
  - `define-syntax`
  - `let-syntax`
  - `letrec-syntax`
  - `syntax-rules`

References

- BiwaSchemeにhygienic macroを入れる | 定期ミートアップ 第7回 yhara
- Macros That Work (a paper)
- Hygienic Macros Through Explicit Renaming
- 5.2 Hygienic macros | Gauche
- Hygienic macro | Wikipedia

Progress

New features

- Hygienic `syntax-rules`
- Quasi-quotation
- `read` and `write` procedures
- Ports and EOF objects

New features (continued)

- Symbol table GC
- `apply` procedure
  - Ribbit Scheme implemented it as a primitive.
  - Stak Scheme realizes it as an extension of calling convention in a VM.
    - Variadic arguments and parameters are symmetric.
    - e.g. Python, Ruby, and JavaScript

Next tasks...

- Record type
- `cond-expand`
- Self-hosting

Summary

- Building hygienic macros is fun.