Dragon kit - language oriented compiler framework (shorter abstract)

Wojciech Meyer

2012-07-16 Mon

Contents

1 Overview

In this demo I will present Dragon framework (or kit) an extensible language oriented compiler framework. OCaml is an excellent choice to write a compiler, but it's lack of native infrastructure makes it difficult it to be considered as a generic solution for a compiler development. LLVM is a very successful project written in C++ and it serves the industry as Swiss army knife in the development of compilers, DSLs and static analysis tools. Although there exist bindings for OCaml for basic functionality in LLVM it remains a blackbox and more over it's written in C++ - different paradigm makes it more difficult to use natively in OCaml.

1.1 Current approach

Current approach to compiler frameworks is to provide some bits and pieces to be able to compile majority of the industrial languages (mostly imperative) to the machine code.

What are their features and what makes Dragon kit significantly different? Let's see that what other can offer us:

- QuickC– - written in Literate (! `http://www.literateprogramming.com/`) OCaml, implementation of C– language, stable, unfortunately not very active
- MLRISC - as for now not active, but perhaps active as a monolithic part of NJ SML and Mlton. Nice idea, well developed and code entirely in SML!
- LLVM - written in C++, very popular, very active, single open IR - not extensible. Excellent community behind it with a lot of research around.
- GCC - single close intermediate language GIMPLE - not extensible and can be used only with plugins (or through MELT). Robust, many backends - long history.

1.2 What are the problems

We can observe that each of this is almost ruled by a single centralised intermediate representation. This makes it difficult from point of view:

- writing optimisation passes - instead of transforming one language to different simpler one - we are forced to the transformation in one go. Usual practice is LLVM is to collect and cache information during the pass, and then do the destructive update on IR (adding instructions, removing basicblocks, reordering them etc.)
- specification bloat - due to number of different languages and backends that the compiler framework needs to support - we can't use single representation as it does not always fit the purpose - quickly leaky abstractions are forcing us to extend the IR - and handle even more special cases in the code that suppose to be generic. More over documenting and understanding one big intermediate representation is much more difficult than grasping around few single dedicated IR's.
- safety - lowering to simpler language is usually a good way of assuming that some of the interesting properties are preserved by encapsulating them in the simpler language. If we add type checking to each of these languages, we can be more less sure, we preserved the basic invariants in the language we are targeting. These means that we can throw our all the lousy assumptions that exist in one big IR after some of the passes, and constraint them by a new more limited language.

If there was a way of implementing quickly language that is dedicated to some particular task (e.g. transform only, a compiler frontend AST, helper DSL to abstract something and generate some code or data etc.) then we will end up with a really novel approach to the compilation - but not only limited to compilation.

1.3 Dragon kit

This is the essence of the Dragon framework. Dragon framework will consist of stacked languages to perform dedicated tasks. The key idea is that they can be composed like modules with functors, and each of them share the same interface. So looking at OCaml code base we could formalise the pipeline in the compiler consisting of the several intermediate languages and transforms to some more generic form, and register them as dragons in the Dragon framework, then pass them to functors to to compile them to the backend language and possibly implement some optimisation phases.

Please visit `https://github.com/danmey/DragonKit` for more information. Worth to note it doesn't contain yet all the modules, in particular the backend part is pre-generated from the "Arch" DSL which is not yet ready for github.

1.4 Status for today

So far the Dragon framework is under heavy development, the idea is there, the code not fully yet there, although we are able to bootstrap as of today a simple JIT compiler for X86 for Pascal like language already. Now the majority of work is being done on a backend languages and infrastructure for supporting it, especially some work recently was dedicated to the typesafe DSL "arch" language specifying the backend and the target architecture. Some work has been also done on "llvm" Dragon that provides bi-directional gate to LLVM - both compiling down Dragon IR to LLVM and use the existing industrial strength backend, and reading LLVM IR from any frontend and use our Dragon framework backend to compile it to machine).

Dragon framework is being developed in breadth first mode by one man, I believe that it's very important in such framework to build up needed infrastructure properly first and before the tool will start growing. I hope it will be in interests of researchers and the community to see that we can provide an excellent "compiler framework" natively in OCaml!