TDDB44/TDDD55 Lecture 14: Bootstrapping of a Compiler

Optional Material

Martin Sjölund

Department of Computer and Information Science
Linköping University

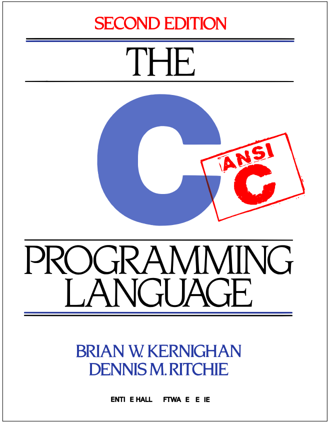
2018-12-17

How to Implement a Compiler

- Implement your compiler in an existing language (easy).
- Writing your compiler in the language it is trying to compile (bootstrapping):
  - ☺ Another compiler already exists, with binaries for your build architecture.
  - 😐 Another compiler already exists, but no binaries for your build architecture (only 32-bit; your system is 64-bit; cross-compiling + bootstrapping).
  - ☹ No other compiler exists.

Example: Origins of C

- Started as the language B, a simple dialect of BCPL.
- The B compiler was implemented in TMG, a language for writing compiler, itself written in PDP-7 assembler.
- The B compiler was then rewritten in B itself, and compiled using the TMG version of the B compiler.
- The B compiler was then tweaked into “New B”, and eventually became the C language and compiler.



Bootstrapping Language x: Alternatives

Notation: ${}^{k}C_{x}^{o}$, a compiler C written in the language x which compiles the source language k into the object language o.

- Implement a small, stupid compiler for x_{subset} in another language y, producing native executables. Bootstrap using this compiler ($A_1 = {}^{y}C_{x_{subset}}^{native,unoptimized}$).
- Bootstrap using a different compiler that implements x or x_{subset} ($A_2 = {}^{?}C_{x_{subset}}^{native}$).
- Keep a tarball of translated C-code that produces an x_{subset} compiler. Compile this old, basic version of the compiler ($A_3 = {}^{C}C_{x}^{native,unoptimized}$, generated by ${}^{x}C_{x}^{C}$).
- Write an interpreter for x_{subset}. Feed it your compiler as input, ... (A_4)
- Or keep a tarball of bytecode for x that you can interpret. ($A_5 = {}^{bytecode}C_{x_{subset}}^{native,unoptimized}$)
- Interpret x code with a human in the loop, being fed your compiler as input. (A_6)

Bootstrapping Language x: Step 2

- Compile a (subset) version of your compiler ($B = {}^{x}C_{x_{subset}}^{native,unoptimized}$) using this other compiler (A_n). This version might be incomplete (optimization modules disabled, etc, that A_n does not support).
- Compile a full version of your compiler ($C = {}^{x}C_{x}^{native}$), using ($B = {}^{x}C_{x_{subset}}^{native,unoptimized}$).
- Compile an optimized, full version of your compiler ($D = {}^{x}C_{x}^{native}$) using ($C = {}^{x}C_{x}^{native}$), targeting (possibly cross-compiling) your host platform.

Rationale

- It is a proof that your language is powerful enough to do something useful.
- Why should I use your programming language if you yourself use C?
- Only need to learn one language to be a compiler developer.
- Improving the performance for the language also improves the performance of the compiler.

OpenModelica Bootstrapping History (1)

- Implementation of a Modelica compiler using rml2c

OpenModelica Bootstrapping History (1)

- Implementation of a Modelica compiler using rml2c
- Design of an early MetaModelica language version as an extended subset of Modelica, spring 2005.

OpenModelica Bootstrapping History (1)

- Implementation of a Modelica compiler using rml2c
- Design of an early MetaModelica language version as an extended subset of Modelica, spring 2005.
- Implementation of a MetaModelica Compiler (MMC) which translates MetaModelica into RML intermediate form, spring-fall 2005.

OpenModelica Bootstrapping History (1)

- Implementation of a Modelica compiler using rml2c
- Design of an early MetaModelica language version as an extended subset of Modelica, spring 2005.
- Implementation of a MetaModelica Compiler (MMC) which translates MetaModelica into RML intermediate form, spring-fall 2005.
- Automatically translating the whole OpenModelica compiler, 60 000 lines, from RML to MetaModelica.

OpenModelica Bootstrapping History (1)

- Implementation of a Modelica compiler using rml2c
- Design of an early MetaModelica language version as an extended subset of Modelica, spring 2005.
- Implementation of a MetaModelica Compiler (MMC) which translates MetaModelica into RML intermediate form, spring-fall 2005.
- Automatically translating the whole OpenModelica compiler, 60 000 lines, from RML to MetaModelica.
- In parallel, developing MDT (Modelica Development Tooling), including debugger for MMC, 2005-2006.

OpenModelica Bootstrapping History (1)

- Implementation of a Modelica compiler using rml2c
- Design of an early MetaModelica language version as an extended subset of Modelica, spring 2005.
- Implementation of a MetaModelica Compiler (MMC) which translates MetaModelica into RML intermediate form, spring-fall 2005.
- Automatically translating the whole OpenModelica compiler, 60 000 lines, from RML to MetaModelica.
- In parallel, developing MDT (Modelica Development Tooling), including debugger for MMC, 2005-2006.
- Switching to using this MetaModelica 1.0, the MMC compiler, and MDT for the OpenModelica compiler development, at that time 3-4 full-time developers. Fall 2006.

OpenModelica Bootstrapping History (1)

- Implementation of a Modelica compiler using rml2c
- Design of an early MetaModelica language version as an extended subset of Modelica, spring 2005.
- Implementation of a MetaModelica Compiler (MMC) which translates MetaModelica into RML intermediate form, spring-fall 2005.
- Automatically translating the whole OpenModelica compiler, 60 000 lines, from RML to MetaModelica.
- In parallel, developing MDT (Modelica Development Tooling), including debugger for MMC, 2005-2006.
- Switching to using this MetaModelica 1.0, the MMC compiler, and MDT for the OpenModelica compiler development, at that time 3-4 full-time developers. Fall 2006.
- Preliminary implementation of pattern-matching and exception handling in the OpenModelica compiler, to enable future bootstrapping. Spring-fall 2008.

OpenModelica Bootstrapping History (2)

- Continuation of the work on better support for pattern-matching compilation, support for lists, tuples, records, etc. in OpenModelica. Spring-fall 2009.

OpenModelica Bootstrapping History (2)

- Continuation of the work on better support for pattern-matching compilation, support for lists, tuples, records, etc. in OpenModelica. Spring-fall 2009.
- Implementation of higher-order functions (used in MetaModelica), also in OpenModelica. Fall 2009, spring 2010.

OpenModelica Bootstrapping History (2)

- Continuation of the work on better support for pattern-matching compilation, support for lists, tuples, records, etc. in OpenModelica. Spring-fall 2009.
- Implementation of higher-order functions (used in MetaModelica), also in OpenModelica. Fall 2009, spring 2010.
- The bootstrapped compiler supporting most of MetaModelica 2.0, which includes standard Modelica. Fall 2010, spring 2011.

OpenModelica Bootstrapping History (2)

- Continuation of the work on better support for pattern-matching compilation, support for lists, tuples, records, etc. in OpenModelica. Spring-fall 2009.
- Implementation of higher-order functions (used in MetaModelica), also in OpenModelica. Fall 2009, spring 2010.
- The bootstrapped compiler supporting most of MetaModelica 2.0, which includes standard Modelica. Fall 2010, spring 2011.
- Adding garbage collection. Fall 2012.

OpenModelica Bootstrapping History (2)

- Continuation of the work on better support for pattern-matching compilation, support for lists, tuples, records, etc. in OpenModelica. Spring-fall 2009.
- Implementation of higher-order functions (used in MetaModelica), also in OpenModelica. Fall 2009, spring 2010.
- The bootstrapped compiler supporting most of MetaModelica 2.0, which includes standard Modelica. Fall 2010, spring 2011.
- Adding garbage collection. Fall 2012.
- Improving the build system, parallel builds. Reaching full testsuite coverage, good performance, and running the tests nightly. 2013.

OpenModelica Bootstrapping History (2)

- Continuation of the work on better support for pattern-matching compilation, support for lists, tuples, records, etc. in OpenModelica. Spring-fall 2009.
- Implementation of higher-order functions (used in MetaModelica), also in OpenModelica. Fall 2009, spring 2010.
- The bootstrapped compiler supporting most of MetaModelica 2.0, which includes standard Modelica. Fall 2010, spring 2011.
- Adding garbage collection. Fall 2012.
- Improving the build system, parallel builds. Reaching full testsuite coverage, good performance, and running the tests nightly. 2013.
- Removing support for MMC.

OpenModelica Bootstrapping History (2)

- Continuation of the work on better support for pattern-matching compilation, support for lists, tuples, records, etc. in OpenModelica. Spring-fall 2009.
- Implementation of higher-order functions (used in MetaModelica), also in OpenModelica. Fall 2009, spring 2010.
- The bootstrapped compiler supporting most of MetaModelica 2.0, which includes standard Modelica. Fall 2010, spring 2011.
- Adding garbage collection. Fall 2012.
- Improving the build system, parallel builds. Reaching full testsuite coverage, good performance, and running the tests nightly. 2013.
- Removing support for MMC.
- Further adding, enhancing, and redesigning MetaModelica language features, based on usage experience, the Modelica design effort, and inspiration from functional languages and languages. Refactoring parts of the compiler to use the enhanced features.

OpenModelica Bootstrapping

- Start with a tarball of source-code (only code necessary for bootstrapping).
- This source-code was at one time generated by OMC compiled with RML/MMC.
- At some point, OMC was able to generate its own tarball.
- Then support for RML/MMC was dropped and new language features added to OMC (that RML/MMC did not support).
- At a later time, these new language features were used in the compiler itself (and a new tarball was generated).
- Parts of the compiler that are not used during bootstrapping can use new language features before a new tarball is generated.
- ...

OpenModelica Cross-Compiling (ARM host, x86 build)

- Start with a tarball of source-code.
- Bootstrap the x86 version of OpenModelica, save this somewhere. Make clean.
- ./configure –with-omc=path/to/x86/omc
- Cross-compile the ARM version of OpenModelica using the x86 version of OMC to produce code.
- Note: OMC generates C-code, so you need a cross-compiler tool-chain installed.
- For gcc, a similar approach is used, but you then use the regular gcc to compile a version of gcc that runs on x86 but produces ARM executables (including assemblers and linkers).
- clang (LLVM) is able to produce assembly for multiple targets using the same compiler (but it does not integrate assemblers, linkers, or C++ run-times for these targets, so you usually need to install a gcc cross-compilation tool-chain anyway).

www.liu.se