<2020-10-10 Sa> Dr. Arne Babenhauserheide / draketo.de

Small Snippets with Wisp

Small snippets from my Wisp REPL.

Contents

1 Scheme overhead of records, lists and cons-pairs

If I have many lists of 16 elements, what's the overhead of records, lists and cons-pairs? This is adapted from cost-of-records that only looked at two-element structures.

Preparation:

```
;; 20 MiB res memory
import : srfi srfi-9 ;; records
         only (srfi srfi-1) fold
;; 37 MiB res memory
define-record-type <roll-result>
  roll-result a b c d e f g h i j k l m n o p
  . roll-result?
  . (a ra) (b rb) (c rc) (d rd) (e re) (f rf) (g rg) (h rh)
  . (i ri) (j rj) (k rk) (l rl) (m rm) (n rn) (o ro) (p rp)
;; 48 MiB res memory
define up-to-one-million : iota : expt 2 20
;; 55 MiB res memory
```

cons, records and lists added individually to avoid memory interaction:

```
define results-record : map (λ (x) (apply roll-result (iota 16 x)))
 ↪  up-to-one-million
;; 311 MiB res memory, diff: 256 MiB
```

```
define results-cons : map (λ (x) (fold cons x (iota 15 (+ x 1))))
 ↪  up-to-one-million
;; 440 MiB res memory, diff: 384 MiB
```

```
define results-list : map (λ (x) (apply list (iota 16 x)))
 ↪  up-to-one-million
;; 457 MiB res memory, diff: 402 MiB
```

Let's try a single vector (but filled with all zeros, for simplicity — I verified that there is no special handling for zero, using different numbers per Element gives the same result):

```
define 16-million-zeros-vector : make-vector 16000000 0
;; 179 MiB res memory, diff 124 MiB
```

Result: From cost-of-records we know that for two-element structures a cons-pair wastes the least amount of space. For 16 element structures however, record wins by a wide margin. For storing 16 million numbers this needs 256 MiB, 268435456 bytes, so each number needs **16.78 bytes**.

A plain vector with 16 million times 0 (zero) takes up 124 MiB, **8.13 bytes** per number, so if we use records to structure large amounts of data, we have to live with factor 2 overhead compared to packing all values into a single big vector and doing index-magic to retrieve the right values.

You can reduce this to 4.13 bytes per number by explicitly using a u32-vector, accepting the constrain on number-size: less than about 4.3 billion:

```
define 16-million-zeros-u32vector : make-u32vector 16000000 0
;; 118 MiB res memory, diff 63 MiB
```

A hash-table with 16 million x:x key-value pairs takes up 1.3 GiB, 87 bytes per pair.

2 2d6 + d10, all results

Calculate all possible results for rolling 2d6 and 1d10. This got a bit out of hand while I generalized it to arbitrary dice.

It is absolutely brute-force.

```
import : srfi srfi-1
define : roll-dice . dice
  . "Roll arbitrary DICE.

Each die is a list of its faces. Example: roll-dice '(1 2 3 4) '(1 2 3
↪  4)"
  define : roll mod . dice
    . "Roll DICE with modifier MOD. Example: 1d6+2 is roll 2 '(1 2 3 4 5
    ↪  6)"
    cond
       : null? dice
         . '()
       : null? : cdr dice
         map : λ (pip) : + pip mod
               car dice
       else
         apply append
             map : λ (pip) : apply roll : cons (+ pip mod) : cdr dice
               car dice
  apply roll : cons 0 dice


define : frequency results
    . "Count the frequency of numbers in the results"
    define counter : make-hash-table
    define : count value
        hash-set! counter value
             + 1 : if (hash-ref counter value) (hash-ref counter value)
              ↪  0
    map count results
    sort : hash-map->list cons counter
           λ (x y) : < (car x) (car y)

define d6 '(1 2 3 4 5 6)
define d10 '(0 1 2 3 4 5 6 7 8 9)
frequency : roll-dice d6 d6 d10
```

3 Fibers minimal producer and cooperating consumers

Requires Guile Fibers installed.

```
import (fibers) (fibers channels)

define c : make-channel

define : speaker
  define : put-and-yield msg
    ;; blocks until the message is received
    put-message c msg
    ;; allows other fibers to run, this is from (ice-9 threads)
    yield
  map put-and-yield
    iota 1000
  . #f ;; no result

define : writer1
  while #t
    ;; use only one display call to avoid re-ordering
    display : cons 'one (get-message c)
    ;; the newline could get re-ordered
    newline

define : writer2
  while #t
    display : cons 'two (get-message c)
    newline

run-fibers
  λ :
    spawn-fiber writer1
    spawn-fiber writer2
    speaker ;; blocks until the last message has been taken
            ;; then the program ends
```

[2021-10-12 Di]

4 roll xd10 keep y

```
set! *random-state*  : random-state-from-platform
import : only (srfi :1) take

define : d10
  1+ (random 10)

define : roll1d10-exploding
  let loop : : res (d10)
     if : zero? (modulo res 10) ;; explode
         loop : + res (d10)
         . res

define : rollxd10 count keep
   let loop : (results '()) (count count)
     if : zero? count
       ;; sum biggest KEEP results
       apply + : take (sort results >) keep
       loop (cons (roll1d10-exploding) results) {count - 1}
```

Equivalent Python-code:

```
import random

def rollxd10(count, keep):
    results = []
    for i in range(count):
        res = random.randint(1, 10)
        while (res % 10) == 0:
            res += random.randint(1, 10)
        results.append(res)
    results.sort()
    return sum(results[-keep:]) # last y results
```

[2022-08-11 Do]

5 Writing usable REST endpoints with Guile

At work I'm used to Spring endpoints that can be recognized by just looking at their annotation. But Spring uses lots of magic and in Scheme I want to keep my code more explicit.

Therefore I wrote simple tooling that provides me the most important feature without any magic: I want to define a handler that looks like this:

```
define-handler 'GET "/hello" : hello-world-handler request body
  ;; definition here
  ;; result:
  values
    build-response
      . #:headers `((content-type . (text/plain)))
    . "Hello World" ;; body
```

Method and path are at the beginning of the definition, easy to recognize at a glance. The implementation uses a simple definition of handlers (currently limited to PUT and GET, the rest will follow).

```
;; an alist of handlers: path-prefix . procedure
define put-handlers '()
define get-handlers '()
;; adding a handler
define : register-handler method path-prefix proc
    define : add-handler handlers
       cons (cons path-prefix proc) handlers
    cond
        : equal? method 'GET
          set! get-handlers : add-handler get-handlers
        : equal? method 'PUT
          set! put-handlers : add-handler put-handlers
        else #f
;; finding a matching handler
define : find-handler method path
    define handlers
      ` : GET . ,get-handlers
          PUT . ,put-handlers
    define : matching? handler-entry
      string-prefix? (car handler-entry) path
    define : find-proc handlers
      and=> (find matching? handlers) cdr
    and=> (assoc-ref handlers method) find-proc

;; define-handler provides syntactic sugar for the handler definition
define-syntax-rule : define-handler method path-prefix (name request
↪  body) rest ...
  begin
```

```
      define (name request body) rest ...
      register-handler method path-prefix name
      . name
```

A full server implementation:

```
import : only (srfi srfi-11) let-values
         only (srfi srfi-1) find
         prefix (fibers web server) fibers: ;; using
          ↪  https://github.com/wingo/fibers
         prefix (fibers channels) fibers:
         prefix (fibers) fibers:
         web client
         web request
         web response
         web uri

define : run-ipv4-fibers-server handler-with-path ip port
    fibers:run-server handler-with-path #:family AF_INET
      . #:port port #:addr INADDR_ANY

define : run-ipv6-fibers-server handler-with-path ip port
    define s
        let : : s : socket AF_INET6 SOCK_STREAM 0
            setsockopt s SOL_SOCKET SO_REUSEADDR 1
            bind s AF_INET6 (inet-pton AF_INET6 ip) port
            . s
    fibers:run-server handler-with-path #:family AF_INET6
      . #:port port #:addr (inet-pton AF_INET6 ip) #:socket s

{{{rest-handler-impl}}}
{{{rest-handler}}}

;; the server with handlers and a fallback
define : rest-server ip port
    define : handler-with-path request body
        define method : request-method request
        define path : uri-path : request-uri request
        define handler : find-handler method path
        if handler
            let-values : : (headers body) : handler request body
                values headers body
            values
```

```
                build-response
                  . #:headers `((content-type . (text/plain)))
                  . #:code 404
                . "404 not found"
    if : string-contains ip ":"
         run-ipv6-fibers-server handler-with-path ip port
         run-ipv4-fibers-server handler-with-path ip port
    . #f

;; start the server
rest-server "::" 8080
```

[2022-10-31 Mo]

6 fizzbuzz

Because I can :-)

```
import : ice-9 pretty-print

define : fizzbuzz n
  define : divisible m
    zero? : modulo n m
  define by3 : divisible 3
  define by5 : divisible 5
  cond
    : and by3 by5
      . 'Fizzbuzz
    by3 'Fizz
    by5 'Buzz
    else n

for-each pretty-print : map fizzbuzz : iota 15 1
```

[2022-11-10 Do]

7 Web-Scraping: find all links

Needs `htmlprag` from guile-lib and uses web client from Guile.

Find all links on a website.

```
import : only (htmlprag) html->shtml
         only (web uri) string->uri
         only (web client) http-get
         only (ice-9 pretty-print) pretty-print
         only (srfi :26) cut
         only (srfi :1) remove

define-values : resp body
    http-get "https://www.draketo.de/software/wisp-snippets.html"
define shtml : html->shtml body

define : find-tag shtml tagname
  let loop : (shtml shtml) (found '())
    cond
      : not : list? shtml
        . found
      : equal? tagname : car shtml
         cons shtml found
      else
        apply append : remove null? : map (cut loop <> found) shtml
pretty-print
  find-tag shtml 'a
```

```
GNU Guile 3.0.8
Copyright (C) 1995-2021 Free Software Foundation, Inc.

Guile comes with ABSOLUTELY NO WARRANTY; for details type `,show w'.
This program is free software, and you are welcome to redistribute it
under certain conditions; type `,show c' for details.

Enter `,help' for help.
((a (@ (accesskey "h") (href "../software.html"))
    " UP ")
 (a (@ (accesskey "H") (href "../")) " HOME ")
 (a (@ (href "../wissen.html")
       (class "category-tab tab-inactive tab-wissen"))
    "Wissen")
 (a (@ (href "../software.html")
       (class "category-tab tab-inactive tab-software"))
    "Software")
 (a (@ (href "../politik.html")
       (class "category-tab tab-inactive tab-politik"))
    "Politik")
```

```
(a (@ (href "../index.html")
      (class "category-tab tab-inactive tab-photo")
      (title "Startpage")
      (aria-label "Startpage"))
   "Â\xa0")
(a (@ (href "../anderes.html")
      (class "category-tab tab-inactive tab-anderes"))
   "Anderes")
(a (@ (href "../kreatives.html")
      (class "category-tab tab-inactive tab-kreatives"))
   "Kreatives")
(a (@ (href "../rollenspiel.html")
      (class "category-tab tab-inactive tab-rollenspiel"))
   "Rollenspiel")
(a (@ (href "http://www.draketo.de/english/wisp"))
   "Wisp")
(a (@ (href "wisp-snippets.pdf"))
   (img (@ (title "PDF")
           (src "../assets/pdf-thumbnail.png"))))
(a (@ (href "wisp-snippets.pdf")) "PDF")
(a (@ (href "#orgf13f96c"))
   "Scheme overhead of records, lists and cons-pairs")
(a (@ (href "#org008f1ea"))
   "2d6 + d10, all results")
(a (@ (href "#fibers-minimal"))
   "Fibers minimal producer and cooperating consumers")
(a (@ (href "#roll-xd10-keep-y"))
   "roll xd10 keep y")
(a (@ (href "#rest-endpoints"))
   "Writing usable REST endpoints with Guile")
(a (@ (href "#fizzbuzz")) "fizzbuzz")
(a (@ (href "https://www.draketo.de/proj/with-guise-and-guile/rpg-backend.html#cost-
   "cost-of-records")
(a (@ (href "https://www.draketo.de/proj/with-guise-and-guile/rpg-backend.html#cost-
   "cost-of-records")
(a (@ (href "https://github.com/wingo/fibers"))
   "Guile Fibers")
(a (@ (href "http://www.draketo.de/ich/impressum"))
   "Impressum")
(a (@ (href "http://gnu.org/l/gpl"))
   "GPLv3 or later")
(a (@ (href "https://creativecommons.org/licenses/by-sa/4.0/"))
   "cc by-sa"))
```

[2022-12-07 Mi]

8 pivot a table

```
apply map list '((1 2) (1 3))
```

```
((1 1) (2 3))
```

[2023-07-05 Mi]

9 Simple endpoint definition

If you want simple top-level endpoint definition in your backend as you're used to from annotations in Spring or Python (or ...) frameworks like the following, you can do that in under 64 lines.

```
define-handler 'GET "/health" : get-health-handler request body
   ...
```

Just define your handlers and add a simple syntax rule that selects from them:

```
;; an alist of handlers: path-prefix . procedure
define put-handlers '()
define get-handlers '()
define post-handlers '()
define patch-handlers '()
define delete-handlers '()
;; adding a handler
define : register-handler method path-prefix proc
    cond
        : equal? method 'GET
          set! get-handlers : cons (cons path-prefix proc) get-handlers
        : equal? method 'PUT
          set! put-handlers : cons (cons path-prefix proc) put-handlers
        : equal? method 'POST
          set! post-handlers : cons (cons path-prefix proc)
           ↪  post-handlers
        : equal? method 'PATCH
          set! patch-handlers : cons (cons path-prefix proc)
           ↪  patch-handlers
        : equal? method 'DELETE
```

```
          set! delete-handlers : cons (cons path-prefix proc)
           ↪  delete-handlers
        else #f
;; finding a matching handler
define : find-handler method path
    cond
        : equal? method 'GET
          and=>
            find : λ(x) : string-prefix? (car x) path
                 . get-handlers
            . cdr
        : equal? method 'PUT
          and=>
            find : λ(x) : string-prefix? (car x) path
                 . put-handlers
            . cdr
        : equal? method 'POST
          and=>
            find : λ(x) : string-prefix? (car x) path
                 . post-handlers
            . cdr
        : equal? method 'PATCH
          and=>
            find : λ(x) : string-prefix? (car x) path
                 . patch-handlers
            . cdr
        : equal? method 'DELETE
          and=>
            find : λ(x) : string-prefix? (car x) path
                 . delete-handlers
            . cdr
        else #f
;; define-handler provides syntactic sugar for the handler definition
↪  sugar
define-syntax-rule : define-handler method path-prefix (name request
↪  body) rest ...
  begin
      define (name request body) rest ...
      register-handler method path-prefix name
      . name

;; endpoint definitions with define-handler
```

You can now write your endpoints very naturally. For example the standard `/health` endpoint that Docker looks for:

```
define-handler 'GET "/health" : get-health-handler request body
        . "Health check.

        Endpoint: /health

        Example: GET /health
                 => OK"
        values
            build-response
                . #:headers `((content-type . (text/plain)))
            . "OK"
```

Then just use find-handler where you start the server, for example like this:

```
define : run-ipv4-server handler-with-path ip port
    run-server handler-with-path 'http `(#:host "localhost"
                                         #:family ,AF_INET
                                         #:addr ,INADDR_ANY
                                         #:port ,port)

define : run-ipv6-server handler-with-path ip port
    define s
        let : : s : socket AF_INET6 SOCK_STREAM 0
            setsockopt s SOL_SOCKET SO_REUSEADDR 1
            bind s AF_INET6 (inet-pton AF_INET6 ip) port
            . s
    run-server handler-with-path 'http `(#:family ,AF_INET6
                                         #:addr (inet-pton AF_INET6 ip)
                                         #:port ,port
                                         #:socket ,s)

define : run-server ip port
    define : handler-with-path request body
        define method : request-method request
        define path : uri-path : request-uri request
        define handler : find-handler method path
        if handler
            let-values : : (headers body) : handler request body
             ↪   wotstate
                values headers body
```

```
          values
              build-response
                . #:headers `((content-type . (text/plain)))
                . #:code 404
              . "404 not found"

  define ipv6 : string-contains ip ":"
  format : current-error-port
         if ipv6
             . "Started server on http://[~a]:~d\n"
             . "Started server on http://~a:~d\n"
         . ip port
  if ipv6
       run-ipv6-server handler-with-path ip port
       run-ipv4-server handler-with-path ip port
  . #f
```

[2025-02-02 So]