DIY Meta Languages with Common Lisp

Alexander Lier, Kai Selgrad, Marc Stamminger

Friedrich-Alexander University Erlangen-Nuremberg

European Lisp Symposium
April 4 2017, Brussels, Belgium

What to expect?

- experience made building C-Mera
- emerged hindrances
- pragmatic solving approaches
- a little of code
- collection of hints and clues

Meta Language

- design language?
- extend language?
- abstractions?
- macros!

Meta Language

- harness Lisp's power
- exploit the ease of S-Expressions
- mighty macro system for free

C-Mera

doowackadoodles.blogspot.de/2013/02/the-deadly-terrifying-chimera.html

https://github.com/kiselgra/c-mera

C-ish Target

```
#include <iostream>

int main(int argc, char *argv[])
{
    for(int i = 1; i < argc; ++i) {
        std::cout << "␣-␣" << argv[i] << std::endl;
    }
    return 0;
}
```

```
(include <iostream>)

(defmacro println (&rest args)
  `(<< #:std:cout ,@args #:std:endl))

(function main ((int argc) (char *argv[])) -> int
  (for ((int i = 1) (< i argc) ++i)
    (println "␣-␣" argv[i]))
  (return 0))
```

LGDV COMPUTER GRAPHICS • ERLANGEN

Lispy Input

```
(include <iostream>)

(defmacro println (&rest args)
  `(<< #:std:cout ,@args #:std:endl))

(function main ((int argc) (char *argv[])) -> int
  (for ((int i = 1) (< i argc) ++i)
    (println "␣-␣" argv[i]))
  (return 0))
```

Code Transition

```
(set foo (+ 1 2))
```

```
foo = 1 + 2;
```

AST Nodes

```lisp
(defclass infix-node ()
  ((operator :initarg :op)
   (member1  :initarg :lhs)
   (member2  :initarg :rhs)))
```

```lisp
(defmacro + (lhs rhs)
  `(make-instance 'infix-node
                  :op '+
                  :lhs ,lhs
                  :rhs ,rhs))
```

AST Generation

```lisp
(* (/ 1 2) (+ 3 4))
```

```lisp
(make-instance 'infix-node
               :op '*
               :lhs (make-instance 'infix-node
                                   :op '/
                                   :lhs 1
                                   :rhs 2)
               :rhs (make-instance 'infix-node
                                   :op '+
                                   :lhs 3
                                   :rhs 4))
```

AST Processing

```lisp
(defclass node ()
  ((values   :initarg :values)
   (subnodes :initarg :subnodes)))
```

```lisp
(defmethod traverser ((trav t) (node node))
  (with-slots (subnodes) node
    (loop for slot-names in subnodes do
      (let ((subnode (slot-value node slot-name)))
        (when subnode
          (traverser trav subnode))))))
```

Simple Traverser

```
(defclass debug-infix ())

(defmethod traverser ((_ debug-infix) (node infix-node))
  (format t "~a~%" (slot-value node 'op))
  (call-next-method)))
```

Lisp Symbols

```
(include <iostream>)

(defmacro println (&rest args)
  `(<< #:std:cout ,@args #:std:endl))

(function main ((int argc) (char *argv[])) -> int
  (for ((int i = 1) (< i argc) ++i)
    (println "␣-␣" argv[i]))
  (return 0))
```

COMPUTER GRAPHICS • ERLANGEN

Packages

```
(defpackage :cm-c
    (:use :common-lisp)
    (:shadow :+))

(in-package :cm-c)

(defmacro + (&rest _) ...)
```

```
(+ 1 1)
(cl:+ 1 1)
```

Lisp Scope

```
(defmacro lisp (&body body)
  `(macrolet ((+ (lhs rhs) `(cl:+ ,lhs ,rhs)))
     ,@body))
```

```
(+      1 2)     →   #<infix-node + 1 2>   →   1 + 2
(cm-c:+ 1 2)     →   #<infix-node + 1 2>   →   1 + 2
(cl:+   1 2)     →   3

(lisp
  (+      1 2)   →   3
  (cm-c:+ 1 2)   →   3
  (cl:+   1 2))  →   3
```

```
+   ≡   cm-c:+   →   cl:+
```

Escape from CL

```
(defpackage :swap (:use) (:export :+))
```

```
(defmacro swap:+ (lhs rhs)
  `(cm-c:+ ,lhs ,rhs))
```

```
(lisp
  (swap:+ 1 2))   →   3
```

```
swap:+   →   cm-c:+   →   cl:+
```

```
(defmacro swap:+ (lhs rhs)
  (macroexpand-1 `(cm-c:+ ,lhs ,rhs)))
```

LGDV COMPUTER GRAPHICS • ERLANGEN

Case

```
(include <iostream>)

(defmacro println (&rest args)
  `(<< #:std:cout ,@args #:std:endl))

(function main ((int argc) (char *argv[])) -> int
  (for ((int i = 1) (< i argc) ++i)
    (println "␣-␣" argv[i]))
  (return 0))
```

:preserve

```
(setf (readtable-case *readtable*) :preserve)

(DEFUN foo (a b) (+ b c))
(DEFUN bar (a b) (CL:+ a b))

(foo 1 (foo X y) (bar 1 2))
```

:invert

```
(format t "~a" 'foo)   →   FOO
(format t "~a" 'FOO)   →   FOO
(format t "~a" 'Foo)   →   FOO

(setf (readtable-case *readtable*) :invert)
(format t "~a" 'foo)   →   foo
(format t "~a" 'FOO)   →   FOO
(format t "~a" 'Foo)   →   Foo
```

:invert

```
(setf (readtable-case *readtable*) :invert)
(format t "~a" (intern "foo"))   →   FOO
(format t "~a" (intern "FOO"))   →   foo
(format t "~a" (intern "Foo"))   →   Foo
```

COMPUTER GRAPHICS • ERLANGEN

Addons

```
(include <iostream>)

(defmacro println (&rest args)
  `(<< #:std:cout ,@args #:std:endl))

(function main ((int argc) (char *argv[])) -> int
  (for ((int i = 1) (< i argc) ++i)
    (println "␣-␣" argv[i]))
  (return 0))
```

Reader Exploit

```
(set-macro-character #\* #'*-processor)
*argv    →   (dref argv)
```

```
     ++i // i++ // argv[i]
```

```
(set-macro-character #\Space   #'pre-process)
(set-macro-character #\Tab     #'pre-process)
(set-macro-character #\Newline #'pre-process)
```

LGDV COMPUTER GRAPHICS • ERLANGEN

Process Anything

```
(* ++a[4] --b[x++])
(* (aref (prefix++ a) 4) (aref (prefix-- b) (postfix++ x)))
```

```
(+ foo[baz[1]][2][3] &qox)
(+ (aref (aref (aref foo (aref baz 1)) 2) 3) (addr-of qox))
```

```
(set foo->bar->baz 5)
(set (pref (pref foo bar) baz) 5)
```

Except List Heads

```
(set-macro-character #\( #'pre-process-heads)
```

Except List Heads

```
(set-macro-character #\( #'pre-process-heads)
```

```
(set x (arr[i++]->foo))
(set x (funcall (pref (aref arr (postfix++ i)) foo)))
```

Namespaces

```
(include <iostream>)

(defmacro println (&rest args)
  `(<< #:std:cout ,@args #:std:endl))

(function main ((int argc) (char *argv[])) -> int
  (for ((int i = 1) (< i argc) ++i)
    (println "␣-␣" argv[i]))
  (return 0))
```

Special Dispatch

```
(set-dispatch-macro-character #\# #\: #'colon-reader)

(set #:N1::N2::var 4)
  →   (set (from-namespace N1 N2 var) 4)
  →   N1::N2::var = 4;
```

Example

```
(include <png++/png.hpp>)

(function main ((int argc) (char *argv[])) -> int
  (decl (((instantiate #:png:image (#:png::rgb-pixel)) (inImage argv[1]))
         (const unsigned int h = (inImage.get-height))
         (const unsigned int w = (inImage.get-width))
         ((instantiate #:png:image (#:png::gray-pixel)) (outImage h w)))

    (for ((size-t y = 0) (< y h) ++y)
      (for ((size-t x = 0) (< x w) ++x)
        (decl ((const #:png:rgb-pixel rgbPX = (inImage.get_pixel x (- h y 1)))
               (#:png:gray-pixel gPX = (+ (* 0.21f rgbPX.red)
                                           (* 0.72f rgbPX.green)
                                           (* 0.07f rgbPX.blue))))

          (outImage.set-pixel x (- h y 1) gPX))))
    (out-image.write argv[2]))
  (return 0))
```

```
7  $ cm c++ main.lisp -o main.cpp
8  $ g++ main.cpp -lpng
```

Result

```cpp
#include <png++/png.hpp>

int main(int argc, char *argv[])
{
    png::image<png::rgb_pixel> inImage(argv[1]);
    const unsigned int h = inImage.get_height();
    const unsigned int w = inImage.get_width();
    png::image<png::gray_pixel> outImage(h, w);
    for(size_t y = 0; y < h; ++y) {
        for(size_t x = 0; x < w; ++x) {
            const png::rgb_pixel rgbPX = inImage.get_pixel(x, h - y - 1);
            png::gray_pixel gPX = (2.10000000E-1f * rgbPX.red) +
                 (7.20000000E-1f * rgbPX.green) + (7.00000000E-2f * rgbPX.blue);
            outImage.set_pixel(x, h - y - 1, gPX);
        }
    }
    outImage.write(argv[2]);
    return 0;
}
```

LGDV
COMPUTER GRAPHICS • ERLANGEN

Go Meta

```
(macrolet ((img (type) `(instantiate #:png:image ((from-namespace png ,type)))))

  (function main ((int argc) (char *argv[])) -> int
    (decl (((img rgb-pixel) (inImage argv[1]))
           (const unsigned int h = (inImage.get-height))
           (const unsigned int w  = (inImage.get-width))
           ((img gray-pixel) (outImage h w)))

      (for ((size-t y = 0) (< y h) ++y)
        (for ((size-t x = 0) (< x w) ++x)
          (decl ((const #:png:rgb-pixel rgbPX = (inImage.get_pixel x (- h y 1)))
                 (#:png:gray-pixel gPX = (+ (* 0.21f rgbPX.red)
                                            (* 0.72f rgbPX.green)
                                            (* 0.07f rgbPX.blue))))

           (outImage.set-pixel x (- h y 1) gPX))))
    (out-image.write argv[2]))
    (return 0)))
```

Go Meta

```
(macrolet ((img (type) `(instantiate #:png:image ((from-namespace png ,type)))))
  (symbol-macrolet ((red 'green) ; or let
                    (blue 'red)
                    (green 'blue))

    (function main ((int argc) (char *argv[])) -> int
      (decl (((image rgb-pixel) (inImage argv[1]))
             (const unsigned int h = (inImage.get-height))
             (const unsigned int w  = (inImage.get-width))
             ((image gray-pixel) (outImage h w)))

        (for ((size-t y = 0) (< y h) ++y)
          (for ((size-t x = 0) (< x w) ++x)
            (decl ((const #:png:rgb-pixel rgbPX = (inImage.get_pixel x (- h y 1)))
                   (#:png:gray-pixel gPX = (+ (* 0.21f rgbPX.red)
                                              (* 0.72f rgbPX.green)
                                              (* 0.07f rgbPX.blue))))
             (outImage.set-pixel x (- h y 1) gPX))))
      (out-image.write argv[2]))
      (return 0))))
```

```
rgbPX.red   →   (oref rgbPX red)
```

Bound Symbols?

```
(defvar foo 1)
(boundp 'foo) ;; -> T
```

```
(let ((bar 1))
  (boundp 'bar)) ;; -> NIL
```

```
(labels ...)
(flet ...)
(macrolet ...)
(symbol-macrolet ...)
```

Variable Bound!

```
(defun vboundp! (variable &optional env)
  #+sbcl (sb-cltl2::variable-information variable env)
  #+clozure   (ccl::variable-information variable env)
  #+ecl   (or (boundp variable)
              (find variable (first env)
                :test #'(lambda (x y) (eql x (car y)))))
  #-(or sbcl clozure ecl) (error "..."))
```

COMPUTER GRAPHICS • ERLANGEN

Function Bound!

```lisp
(defun fboundp! (function &optional env)
  #+sbcl (sb-cltl2::function-information function env)
  #+clozure   (ccl::function-information function env)
  #+ecl   (or (fboundp function)
              (find function (rest env)
                :test #'(lambda (x y) (eql x (car y)))))
  #-(or sbcl clozure ecl) (error "..."))
```

xboundp

```
(defmacro xboundp (item &environment env)
  (if (or (fboundp! item env)
          (vboundp! item env))
      t      ; item bound
      nil)) ; item unbound
```

Application

```
(defun bar (a b) (cl:+ a b))
; (defun baz (a b) (cl:+ a b))
(defmacro qux (a b) `(+ ,a ,b))
(defmacro qox (a b) `(cl:+ ,a ,b))

(set foo (bar 1 2))
(set foo (baz 1 2))
(set foo (qux 1 2))
(set foo (qox 1 2))
```

```
foo = 3;
foo = baz(1, 2);
foo = 1 + 2;
foo = 3;
```

```
(set foo (funcall 'bar 1 2))
(set foo (funcall 'qux 1 2))
```

Conclusion

- pragmatic
- to our taste
- simple to hack
- collection of hints and clues

Thank you for your attention.

Meta

```
(decl ((float r = (prefiltered 0))
       (float g = (prefiltered 1))
       (float b = (prefiltered 2))
       (float X = (+ (* 0.5149f r) (* 0.3244f g) (* 0.1607f b)))
       (float Y = (/ (+ (* 0.2654f r) (* 0.6704f g) (* 0.0642f b)) 3.0f))
       (float Z = (+ (* 0.0248f r) (* 0.1248f g) (* 0.8504f b)))
       (float V = (* Y (- (* 1.33f (+ 1.0f (/ (+ Y Z) X))) 1.68f)))
       (float W = (+ X Y Z))
       (float luma = (+ (* 0.2126f r) (* 0.7152f g) (* 0.0722f b)))
       (float s = 0.0f)
       (float xl = (/ X W))
       (float yl = (/ Y W))
       (const float xb = 0.25f)
       (const float yb = 0.25f))
  (set xl (+ (* (- 1.0f s) xb) (* s xl))
       yl (+ (* (- 1.0f s) yb) (* s yl))
       Y  (+ (* V 0.4468f (- 1.0f s)) (* s Y))
       X  (/ (* xl Y) yl)
       Z  (- (/ X yl) X Y))
  (decl ((float rgb_r = (+ (* 2.562263f X) (* -1.166107f Y) (* -0.396157f Z)))
         (float rgb_g = (+ (* -1.021558f X) (* 1.977828f Y) (* 0.043730f Z)))
         (float rgb_b = (+ (* 0.075196f X) (* -0.256248f Y) (* 1.181053f Z))))
    (set (scotopic2 0) (fminf 255.0f (fmaxf 0.0f rgb_r)))
    (set (scotopic2 1) (fminf 255.0f (fmaxf 0.0f rgb_g)))
    (set (scotopic2 2) (fminf 255.0f (fmaxf 0.0f rgb_b))))))
```

Meta

```
const uchar4& vec4_473514 = prefiltered[i];
float r = vec4_473514.x;
float g = vec4_473514.y;
float b = vec4_473514.z;
float X = (5.14900000E-1f * r) + (3.24400000E-1f * g) + (1.60700000E-1f * b);
float Y = ((2.65400000E-1f * r) + (6.70400000E-1f * g) + (6.42000000E-2f * b)) / 3.00000
float Z = (2.48000000E-2f * r) + (1.24800000E-1f * g) + (8.50400000E-1f * b);
float V = Y * ((1.33000000E+0f * (1.00000000E+0f + ((Y + Z) / X))) - 1.68000000E+0f);
...
float rgb_r = (2.56226300E+0f * X) + (-1.16610700E+0f * Y) + (-3.96157000E-1f * Z);
float rgb_g = (-1.02155800E+0f * X) + (1.97782800E+0f * Y) + (4.37300000E-2f * Z);
float rgb_b = (7.51960000E-2f * X) + (-2.56248000E-1f * Y) + (1.18105300E+0f * Z);
//Prepare store variable: vec4_473778
float4 vec4_473778 = make_float4(0.00000000E-1f, 0.00000000E-1f, 0.00000000E-1f, 0.00000
vec4_473778.x = fminf(2.55000000E+2f, fmaxf(0.00000000E-1f, rgb_r));
vec4_473778.y = fminf(2.55000000E+2f, fmaxf(0.00000000E-1f, rgb_g));
vec4_473778.z = fminf(2.55000000E+2f, fmaxf(0.00000000E-1f, rgb_b));
//Store: vec4_473778 to (scotopic2)
scotopic2[i] = make_uchar4(((unsigned char)vec4_473778.x), ((unsigned char)vec4_473778.y
```

Meta

```
const __m256i xmm_472477 = _mm256_loadu_si256(((const __m256i*)&prefiltered[i]));
__m256 r = _mm256_cvtepi32_ps(_mm256_srli_si256(_mm256_slli_epi32(xmm_472477, 24)
__m256 g = _mm256_cvtepi32_ps(_mm256_srli_epi32(_mm256_slli_epi32(xmm_472477, 16)
__m256 b = _mm256_cvtepi32_ps(_mm256_srli_epi32(_mm256_slli_epi32(xmm_472477, 8),
__m256 X = _mm256_add_ps(_mm256_add_ps(_mm256_mul_ps(xmm_constant_0_5149__471431,
__m256 Y = _mm256_div_ps(_mm256_add_ps(_mm256_add_ps(_mm256_mul_ps(xmm_constant_0
__m256 Z = _mm256_add_ps(_mm256_add_ps(_mm256_mul_ps(xmm_constant_0_0248__471438,
__m256 V = _mm256_mul_ps(Y, _mm256_sub_ps(_mm256_mul_ps(xmm_constant_1_33__471441
...
__m256 rgb_r = _mm256_add_ps(_mm256_add_ps(_mm256_mul_ps(xmm_constant_2_562263__4
__m256 rgb_g = _mm256_add_ps(_mm256_add_ps(_mm256_mul_ps(xmm_constant__1_021558__
__m256 rgb_b = _mm256_add_ps(_mm256_add_ps(_mm256_mul_ps(xmm_constant_0_075196__4
//Prepare store variable: (xmm_472964 xmm_472965 xmm_472966 xmm_472967)
__m256 xmm_472964 = _mm256_setzero_ps();
__m256 xmm_472965 = _mm256_setzero_ps();
__m256 xmm_472966 = _mm256_setzero_ps();
__m256 xmm_472967 = _mm256_setzero_ps();
xmm_472964 = _mm256_min_ps(xmm_constant_255_0__471459, _mm256_max_ps(xmm_constant
xmm_472965 = _mm256_min_ps(xmm_constant_255_0__471459, _mm256_max_ps(xmm_constant
xmm_472966 = _mm256_min_ps(xmm_constant_255_0__471459, _mm256_max_ps(xmm_constant
//Store: (xmm_472964 xmm_472965 xmm_472966 xmm_472967) to (scotopic2)
const __m256i r_473021 = _mm256_cvtps_epi32(xmm_472964);
const __m256i g_473022 = _mm256_slli_si256(_mm256_cvtps_epi32(xmm_472965), 1);
const __m256i b_473023 = _mm256_slli_si256(_mm256_cvtps_epi32(xmm_472966), 2);
const __m256i a_473024 = _mm256_slli_si256(_mm256_cvtps_epi32(xmm_472967), 3);
const __m256i rg_473025 = _mm256_or_si256(r_473021, g_473022);
const __m256i ba_473026 = _mm256_or_si256(b_473023, a_473024);
const __m256i rgba_473027 = _mm256_or_si256(rg_473025, ba_473026);
_mm256_storeu_si256(((__m256i*)&scotopic2[i]), rgba_473027);
```