Carp

A Language for the 21st Century

Veit Heller

September 11, 2018

Port Zero

whoami

`whoami`

- PL nerd
- CTO @ Port Zero
- Carp standard library maintainer
- Secretly a turtle

man carp

man carp

- a Lisp-1
- type-inferred
- borrow-checked
- compiles to C
- for realtime applications

`man carp`

- a Lisp-1
- type-inferred ⇒ statically typed, at no extra charge
- borrow-checked ⇒ no GC, at not extra charge
- compiles to C
- for realtime applications

`whence -v carp`

- Haskell implements a Hindley-Milner type system and inference
  - $\Rightarrow$ You don't have to spell types out!
- Rust implements borrow checking
  - $\Rightarrow$ You don't have to manually manage memory, even without a GC!

`whence -v carp`

Let’s put those things together (after simplifying) and rejoice!

$\Rightarrow$ Also add some Lisp macro goodness and a near-seamless C FFI for good measure!

source carp

source carp

```
; (type f)
; f : (Fn [(Ref (Array a)), Int, Int] a)
(defn f [x y z]
  @(Array.nth x (* y z)))
```

Listing 1: A silly Carp function

`look hashmap`

Carp has a typed (but generic) hashmap/dictionary type.

`look hashmap`

It is not a builtin type.

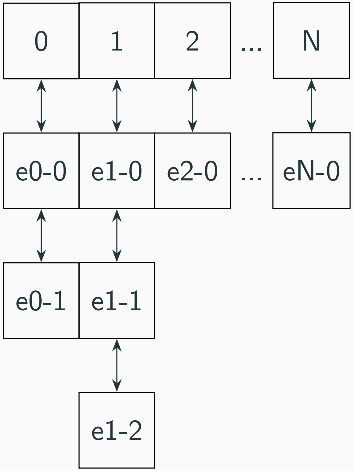
`look hashmap`

Let's briefly look at a simple hashmap implementation

- A hash function determines the placement of an element in an array of arrays.
- We append the element to the array inside the other to deal with hash collisions.
- Lookup combines hashing and a linear search.

look hashmap

Figure 1: A bucketed hashmap.



look hashmap

```
(deftype (Map a b) [buckets (Array (Array (Pair a b)))])
```

Listing 2: The hashmap type, simplified.

look hashmap

```
(defmodule Map
   (def dflt-len 256)

    (defn create []
       (init (Array.repeat dflt-len Array.zero)))

    (defn put [map key value]
      ; ...
    )
)
```

Listing 3: The hashmap module, with omissions.

look hashmap

```
(defn put [map key value]
  (let [buckets (buckets map)
        len (Array.length buckets)
        idx (Int.mod (hash key) len)
        bucket @(Array.nth buckets idx)
        pair (Pair.init @key @value)
        new-bucket (Array.push-back bucket pair)
        new-buckets (Array.aset @buckets
                                idx
                                new-bucket)
       ]
    (set-buckets @map new-buckets)))
```

Listing 4: Defining put.

open demo.live

exit

`trap`

Carp is early stage software.

- ⇒ Small community, few packages
- ⇒ We’re less than a handful of maintainers
- ⇒ Insufficient documentation
- ⇒ May change under your feet
- ⇒ May blow up in your face!

We’re approaching the first stable release (0.3)

`trap`

Full disclosure: At runtime, there are no lists.

`trap`

Kiss `car`, `cdr`, `quote` and `eval` goodbye.

`trap`

At macro expansion, you have business as usual... at the expense of type safety.

trap

```
(defmacro when [condition form]
  (list 'if condition form (list)))

(defmacro unless [condition form]
  (list 'if condition (list) form))
```

Listing 5: Conditionals as macros.

exit

Thank you!

Questions?

Slides at `https://github.com/hellerve/carp_talks`