Module 7b: Local

CPSC 110

Peyton Seigo

2018-10-20

Module 7b: Local

Learning goals

- Write well-formed local expressions
- Diagram lexical scoping on top of expressions using local
- Hand-evaluate local expressions
- Use local to encapsulate function definitions (“private” helper functions)
- Use local to avoid redundant computation

With regards to efficiency, it is easy for programmers to worry

- too much,
- too soon,
- or incorrectly.

Better to design a simple program that’s easy to understand and change. Worry about efficiency later.

Local

The `local` function is comprised of *local definitions* and a *body*.

- Must have 0 or more definitions inside square brackets `[...]`
- Must have a body

```
(local [<definition 1> ; local definitions
        <definition 2>
        ...]
  <expression>)        ; body
```

Local Evaluation Rules

Three steps happen at the same time when evaluating a local expresson.

1. Renaming
   - rename definitions and all references to definitions
   - the new name must be globally unique
2. Lifting
   - lift renamed definitions out of the `local`, into top-level scope (not just out of the expression!)
3. Replace entire `local` with renamed body

    - After evaluation, the `local` expression is gone!

See `02-evaluation-rules.rkt` for a step-by-step evaluation.

Encapsulation

Finding good candidates for encapsulation:

1. One function has 1+ helpers closely linked to it
2. Outside program only cares about the main function, not the helpers

When refactoring existing code, make sure to

- Encapsulate
  - Wrap in new function (include necessary params)
  - Wrap old functions in `[]`
  - "Trampoline" call
  - Write one `@template` tag with `encapsulated`:
    * `(@template <Type1> <Type2> ... <TypeN> encapsulated)`
- Renaming
  - `check-expect`s
  - Stubs
  - `@HtDF` tag
- Delete unnecessary pieces
  - Delete tests for hidden functions (drawback: may lose some base cases)
  - Delete signatures that don't apply anymore
  - Delete old stubs

Structure changes. Functionality does NOT change.

Advantages and Disadvantages of Using `local` for Encapsulation

Advantages:

- Templates can be pre-encapsulated, saving time later on
- Template functions inside `local` don't have to be renamed! That's right, you can keep it as `fn-for-element` and `fn-for-loe` (for example).

Disadvantages:

- Cannot write base case tests for helper functions
  - Can only test the whole function
  - However, at this point in the course, we may not need to actually test the absolute base case test first

Terminology

- **Top-level definition**: definition visible to entire program
- **Local definition**: definition restricted to a certain scope
  - Within that scope, a local definition has precedence over any higher-level definitions
- **Lexical scoping**
  - **Scope contours**: boxes drawn around parts of the programming illustrating scopes
  - **Top-level scope**: global scope; scope of the whole program
  - Scope can be imagined as a bunch of nested boxes, or a tree where the top-most node is the global scope and each subnode is a scope within that scope.
- **Encapsulation**: bundling data with functions that operate on that data
- **Refactoring**: changing a program's code/structure without changing the program's behaviour
- **Namespace Management**: way to deal the problem of large programs inevitably using the same names
  - encapsulating many functions away so that the only public functions are ones with unique, descriptive names
  - ensuring other programmers don't call functions they're not supposed to