CSE 341:
Programming Languages

Spring 2018
Racket — Macros

Topics

- What are macros and what do they mean?
  - Why do they have a bad reputation?
- Racket's macro system and hygiene
  - Free variables in macros
  - Bound variables in macros
  - Why hygiene is usually what you want
- What macros are good and not good for

Macros

To oversimplify, a macro is just a rule for rewriting programs as a prepass to evaluation. So it's very syntactic.

The "level" at which macros are defined affects their usefulness.

- "Sublexical" e.g.: Replace `car` with `hd` would turn `cart` into `hdt`.
  - No macro system does this; so macro-expander must know how to break programs into tokens.
- "Pre-parsing" e.g.: Replace `add(x,y)` with `x + y` (where x and y stand for expressions) would turn `add(x,y) * z` into `x + y * z`.
  - Some macro systems are this "dumb" (i.e., token-based); macro writers use more parens than Racketeers.
- "Pre-binding" e.g.: Replace `car` with `hd` would turn `(let* ([hd 0] [car 1]) hd)` into `(let* ([hd 0] [hd 1]) hd)`.
  - Few macro systems let bindings shadow macros; Racket does

The bad news

- Macros are very hard to use well.
- Most macro systems are so impoverished they make it harder.
- Actual uses of macros often used to ameliorate shortcomings in the underlying language.

But:

- Macros have some good uses
- Racket has a very sensible, integrated macro system
- So let's do macros justice for the day.

Hygiene

A "hygienic" macro system:

- Gives fresh names to local variables in macros *at each use* of the macro

- Binds free variables in macros *where the macro is defined*

Without hygiene, macro programmers:

- Get very creative with local-variable names

- Get creative with helper-function names too

- Try to avoid local variables, which conflicts with predictable effects

Why macros

Non-reasons:

- Anything where an ordinary binding would work just as well.

- Including manual control of inlining.

Reasons:

- Cosmetics

- "Compiling" a domain-specific language
  - But error messages a tough issue

- Changing evaluation-order rules
  - Function application will not do here

- Introducing binding constructs
  - A function here makes no sense