The Dolorem Pattern: Growing a Language Through Compile-Time Function Execution (Artifact)

Simon Henniger
Technische Universität München, Germany

Nada Amin
Harvard University, Cambridge, MA, USA

Abstract

Programming languages are often designed as static, monolithic units. As a result, they are inflexible. We show a new mechanism of programming language design that allows to more flexible languages: by using compile-time function execution and metaprogramming, we implement a language mostly in itself. Our approach is usable for creating feature-rich, yet low-overhead system programming languages. We illustrate it on two systems, one that lowers to C and one that lowers to LLVM.

2012 ACM Subject Classification Software and its engineering → Compilers; Software and its engineering → Language features

Keywords and phrases extensible languages, meta programming, macros, program generation, compilation

Digital Object Identifier 10.4230/DARTS.9.2.13

Related Article Simon Henniger and Nada Amin, "The Dolorem Pattern: Growing a Language Through Compile-Time Function Execution", in 37th European Conference on Object-Oriented Programming (ECOOP 2023), LIPIcs, Vol. 263, pp. 41:1–41:27, 2023.
`https://doi.org/10.4230/LIPIcs.ECOOP.2023.41`

Related Conference 37th European Conference on Object-Oriented Programming (ECOOP 2023), July 17–21, 2023, Seattle, Washington, United States

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2023 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

The artifact is a VM with Fedora Linux and the following software components pre-installed on the desktop:

- Code of dolorem-c and dolorem-llvm (both can be found on the desktop or in `/home/artifact/Desktop`)
- Binaries of dolorem-c (with gcc), dolorem-llvm (with tcc), and dolorem-llvm
- Examples and benchmarks

This means the artifact comes with a caveat: Since it is a virtual machine, any measurements within it will always be more noisy than measurements on real hardware (like the measurements in the paper).

The following from the paper can be seen in the artifact:

- Running code in dolorem-c – no specific claim, but we feel the artifact is useful to test the language
- Running code in dolorem-llvm (again, no specific claim)
- Running dolorem-llvm Pong
- dolorem-llvm Compile speed measurements (figure 3 on p. 25)
- Time spent in the C compiler for dolorem-c (claims from section 5.5)

See below for precise instructions.

2 Content

The artifact package includes:

- the VM image.

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on Zenodo. The artifact is available at: `https://zenodo.org/record/7720029`

4 Tested platforms

The VM was tested in VirtualBox 7.0, but should work with other hypervisors as well.

5 License

Anything newly written for the artifact is availble under CC-BY-AT.

6 MD5 sum of the artifact

9b5d480f8227dfbfcaf693506a960fec

7 Size of the artifact

3.3 GB

A Getting Started

Start the VM. It contains a Fedora Linux install with only one user, “artifact”. The user does not have a password. When prompted for a passwort, simply press Enter. Note that, by default, the VM uses a US layout. You can change this by right-clicking on the American flag in the bottom-right corner and pressing “Keyboard Handler Settings”, adding your keyboard layout to the list and then clicking the flag in the panel to switch layouts.

You will find three desktop folders with source code and binaries, one each for dolorem-c (with gcc), dolorem-c with tcc, and dolorem-llvm.

Right-click the implemenation you want and use “Open in Terminal”. Important: For every implementation, you first need to type “source confld.sh” in your terminal to setup a local variable. Otherwise, you will see a dl error.

Here’s how to verify our claims:

A.1 Running code in dolorem-c

The artifact can be used to try and develop code for dolorem-c (both with clang and with tcc). Try `./dolorem rlpl.dlr` for a loop that generates C code for dolorem-c code you type in. Note that the VM is quite slow for dolorem-c with clang (see caveat above). A few suggestions:

- Start with something simple. Type in a number or a string literal.
- Then maybe use an operator. Try `(add 1 1)`.
- Try calling a function, like `(puts "hello, world")`.

- Now, let's make a function. Try `(function hello-world () void (puts "hello, world"))`.
- Write `(compile (function hello-world () void (puts "hello, world")))`. (You may know this is equivalent to `(defun hello-world () void (puts "hello, world"))` because of how "defun" is defined.)
- You should have seen that (obviously) compiling yields no new C code. But after compilation, our function is available to use! So let's write a macro that uses it: `(defmacro hello-world-macro (progn (hello-world) (make-cexp "" "" "" "")))`
- Finally, call the macro: `(hello-world-macro)`

A.2 Running code in dolorem-llvm

You can do something similar for dolorem-llvm. The file "a.dlr" contains some example code that shows off a few language features. We recommended you open a text editor and change this file. Then run it with `./dolorem a.dlr`.

A.3 Pong

We reference a Pong implementation that runs based on SDL2 and dolorem-llvm (with two small C adaptor functions we link in). Run it by typing `./dolorem pong.dlr` in the dolorem-llvm folder and play Pong. Use the arrow keys to control the game, and Esc to quit.

A.4 Compile speed measurements

Type `./benchmark_compile.sh` to get the numbers for figure 4.

If your performance measurements are all zeros, remember to type `source confld.sh`.

A.5 Time spent in the C compiler for dolorem-c

In section 5.5, we make various claims on how much time is spent in the C compiler. To check the claims, run `./dolorem -M def.dlr` for both dolorem-c with gcc, and dolorem-c with tcc and compare the results. You can also try this on other dolorem-c code.