Significantly Abridged: Optimize for speed and safety in Common Lisp and Coalton through Gradual typing

Michal Atlas

March 27, 2023

Part I: The SBCL Type System[a]

[a] Kaplan-Ullman Flow Typing

A simple add

C

```c
int add1 (int x, int y)
{ return x + y; }
```

Lisp

```lisp
(defun add1 (x y)
  (+ x y))
```

The difference

C

```
int add1 (int x, int y)
{ return x + y; }

add1("hey", "ho");
```

---

warning: passing argument 1 of ‘add1’ makes integer from pointer without a cast

Lisp

```
(defun add1 (x y)
  (+ x y))

(add1 "hey" "ho")
```

```
; wrote ~/tmp/b.fasl

; compilation finished in 0:00:00.017
```

Adding types

```
(defun add1 (x y)
  (+ x y))

;; (FUNCTION (T T) (VALUES NUMBER &OPTIONAL))
```

What's in a type?

```
;   * (describe 'integer)
;
; COMMON-LISP:INTEGER
;   [symbol]
;
; INTEGER names the built-in-class #<BUILT-IN-CLASS COMMON-LISP:INTEGER>:
;   Class precedence-list: INTEGER, RATIONAL, REAL, NUMBER, T
;   Direct superclasses: RATIONAL
;   Direct subclasses: BIGNUM, FIXNUM
;   Sealed.
;   No direct slots.
;
; INTEGER names a primitive type-specifier:
;   Lambda-list: (&OPTIONAL (LOW (QUOTE *)) (HIGH (QUOTE *)))
```

Adding types to a function

```
(-> add1 (fixnum fixnum) fixnum)
(defun add1 (x y)
  (+ x y))

;; (FUNCTION (FIXNUM FIXNUM) (VALUES FIXNUM &OPTIONAL))
```

Nice errors

```
(add1 "Hey" "Ho")

;; debugger invoked on a TYPE-ERROR @5389A3AA in thread
;; #<THREAD "main thread" RUNNING {10010B81B3}>:
;;   The value
;;     "a"
;;   is not of type
;;     FIXNUM
;;   when binding X
                ; ^ It never even properly enters your function
```

Nice compile-time errors

```
(-> add1 (fixnum fixnum) fixnum)
(defun add1 (x y)
  (+ x y))

(defun foo ()
  (add1 "Hey" "Ho"))

; in: DEFUN FOO
;     (ADD1 "Hey" "Ho")
;
; note: deleting unreachable code
;
; caught WARNING:
;   Constant "Hey" conflicts with its asserted type FIXNUM.
```

Really nice errors

```
46            (permute (v)
47              (let ((n (length v))
48                    (c (make-array 9 :element-type 'character))
49                    (i 1))
50                (validate v)
51                (loop while (< i n) do
52                      (if (< (aref c i) i)
53                          (progn
54                            (let ((temp (
55                              (if (evenp
56                                  (progn
57                                    (setf
58                                    (setf
59                                  (progn
60                                    (setf
61                                    (setf
62                              (validate v)
63                              (incf (aref c
64                              (setf i 1))
65                          (progn
66                            (setf (aref c i) 0)
67                            (incf i))
68                          ))))
69            (powerset (vec)
70              (let ((i 1)
71                    (end (ash 1 (length vec))))
72                (loop while (< i end) do
73                      (let ((subv (make-array 0 :fill-pointer 0 :element-type 'base-char))
74                            (y 1))
75                        (loop for il across vec
76                              do (if (not (zerop (logand y i)))
77                                     (vector-push-extend il subv))
78                            (setf y (ash y 1)))
```

```
[sly] Derived type of
(SB-KERNEL:DATA-VECTOR-REF-WITH-OFFSET ARRAY
                                       (SB-KERNEL:CHECK-BOUND ARRAY
                                                              (ARRAY-DIMENSION
                                                               ARRAY 0)
                                                              SB-INT:INDEX)
                                       0)
is
  (VALUES CHARACTER &OPTIONAL),
conflicting with its asserted type
  REAL.
```

Some other types

There's a lot

arithmetic-error function simple-condition array generic-function simple-error atom hash-table simple-string base-char integer simple-type-error base-string keyword simple-vector bignum list simple-warning bit logical-pathname single-float bit-vector long-float standard-char broadcast-stream method standard-class built-in-class method-combination standard-generic-function cell-error nil standard-method character null standard-object class number storage-condition compiled-function package stream complex package-error stream-error concatenated-stream parse-error string condition pathname string-stream cons print-not-readable structure-class control-error program-error structure-object division-by-zero random-state style-warning double-float ratio symbol echo-stream rational synonym-stream end-of-file reader-error t error readtable two-way-stream extended-char real type-error file-error restart unbound-slot file-stream sequence unbound-variable fixnum serious-condition undefined-function float short-float unsigned-byte floating-point-inexact signed-byte vector floating-point-invalid-operation simple-array warning floating-point-overflow

It's all just sets

1/4 - Transformations

```
(defun foo (x)
  (declare (type (integer 35 45) x))
  (- x 15))

;; (FUNCTION ((INTEGER 35 45))
;;           (VALUES (INTEGER 20 30) &OPTIONAL))
```

2/4 - Merging

```
(defun foo (x y z)
  (if x
      (the (integer 0 20) y)
      (the (integer -20 0) z)))

;; (FUNCTION (T T T) (VALUES (INTEGER -20 20) &OPTIONAL))
```

2.5/4 - Serapeum Exhaustiveness Check

```
(deftype switch () '(member :on :off :broken))

(ecase-of switch :foo
  (:on 1)
  (:off 0))

; caught WARNING:
;   Non-exhaustive match: (MEMBER :ON :OFF) is a proper subtype of SWITCH.
;   There are missing types: ((EQL :BROKEN))

; The value
;   :FOO
; is not of type
;   COMMON-LISP-USER::SWITCH
```

3/4 - Narrowing

```
(defun foo (x)
  (if x
      (the (integer 0 20) 5)
      (the (integer -20 0) -3)))


;; (FUNCTION (T)
;;           (VALUES (OR (INTEGER 5 5) (INTEGER -3 -3)) &OPTIONAL))
```

4/4 - Satisfiability Types

```
(deftype prime ()
  '(satisfies prime-p))

(-> rsa (prime prime) *)
(defun rsa (q p)
  ...)
```

```
(rsa 7 20)

; debugger invoked on a TYPE-ERROR @538A0CBA in thread
; #<THREAD "main thread" RUNNING {10010B81B3}>:
;   The value
;     20
;   is not of type
;     (SATISFIES CL-MOD-PRIME:PRIME-P)
;   when binding P
```

The SBCL Optimizer

Generic Nastiness

```
; disassembly for ADD1
; Size: 37 bytes. Origin: #x5389A2EB                ; ADD1
; 2EB:       498B4D10         MOV RCX, [R13+16]     ; thread.binding-stack-pointer
; 2EF:       48894DF8         MOV [RBP-8], RCX
; 2F3:       488BD6           MOV RDX, RSI
; 2F6:       488BF8           MOV RDI, RAX
; 2F9:       FF142568050050   CALL [#x50000568]     ; #x52A00DC0: GENERIC-+
; 300:       488B45E8         MOV RAX, [RBP-24]
; 304:       488B75F0         MOV RSI, [RBP-16]
; 308:       488BE5           MOV RSP, RBP
; 30B:       F8               CLC
; 30C:       5D               POP RBP
; 30D:       C3               RET
; 30E:       CC10             INT3 16               ; Invalid argument count trap
```

The great SBCL optimizer

```
(declaim (optimize speed))
```

Optimizing advice

```
...
...
; note: forced to do GENERIC-+ (cost 10)
;       unable to do inline fixnum arithmetic (cost 2) because:
;       The first argument is a T, not a FIXNUM.
;       The second argument is a T, not a FIXNUM.
;       The result is a (VALUES NUMBER &OPTIONAL), not a (VALUES FIXNUM &OPTIONAL).
;       unable to do inline float arithmetic (cost 2) because:
;       The first argument is a T, not a SINGLE-FLOAT.
;       The second argument is a T, not a SINGLE-FLOAT.
;       The result is a (VALUES NUMBER &OPTIONAL), not a (VALUES SINGLE-FLOAT
;                                                          &OPTIONAL).
;       etc.
;
; compilation unit finished
;   printed 9 notes
```

No more calls

```
; disassembly for ADD1
; Size: 35 bytes. Origin: #x5389A9CC                ; ADD1
; CC:       498B4510        MOV RAX, [R13+16]       ; thread.binding-stack-pointer
; D0:       488945F8        MOV [RBP-8], RAX
; D4:       498D3438        LEA RSI, [R8+RDI]
; D8:       488BD6          MOV RDX, RSI
; DB:       48D1E2          SHL RDX, 1
; DE:       700A            JO L0
; E0:       488D1436        LEA RDX, [RSI+RSI]
; E4:       488BE5          MOV RSP, RBP
; E7:       F8              CLC
; E8:       5D              POP RBP
; E9:       C3              RET
; EA: L0:   CC52            INT3 82                 ; OBJECT-NOT-FIXNUM-ERROR
; EC:       1A              BYTE #X1A               ; RSI(s)
; ED:       CC10            INT3 16                 ; Invalid argument count trap
```

The greatly dangerous SBCL optimizer

```
;; Please never do this
(declaim (optimize (speed 3) (safety 0)))
```

WOAH!?!

```
; disassembly for ADD1
; Size: 9 bytes. Origin: #x534D9B96          ; ADD1
; 6:       4801FA            ADD RDX, RDI
; 9:       488BE5            MOV RSP, RBP
; C:       F8                CLC
; D:       5D                POP RBP
; E:       C3                RET
```

Containers

```
(defun get3 (x)
  (aref x 3))
;; (FUNCTION (T) (VALUES T &OPTIONAL))

; disassembly for GET3
; Size: 15 bytes. Origin: #x536C2706                    ; GET3
; 06:       BF06000000       MOV EDI, 6
; 0B:       FF7508           PUSH QWORD PTR [RBP+8]
; 0E:       FF2425300400 50  JMP [#x50000430]          ; SB-KERNEL:HAIRY-DATA-VECTOR-REF
```

Containers

```
(-> get3 ((simple-array number (4))) *)
(defun get3 (x)
  (aref x 3))
;; (FUNCTION ((SIMPLE-ARRAY NUMBER (4)))
;;           (VALUES NUMBER &OPTIONAL))

; disassembly for GET3
; Size: 10 bytes. Origin: #x53577F06              ; GET3
; 06:       488B5219         MOV RDX, [RDX+25]
; 0A:       488BE5           MOV RSP, RBP
; 0D:       F8               CLC
; 0E:       5D               POP RBP
; 0F:       C3               RET
```

Issues

The glaring optimization issue

```
(add1 "Hey" "Ho")
;=> 68771925391
```

The great SBCL optimizer

```
;; Please never do this
;; (declaim (optimize (speed 3) (safety 0)))
(declaim (optimize speed))
```

Ordering Issue

C

```c
void foo ()
{ add1("hey", "ho"); }

int add1(int x, int y)
{ return x + y; }
```

---

warning: implicit declaration of function ‘add1’

Lisp

```lisp
(defun foo ()
  (add1 "Hey" "Ho"))

(-> add1 (fixnum fixnum) fixnum)
(defun add1 (x y)
  (+ x y))
```

```
; compiling file "a.lisp":

; wrote /home/michal_atlas/tmp/a.fasl
; compilation finished in 0:00:00.025
```

Ordering Issue - Solved... ?

C

```c
int add1(int x, int y);

void foo ()
{ add1("hey", "ho"); }

int add1(int x, int y)
{ return x + y; }
```

Lisp

```lisp
(-> add1 (fixnum fixnum) fixnum)

(defun foo ()
  (add1 "Hey" "Ho"))

(defun add1 (x y)
  (+ x y))
```

```
; caught WARNING:
;   Constant "Hey" conflicts
;   with its asserted type FIXNUM.
```

This isn't ideal

Lisp is too flexible to be sure of anything

```
(restart-case
    (handler-bind ((error #'(lambda (c) (invoke-restart 'my-restart 7))))
      (+ "Foo." 0)) ;<= Type error
  (my-restart (&optional v) (princ "Nah, tis fine")))

;=> "Nah, tis fine"
```

How does Lisp solve problems?
??????????????????

How does Lisp solve problems?

METAPROGRAMMING!!!

Part II: Enter Coalton

Lovechild

Flexing on CL

CL

```
(defun add1 (x y)
  (+ x y))

;; (FUNCTION
;;   (T T)
;;   (VALUES NUMBER &OPTIONAL))
```

Coalton

```
(define (add1 x y)
  (+ x y))

;; ?????
```

Flexing on CL

CL

```
(defun add1 (x y)
  (+ x y))

;; (FUNCTION
;;   (T T)
;;   (VALUES NUMBER &OPTIONAL))
```

Coalton

```
(define (add1 x y)
  (+ x y))

;; ADD1 :: ∀ A. NUM A ⇒ (A → A → A)
```

Coalton

What do we lose?

- Only Homogenous Lists
- If branches have to be of the same type

What do we gain?

- Completely Static Typing
- Algebraic Data Types
- Currying

The Secret?

Coalton is Common Lisp

```
(coalton-toplevel
 (define (greet s)
         (concat "hello from coalton " s)))

(print (greet "Lang-talks")) ;=> "hello from coalton Lang-talks"
            ; ^ Still typechecked but at Lisp->Coalton boundary
```

1

[1] I've been leaving out package namespace shenanigans

Truly just a CL function

```
(describe 'greet)

; GREET names a compiled function:
;   Lambda-list: (S-68)
;   Derived type: (FUNCTION (STRING) (VALUES STRING &OPTIONAL))
;   Documentation:
;     GREET :: (STRING → STRING)
```

Calling Polymorphic functions

```
(coalton-toplevel
 (define (greet s)
         (+ s s)))

(print (coalton (greet 20)))
```

Polymorphic signatures

```
(describe 'greet)

; GREET names a compiled function:
;   Lambda-list: (DICT436 S-68)
;   Derived type: (FUNCTION (COALTON-LIBRARY/CLASSES::CLASS/NUM T)
;                           (VALUES T &OPTIONAL))
;   Documentation:
;     GREET :: ∀ A. NUM A ⇒ (A → A)
```

Calling Lisp from Coalton

```
(defun foo (n) ...)

(coalton-toplevel
 (define (bar f)
   (lisp Integer (f)
         (foo f))))
```

What about our problem from earlier?

```
(define (foo)
  (add1 "Hey" "Ho"))

(define (add1 x y)
  (+ x y))

;  error: Ambigious predicate
;  --> COALTON-TOPLEVEL (NIL):2:21
;   |
; 2 |                        (ADD1 "Hey" "Ho"))
;   |                         ^^^^ Ambigious predicate NUM STRING
```

And quite a lot more

```
(define (foo x) (> 20 x))

(foo "a")

;  error: Unable to codegen
;  --> COALTON (NIL):1:9
;   |
; 1 |  (COALTON (FOO "a"))
;   |           ^^^^^^^^^ expression has type ∀. (NUM STRING) => BOOLEAN
;   |                     with unresolved constraint (NUM STRING)
;   |           --------- Add a type assertion with THE to resolve ambiguity
```

Destructuring

```
(define (foo x)
  (let (Some y) = x)
  (trace y))

;; FOO :: ((OPTIONAL STRING) → UNIT)

(foo (Some "Hey"))
;=> Hey
```

Do & Monads

```
(define (bar x)
  (do (x <- x)
      (+ 10 x)))

;; BAR :: ∀ A B. (MONAD A) (NUM (A B)) ⇒ ((A (A B)) → (A B))

(bar (Some 6))
;=> #.(SOME 16)

(bar None)
;=> #.NONE
```

Beautiful STD

- Into
- Iterators
- Maps
- Trees
- Slices
- Tons of Types
- And much more

Quick Showoff before the end

Typeclasses and Higher Kinded Types anyone?

```
(define-type (Wrapper :m :a)
    (Wrap (:m :a)))

(define (repack present)
    (match present ((Wrap x) (pipe x transform Wrap))))

(define-class (Transform :m :t)
    (transform ((:m :a) -> (:t :a))))

(define-instance (Transform Optional List)
    (define (transform x)
        (match x ((Some x) (make-list x))
                 (None     (make-list)))))
```

Typedump 1/2

```
;; [package COALTON-USER]

;;   WRAPPER :: (* → *) → (* → *)

;; [package COALTON-USER]

;;   TRANSFORM :: ∀ A B C. TRANSFORM A C ⇒ ((A B) → (C B))
;;   REPACK :: ∀ A B C. TRANSFORM A C ⇒ (((WRAPPER A) B) → ((WRAPPER C) B))
;;   FANCY :: ∀ A. NUM A ⇒ (A → A)
;;   WRAP :: ∀ A B. ((A B) → ((WRAPPER A) B))
```

Typedump 2/2

```
;; [package COALTON-USER]

;;   [TRANSFORM ((A B) :: (* → * * → *))]
;;     TRANSFORM :: ((A C) → (B C))


;; [package COALTON-USER]

;;   [TRANSFORM ((A B) :: (* → * * → *))]
;;     TRANSFORM OPTIONAL LIST
```

It just works

```
(cl:print
 (coalton
  (the (Wrapper List Integer)
    (repack
     (Wrap (Some 20))))))

;=> #.(WRAP (20))
```

Thank you for your time