An introduction to compiler construction and functional programming through the implementation of GNU epsilon

An LCR seminar

LIPN - Universite Paris 13

Luca Saiu <positron@gnu.org> 2007-04-02

Who I am and what I do here

Introducing myself

- I come from Pisa...

I'm working on Marionnet with Jean-Vincent Loddo:

- Para-virtualization
- Networks
- Kernel programming
- GUIs in a functional language

Presentation outline

We're going to talk about...

- Functional programming in a practical context
- GNU epsilon: an overview of how it is now...
- ...an of how it will be
- (free software and the GNU Project)

Functional programming in practice

Now I could...

- explain λ–calculus, ω–order, types, syntax, semantics
- or just impress you with a demonstration

A practical demonstration

No time for showing you a compiler, unfortunately...

$dy/dx = 0$ (with $y \neq x$)

$dx/dx = 1$

$d(f(x) + g(x))/dx = df(x)/dx + dg(x)/dx$

$d(f(x) - g(x))/dx = df(x)/dx - dg(x)/dx$

$d(f(x) * g(x))/dx = df(x)/dx * g(x) + f(x) * dg(x)/dx$

$d(\sin(f(x)))/dx = \cos(f(x)) * df(x)/dx$

$d(\cos(f(x)))/dx = -\sin(f(x)) * df(x)/dx$

The demonstration

Luca Saiu <positron@gnu.org> 2007-04-02

epsilon: applications

As you have just seen epsilon **is** usable, but not polished yet.

Applications:

- Programming tools: `epsilonlex` and `epsilonyacc`
- Interpreters and compilers: BASIC, Lisp, Prolog, ...
- Simple graphic hacks
- ICFP Programming Contests: 2004 and 2005
- Some AI applications (games)

epsilon: features

Some features of the implementation you saw:

- Higher-order
- Static typing: type inference, parametric polymorphism
- Quite a lot of ground types
- Monadic I/O
- An half-decent library
- Relatively efficient

epsilon: history

Three implementations: 2001-2002, 2002-2006, 2006-

- ~60,000 lines of C code
- My MD thesis describes each implementation in (relative) detail

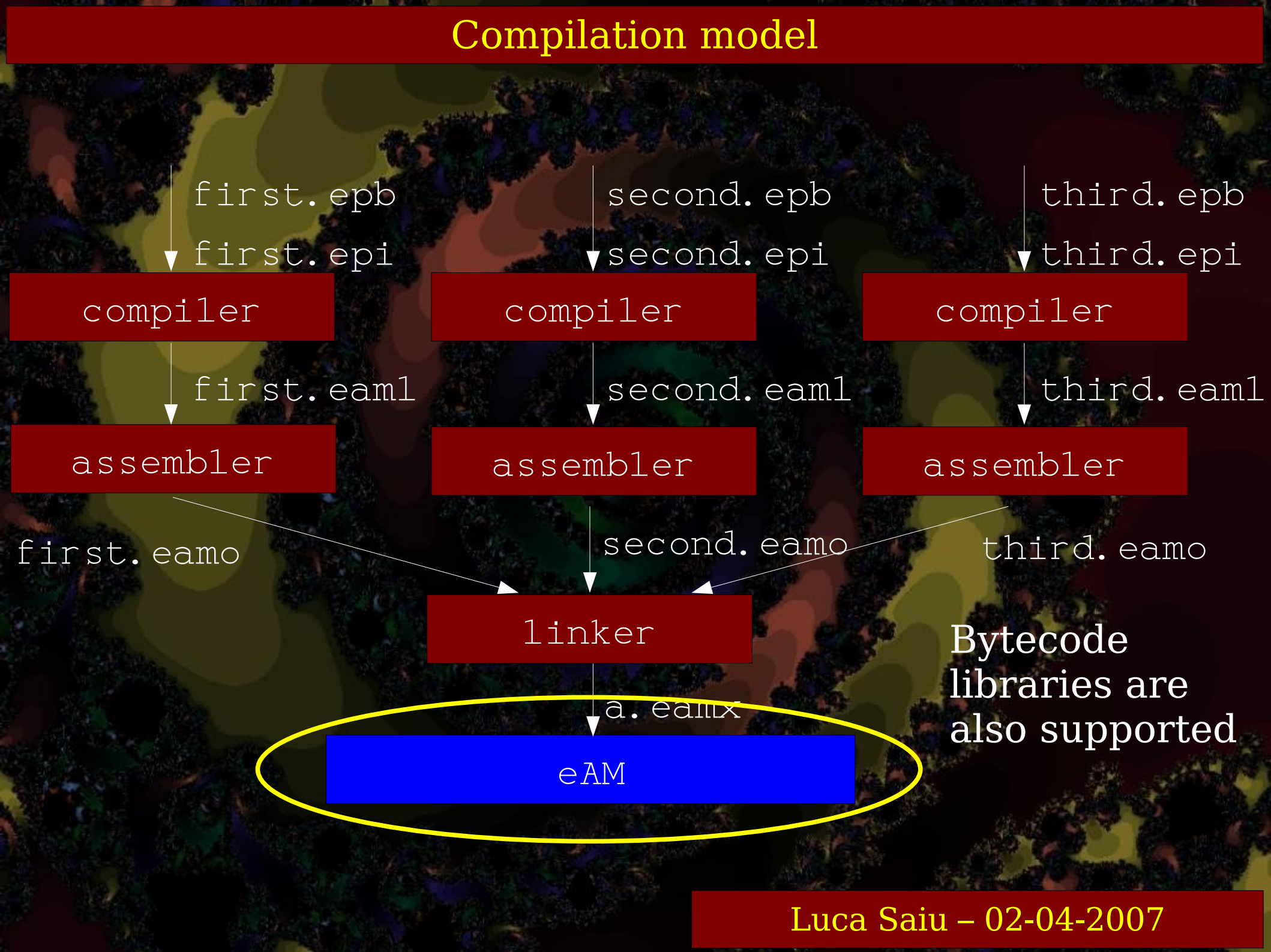
  (it's available on my home-page here at LIPN)
- You saw the second implementation

epsilon: the second implementation

- Compiler written in C
- Assembler from textual notation to bytecode
- Bytecode interpreter

Separate compilation is supported:

Compilation model



Second implementation (2002-2006): eAM

Two execution models:

This file may be **huge** (~megabytes)...

...which limits the solution's practicality

epsilon and GNU

- epsilon is part of the GNU Project

Free Software

A philosophical movement started by Richard M. Stallman in the early Eighties...

(Original author of Emacs, GCC, ...)

- ...its history is extremely interesting, but we have no time now (hint: contact me)
- Huge success: tens of thousands of programs, several whole operating systems...
- ...but first of all a movement based on ethical values

The four freedoms

- The freedom to use the software, for any purpose
- The freedom to modify it to suit your needs
- The freedom of distributing copies of it
“to help your neighbor”
- The freedom of distributing modified copies of it
“to help the community”
- Copyleft: the GNU General Public License

Free Software vs. Open Source

- Two **very** different movements:
ethical values vs. practical convenience

- ...but adherent of different movements often cooperate on practical projects

- Linus Torvalds and Eric Raymond have the spotlight these days. They advocate Open Source, not Free Software.

By the way, the operating system name is “GNU/Linux”; “Linux” is its kernel...

Luca Saiu <positron@gnu.org> 2007-04-02

The GNU Project

A project to build a whole operating system which is free software, compatible with Unix.

- Started in 1983, complete in the first Nineties, except for the kernel...
  the HURD
- ...Linux arrived to fill the last gap
- The system is mature, complete, efficient, solid
- ...but applications are always needed: the project grew more ambitious

Luca Saiu <positron@gnu.org> 2007-04-02

The GNU Project today

First of all, spreading the word about freedom

- A responsibility for other reasons: GNU software has also a fame of very high techincal quality
- Technical requisites: GNU projects must fit well together
- Uniform coding and documentation style
- Avoiding duplicate work
- Strategically important applications: making proprietary software obsolete
- Having a clear position in political battles: software patents, copyright on interfaces, DRM...

epsilon in GNU

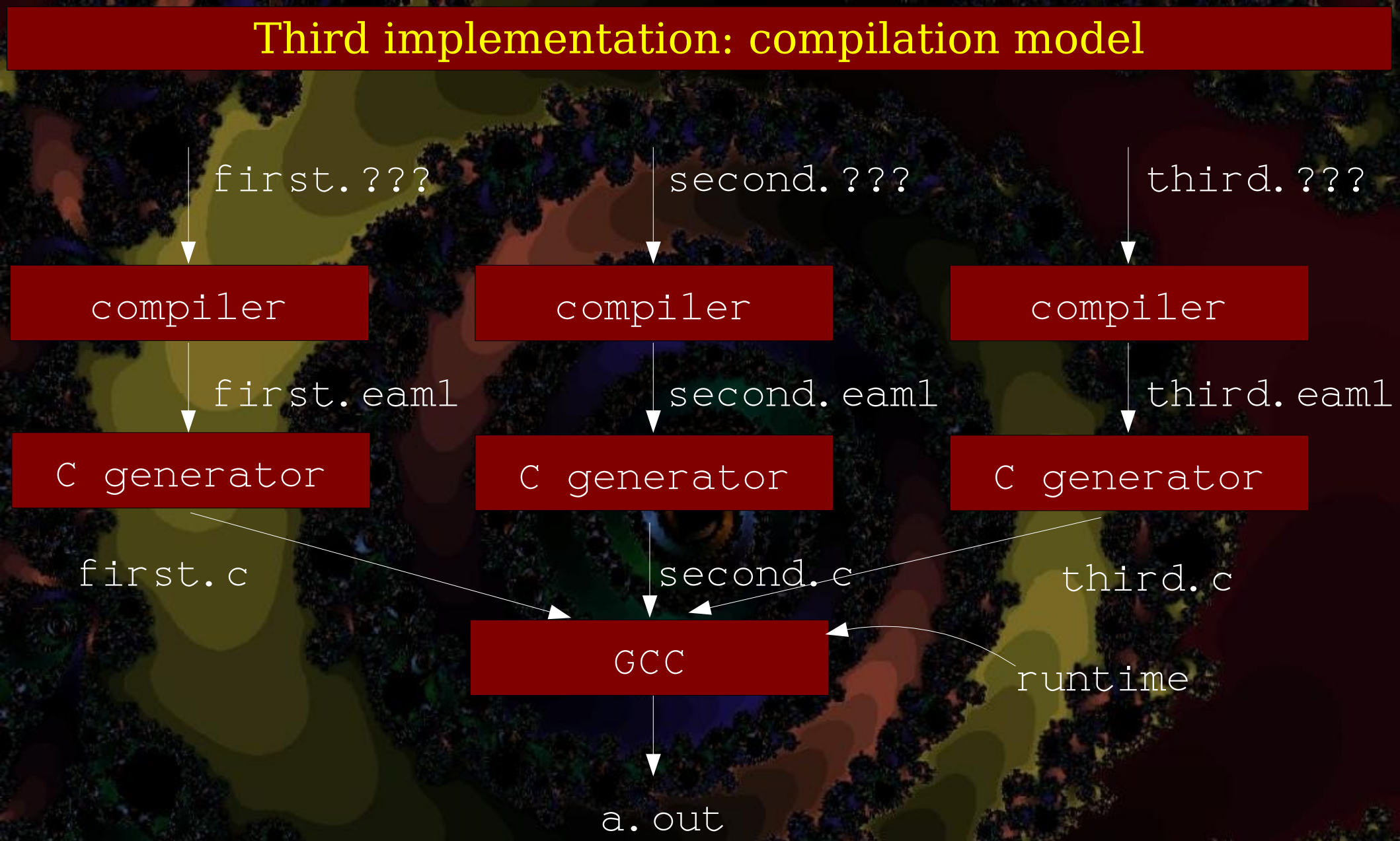
epsilon is (currently) a peripherical project within the system

- ...but no other statically-typed functional languages are in GNU
- it fits well with the rest of the system
- Extremely portable (x86, PowerPC, UltraSparc)
- Useful as an extension language, with Guile

eAM: third implementation (2006-)

- Bottom-up strategy
  A general language-neutral machine
  - The compiler isn't written yet
- Objectives:
  - Maximum Generality and efficiency
    (also compromising simplicity)
  - User-definable types and primitive operators, with
    C; zero overhead
  - Predefined general registers...
    and no other data structure
    - By default: no stack, no heap
    - No fixed calling convention

Third implementation: compilation model



Generating native code is now realistic.

Third implementation: notable aspects

- eAM registers to assembly registers:

Es.: x86

```
%r0 ──► %ebx
%r1 ──► %edi
%r2 ──► %esi
```

- Basic block as C procedures:

```
    %r1 := %r2
    goto target:
    ...
target:
    %r3 := %r1
    ...
```

```
continuation_t block1(void){
    REGISTER_1 := REGISTER_2;
    return target();
    ...
}

continuation_t target(void){
    REGISTER_3 := REGISTER_1;
    ...
}
```

...and “slow jumps” when ABIs forbid tail-call optimizations

New eAM: an extension example

Integers

```
exported primitive "+"
  arity 2
c-name
  "PRIMITIVE_PLUS"
c-declaration
{#define PRIMITIVE_PLUS(X, Y) \
   ((word_t)((integer_t)(X) + (integer_t)(Y))) }
c-implementation
{}
end primitive
```

Primitive definition

Primitive use in eAML

```
L:
    %r0 := +(%r2, %r3)
    %r1 := +(%r1, %r3)
    %r3 := +(%r3, %r2)
```

No stack and no garbage collector by default

New eAM: a micro-benchmark

MCD:

```
/* ... */
while(x != y){
    if(x < y)
      y -= x;
    else
      x -= y;
}
```

C

```
define mcd =
  fix \ mcd . \ x . \ y .
  if x = y then
    x
  else if x < y then
    mcd x (y - x)
  else
    mcd (x - y) y;
```

epsilon

```
mcd:
  if =(%r0, %r1) goto %r3
  if <=(%r0, %r1) goto less:
  %r0 := -(%r0, %r1)
  goto mcd:
less:
  %r1 := -(%r1, %r0)
```

eAML

...

The future

epsilon should be implemented with a cleanly layered approach:

- extended language, epsilon, core language, eAM
- Each layer useful and reusable by itself

User-definable syntax, semantics and constraints

Easy to reason about: partial evaluation, analysis for parallel execution, ...

- ...and a Scheme backend (I promised it to RMS)

Thanks

For more information...

http://www-lipn.lipn.univ-paris13.fr/~saiu

http://www.gnu.org/software/epsilon

positron@gnu.org

saiu@lipn.univ-paris13.fr

Thanks.