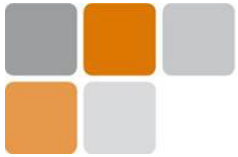


GCC MELT

a high-level domain specific language to extend the GCC compiler

gcc-melt.org

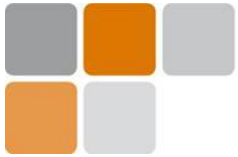
Basile STARYNKEVITCH

basile@starynkevitch.net or basile.starynkevitch@cea.fr

(Laboratoire de Sûreté du Logiciel = Software Reliability Lab)

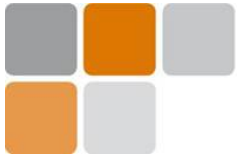
CEA, LIST (DILS), NanoInnov b862 PC174, CEA/Saclay, 91191 Gif/Yvette Cedex, France

TAPAS2012 tool presentation within SAS2012, Deauville, september 14th 2012



#include <stddisclaimer.h>

- **all opinions here are only mine**
- I don't speak for my employer CEA, LIST
  (or for any funding agencies, or any other institution)
- I don't speak for the GCC community
  (I have strong opinions about GCC not shared by it)



CONTENTS

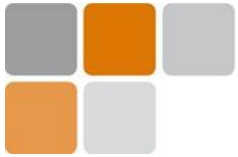
- compilers do use static analysis techniques.
- static analysers do share a lot of luggage with compilers:
  1. parsing and abstract syntax tree representations
  2. internal representations of a compiler (e.g. “control flow graph”, “liveness of variables”, “cross-referencing”)
  3. utilities and framework (e.g. giving warnings to the user)
  4. etc...

  Take profit of a lot of work available in free software compilers
- sophisticated static analysis could profit to weird uses of compilers:
  1. extreme optimizations (e.g. -O∞)
  2. coding rules validations

  Apply your geniune static analysis techniques to compilation issues

Usable free compilers for common low-level languages LLVM or GCC.

- `llvm.org` with Clang
  - BSD licensed, weaker contribution from industry; Apple dominated
  - clean design and code in C++, well documented
  - few source langages (C, C++, Objective C)
  - few targets (x86, ARM, ...)
  - LLVM stricto sensu is a JIT-ing library, Clang is the compiler frontend



I know GCC (but Uday Khedker knows it much better www.cse.iitb.ac.in/grc)

- gcc.gnu.org
  - messy, old, legacy compiler
  - GPLv3 licensed, so strong industry contributions; FSF owned, so no single industrial dominator, but "harsh" community
  - legacy [spagetti?] code, under-documented
  - many source languages (C, C++, Objective C, Go, Ada, Fortran, D)
  - many targets (more than 30, including x86, PowerPC, ARM, and many "weird" processors) and systems (Linux, Windows, FreeBSD, Android, ...)
  - source code in C, now going into C++[1]
  - GCC is a compiler collection with compiler generators

Nobody knows well both GCC and LLVM

[1] Much more dirty than LLVM C++ class hierarchy

MELT `gcc-melt.org` is a [meta-]plugin for GCC providing a high-level domain specific language to extend GCC.

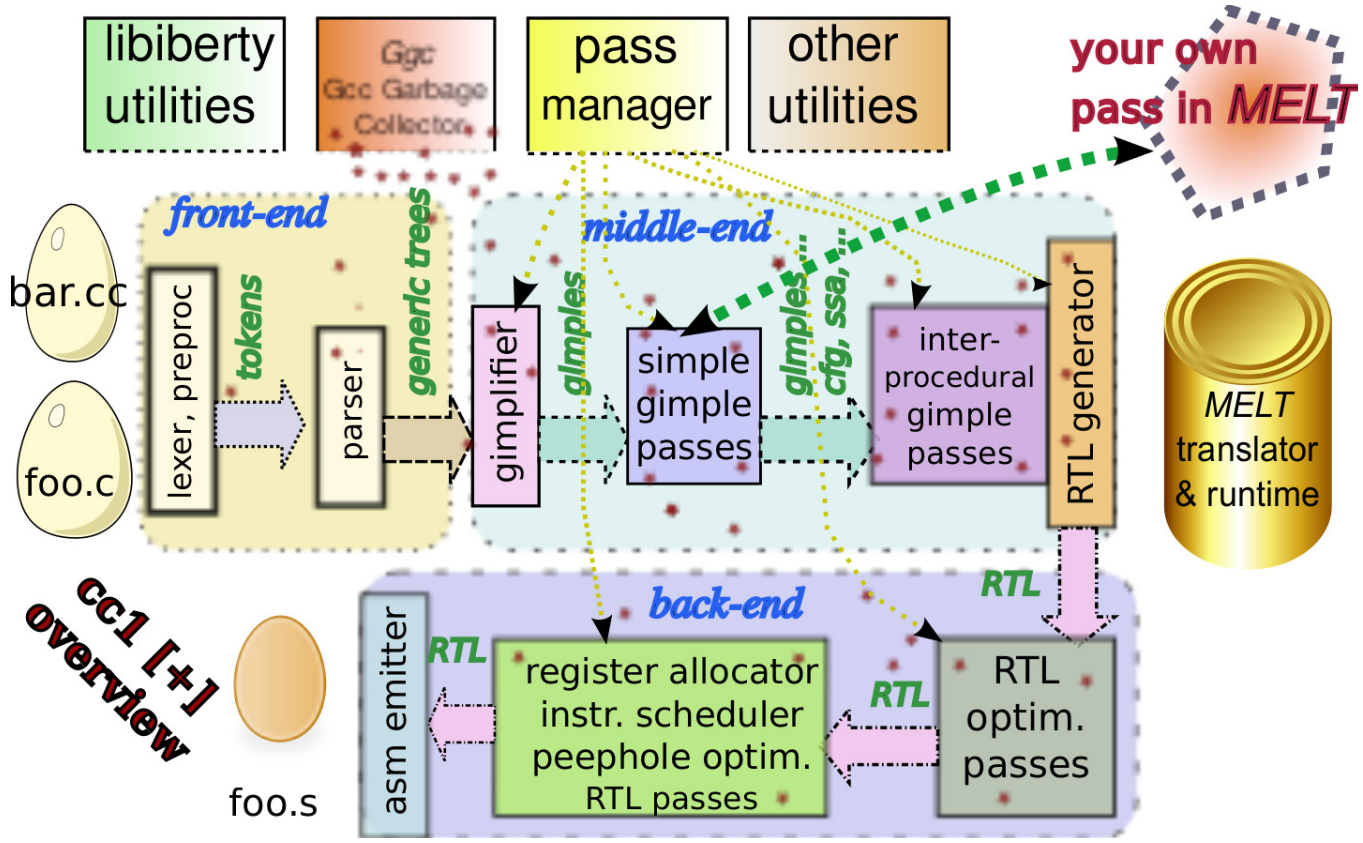
- plugging Ocaml into GCC is not humanly feasible (I tried) GCC has more than 2000 types and $\approx 10 MLOC$ [2]
- MELT is a free (GPLv3 licensed, FSF copyrighted) plugin for GCC 4.6 or 4.7
- MELT is a DSL fitting into GCC internals
- MELT provide some features of Ocaml (or Scheme)
  1. garbage collection of values
  2. pattern matching
  3. high-order programming (closures)
  4. (but not static typing or type inference) unlike Ocaml, MELT is a mostly dynamicly typed language (à la Scheme)

[2] See David Malcom's `gcc-python-plugin`

[I don't know really abstract interpretation]

- no sophisticated analysis done (yet!) in MELT
- but some simple ones
- and a usable infrastructure
- coding in MELT is probably more concise than coding plugins in C for GCC

GCC & MELT picture

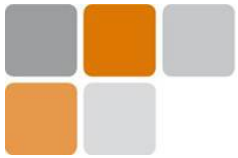


GCC has many rich internal representations
(thousands of C data types, i.e. `struct`)

- *Tree*-*s*[3] for the AST of declarations, source [or SSA] variables, operands
- *Gimple*-*s*[4] for the simple instructions (e.g. 3 operands instructions à la $x \leftarrow y + z$)
- `basicblock`-s made of `gimple`-s (thru `gimpleseq`-s)
- edge-s for the control flow graph, between `basicblock`-s
- etc

The `GTY(())` annotation is for garbage collection in Gcc source code

[3] 200 different variants of `tree`-s, see file `gcc/tree.def` of Gcc
[4] 38 different variants of `gimple`-s, see file `gcc/gimple.def`, half for OpenMP



Looking into some of the GCC internals:

- dumping facilities, e.g. `gcc -fdump-tree-all -O -c foo.c` gives hundreds of files like[5] `foo.c.073t.phiopt1` ...
- with MELT's probe facility:
  `gcc -fplugin=melt -fplugin-arg-melt-mode=probe -O -c foo.c`
  - `-fplugin=melt` loads the MELT plugin[6]
  - `-fplugin-arg-melt-mode=probe` gives the *mode* for the MELT plugin[7]
  - MELT has many other options `-fplugin-arg-melt-debug` shows a lot of debugging output (to debug MELT or your MELT extensions).

---

[5] the number `073t` is absolutely meaningless

[6] You could load several plugins, but you usually load one at most

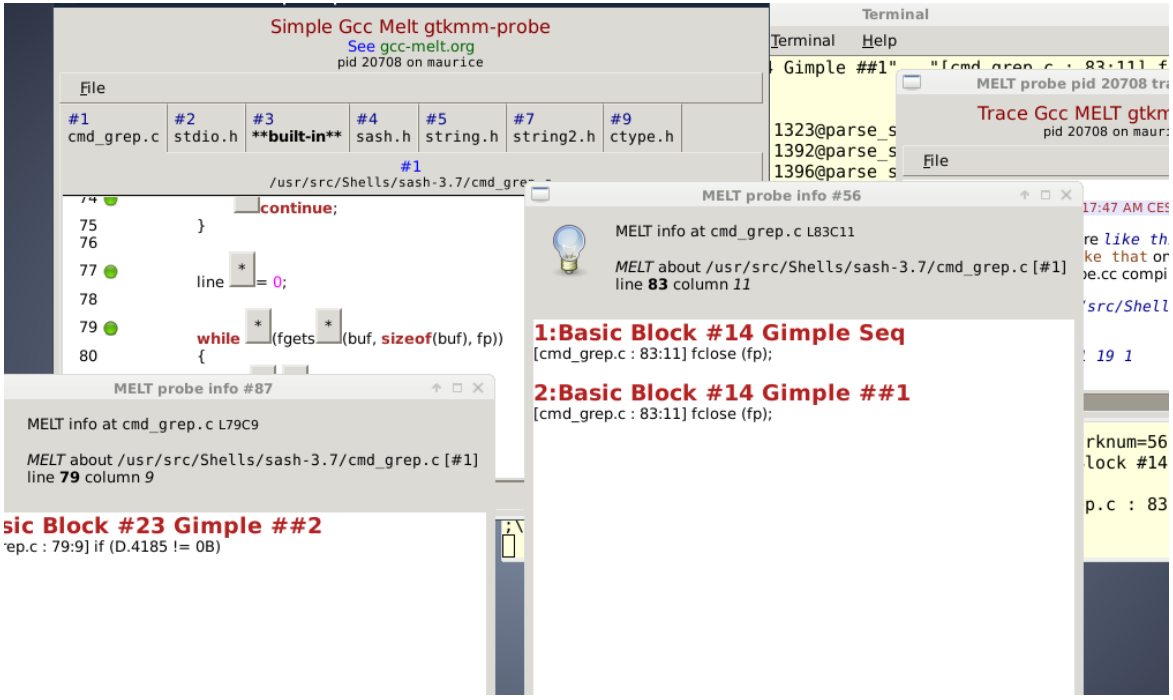
[7] without any mode, MELT does nothing. Use the `help` mode to get help about existing modes.

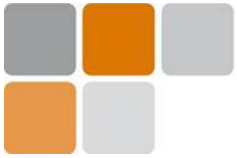
MELT probe demo

with source of `sash-3.7` file `cmd_grep.c` l.70

```
gcc -fplugin=melt -fplugin-arg-melt-mode=probe \
          -O -c cmd_grep.c
```

(a buggy GTK probe GUI interface to MELT with textual protocols to/from GCC+MELT)

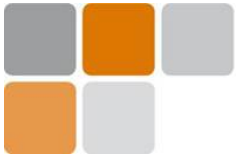




GCC infrastructure

- utilities, e.g. diagnostic messages or options handling
- pass manager (about ≈ 250 passes in GCC)
- [poor man's] GCC garbage collector *Ggc*
  only called *between* passes, don't handle local[8] data!
- extending GCC by adding **your pass**
  - various kind of passes, notably Gimple, IPA (interprocedural analysis), RTL
  - where should you add your pass???

[8] *Ggc* is not managing pointers in the call stack; not managing data internal to a pass; usable for data shared between passes



Lisp-like syntax (*operator operands* ...)

- (**let** ((σ_1 ϵ_1) (σ_2 ϵ_2)) β_1 β_2 β_3) like Ocaml's `let` σ_1 `=` ϵ_1 `in let` σ_2 `=` ϵ_2 `in` β_1 `;` β_2 `;` β_3 or Scheme's `let*`; use **letrec** like Ocaml's `let rec`
- (**progn** ϵ_1 ϵ_2 ϵ_3 ϵ_4) like Ocaml's `begin` ϵ_1`;` ϵ_2`;` ϵ_3`;` ϵ_4 `end`
- (**lambda** (x) β) like Ocaml's `fun x ->` β
- (**defun** `foo (x y)` β_1 β_2) to define a named function like Ocaml's `let foo x y =` β_1 `;` β_2 `;;`
- (**if** τ ϵ ϵ') like Ocaml's `if` τ `then` ϵ `else` ϵ'

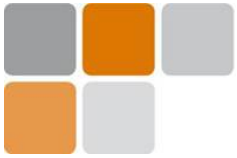
Syntactic sugar: 'ϵ parsed as (**quote** ϵ) for "quotations"; ?ϵ as (**question** ϵ) for patterns; !ϵ as (**exclaim** ϵ) for references. Names (a.k.a symbols) may contain non-letter characters, so `a-b` or `+i` is a single name. Case is not significant.

In MELT (with the patterns π_i usually starting with ?)

```
( match μ
      ( π1 β1,1 β1,2 )
      ( π2 β2,1 β2,2 β2,3 )
      ( π3 β3 ))
```

like in Ocaml

```
begin match μ with
    π1 -> begin β1,1 ; β1,2 end
  | π2 -> begin β2,1 ; β2,2 ; β2,3 end
  | π3 -> β3
end
```



Find every call to `fflush(NULL)` in functions whose name starts with bar with a pass coded in Melt, mostly:

```
(match cfundecl
  ( ?(tree_function_decl_named
      ?(cstring_prefixed "bar") ?_)
     (each_bb_current_fun () (:basic_block bb)
       (eachgimple_in_basicblock (bb)
          (:gimple g)
          (match  g
           ( ?(gimple_call_1  ?_
               ?(tree_function_decl_named
                  ?(cstring_same "fflush") ?_)
               ?(tree_integer_cst 0))
              (inform_at_gimple g
                 "found fflush(NULL)"))
           ( ?_ ())))))
  ( ?_ ()))
```

MELT is translated to C code. That generated C code could be compiled (by a `make` process started by MELT i.e. `gcc -fplugin=melt`) into a module (shared object), then `dlopen`-ed by the same MELT run.

MELT is not a *Gcc* front-end.

The MELT to *C* translator is bootstrapped, i.e. implemented in MELT ($\approx 57 KLOC$). The *C* form of the translator `melt/generated/*.c` is distributed with MELT source code (like `boot/ocamlc` for Ocaml).

Your *C* code can be mixed inside MELT

MELT provides a lot of *linguistic devices* to define MELT constructions in terms of their generated *C* code

Things = MELT values ∪ GCC stuff

1. MELT first-class values (preferable)
   - Nil, closures, lists, boxed strings, boxed `tree`-s, boxed `gimple`-s, MELT objects, etc
   - homogeneous hash-tables or maps: Associate a key to a non-nil value.
     required, because GCC don't permit to extend its data structures (no slot in `tree`-s for client data).
   - fast allocation, because of MELT generational copying collector backed up by *Ggc*
2. GCC stuff (second-class, but useful) - the raw *C* data
   - `gimple`-s, `tree`-s, `edge`-s, `long` etc etc
   - only collected by *Ggc*

MELT is dynamically typed for values, and statically typed for stuff
c-type annotations in MELT code like `:tree`

Defining matchers by macro-strings

```
(defcmatcher gimple_assign_minus
  (:gimple ga)
  (:tree lhs rhs1 rhs2)
  gasminus
  ;; test
  #{/*gimple_assign_minus $GASMINUS ?*/ ($ga && is_gimple_assign($ga)
     && gimple_expr_code($ga) == MINUS_EXPR)}#
  ;; fill
  #{/*gimple_assign_minus $GASMINUS !*/
   $lhs = gimple_assign_lhs($ga);
   $rhs1 = gimple_assign_rhs1($ga);
   $rhs2 = gimple_assign_rhs2($ga); }#
  ;; operator expansion
  #{/*gimple_assign_minus:*/ gimple_build_assign_with_ops(MINUS_EXPR,
    $LHS, $RHS1, $RHS2)}#
)
```

Small examples of MELT code

e.g. `melt-examples/ex06`

Conclusion

- GCC is legacy code: 10MLOC and still growing
- MELT enables to write "quickly" some passes working on (or modifying) GCC internals (notably Gimple)
- ad-hoc pattern matching (with views à la Wadler) is essential

Coming soon in MELT (usually release every 2 months): evaluator of MELT expressions, more Gimple
Future work: LTO!

I'm interested in joining e.g. European or French collaborative research projects to use MELT to enable your sophisticated analyzers in GCC

`basile.starynkevitch@cea.fr`

More on `gcc-melt.org`