Customizing your GCC compiler with MELT extensions

Basile STARYNKEVITCH
www.gcc-melt.org
basile.starynkevitch@cea.fr or basile@starynkevitch.net
CEA LIST (DILS, LSL), CEA NanoInnov b.862 PC174, 91191 GIF/YVETTE CEDEX, France

January, 2016

1 Customize your GCC compiler ?

The GNU COMPILER COLLECTION (see gcc.gnu.org for more) is a widely used free software (GPLv3+ licensed) compiler suite. Its current release is **5.3** (december 2015). It accepts many source languages (*C*, *C++*, *Fortran*, *Go*, *Ada*, *Objective-C* etc...) in their latest standard (e.g. *C++2014*). It targets many processors (including *x86 [-64]*, *ARM*, *Sparc*, *PowerPC*, etc...) for various systems (e.g. *Linux*, *MacOSX*, *Android*, *Windows*, etc...). It can be used as a (straight- or a) **cross-compiler**. So it is a mature and complex software (more than 10 MLOC of size) with a large (≈ 400 full-time developers) community of developers. Hence, it is very competitive (see `openbenchmarking.org` for benchmarks).

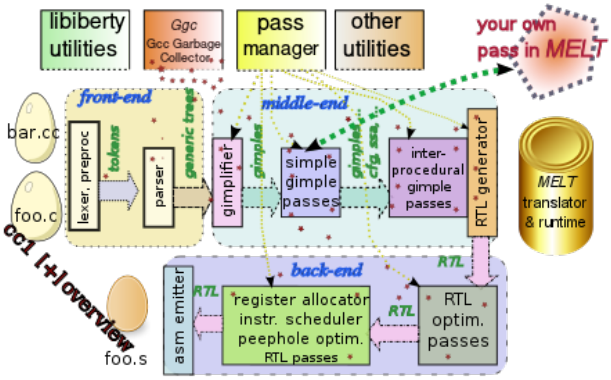

Figure 1: Overview of `cc1` - the compiler proper

The `gcc-5` (or `gcc-4.9`) program is driving several utilities, including `cc1` (the compiler from *C* to assembly, see figure 1), `as` (the assembler), `ld` (the linker), `lto1` (the link-time optimizer) ...

Since its 4.5 release, GCC is *extensible* thru **plugins**, which can :

- add (or remove, or reorganize) their extra passes, working on internal GCC representations (notably *Gimple* a mostly 3-operands instruction set, and *Tree*-s)
- add new *builtins*, *pragmas*, *attributes* thus slightly extending the accepted source language.

This enables you (or your consultant) to heavily **customize** your GCC compiler to suite particular needs, for:

- specific coding rules
- particular optimizations
- aspect oriented programming
- static analysis (but use `www.frama-c.com` for coding and analysis of critical real-time embedded software)
- customized warnings
- code refactoring and navigation help
- any custom GCC extension (working on internal middle-end representations, e.g. *Gimple*, *Tree*, ...) and taking advantage of the numerous internal representations and processing of the compiler

2 MELT domain specific language

MELT is a **high-level domain specific language** to easily **extend** the *Gcc* compiler to suite your specific needs. It enables you to customize your GCC compiler (much more easily than by coding in *C* a plugin for GCC) thru its high-level features:

- simple, systematic, and regular LISP-like syntax
  (*operator* *operand* ...)

- handling of high-level dynamically typed MELT **values** and of low-level GCC specific, statically typed, data **stuff**;
- [dynamic] translation of your MELT code into *C++* code (suited for your GCC);
- powerful runtime support, with a state-of-the-art generational copying **garbage collector** (above existing GCC memory management);
- *very powerful* **pattern-matching** facilities, with extensive coverage of GCC internal data;
- *strong* **meta-programming** facilities thru a Turing-complete Lisp-like macro system;
- ability to **mix small code chunks in *C++*** inside MELT code, and to define MELT constructs by their generated *C++* code, thus using any external C or C++ libraries inside your MELT extension for GCC;
- **high-level programming paradigms**: object-oriented, pattern-matching, functional (and higher-order) programming styles are possible in MELT;
- **interface to** all **GCC plugin hooks**, so you can code your own GCC passes (inspecting or modifying GCC internal representations), add your own *builtins* or *pragmas* in MELT, etc...

The MELT plugin (free software, GPLv3+ license, for GNU/LINUX) also provides a **read-eval-print loop** and runtime evaluation of MELT code.

The figure 2 illustrates the power and simplicity of MELT (assuming the overview of GCC internal representations is understood). It shows a code to search, in your *C* or *C++* code, inside any function whose name starts with `bar` all the calls to `fflush` with a `NULL` argument (which could have appeared *after* inlining, so a textual approach won't find it!).

Future or on-going (pre-α stage) **work** includes a MELT monitor providing a persistent framework (keeping static analysis information) and a web interface (to show them), connected thru sockets to MELT enabled compilation

```
(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)"))
           ( ?_ ()))))))
  ( ?_ ()))
```

Figure 2: excerpt of MELT code

- tutorial talks about GCC extensions with MELT
- **industrial contract** (thru CEA LIST) for any MELT related development or project
- **collaborative research projects**
  In particular H2020 ICT10 call `http://ec.europa.eu/research/participants/portal/desktop/en/opportunities/h2020/topics/5098-ict-10-2016.html` has a focus on *Algorithms and techniques for extracting knowledge (e.g., specifications, designs or models) from the huge amount of existing open source code; tools using that knowledge in the development of new software*, and I am part of a consortium submitting a proposal to it. (and perhaps some FET OPEN proposal, mixing static analysis and natural language processing techniques)
- my **expertise on compilation** and GCC internals

MELT is a free software, GPLv3+ licensed, FSF copyrighted, available as a plugin for recent GCC versions on `gcc-melt.org`

3 Contact for more

Please contact me `basile.starynkevitch@cea.fr`
(office phone +33 1 6908 6595, mobile +33 6 8501 2359) for

This sheet is downloadable on **`gcc-melt.org/gcc-melt-sheet.pdf`**