

João Dias

dias@eecs.harvard.edu
<http://www.eecs.harvard.edu/~dias/>

Interests Programming languages, compilers, and systems

Education Harvard University, Cambridge, MA.
PhD in Computer Science, expected Fall 2008.
Supervisor: Norman Ramsey

Harvard University, Cambridge, MA.
AB in Computer Science, 2002.

Publications *Converting Intermediate Code to Assembly Code Using Declarative Machine Descriptions.*
João Dias and Norman Ramsey. 15th International Conference on Compiler Construction (CC'06), March, 2006.

An Applicative Control-Flow Graph Based on Huet's Zipper.
Norman Ramsey and João Dias. ML Workshop, September, 2005.

Generating a Recognizer from Declarative Machine Descriptions.
João Dias and Norman Ramsey. Harvard Technical Report TR-30-04, 2004.

Research QUICK C-- COMPILER PROJECT

<http://www.cminusminus.org>

Harvard University

The goal of the Quick C-- compiler project is to develop a reusable code generator and optimizer. A compiler for a high-level language may target the C-- language and use the Quick C-- compiler to produce machine code that is almost as good as the machine code produced by a hand-written code generator. Through a run-time interface, C-- also supports run-time services such as garbage collection and exception handling.

My research focuses on the automatic generation of compiler back ends from declarative machine descriptions. The state-of-the-art in retargeting a compiler is to write code that maps the compiler's intermediate representation onto machine instructions. Our goal is to simplify the task of retargeting a compiler by decoupling the knowledge of the machine from the knowledge of compiler internals. We isolate the knowledge of the target machine in a reusable, declarative machine description. The declarative machine description permits analysis, allowing us to automatically generate the machine-specific components of the compiler. The most important components are the code expander and the recognizer. My dissertation demonstrates that we can automatically generate the recognizer and the expander from a declarative machine description. For details on generating the recognizer, see the 2006 CC paper or the 2004 technical report.

As part of my work on the Quick C-- compiler, I developed various parts of the back end:

- The *optimization backplane* is a framework for dynamically reconfiguring the order of compiler phases; the correctness of the configuration is checked before execution. I defined operational semantics for the optimization backplane and wrote the execution engine.
- The *linear-scan register allocator* is a new algorithm for register allocation that is linear in the number of edges in the control-flow graph. I designed and implemented the algorithm, in addition to an Appel-George graph-coloring register allocator.

- The *run-time system* supports stack walking and access to variables and other program data. I wrote the code that emits run-time data exposing the decisions made by the compiler: the layout of stack frames and the locations of program data. I also helped write the run-time system code for stack walking and variable access.
- The *control-flow graph* is the central data structure in the compiler's back end. I helped design and evaluate the control-flow graph abstraction. For more details, see the ML Workshop paper above.

Teaching Experience

2000-2004

HARVARD UNIVERSITY

Cambridge, MA

Teaching fellow for undergraduate courses in hardware, compilers, and introduction to computer science. Taught sections and graded assignments.

Head teaching fellow for introductory computer science course in Spring 2002. Oversaw a 10-person staff of teaching fellows, in addition to standard teaching-fellow duties.

Industrial Experience

Summer 2008

INTERN, MICROSOFT RESEARCH

Cambridge, England

Worked on redesign of the code generator in the Glasgow Haskell Compiler. The redesign is intended to simplify GHC's code generator and improve the quality of the generated code.

Summer 2005

INTERN, SUN MICROSYSTEMS

Burlington, MA

Worked on development of distributions in the Fortress programming language. Distributions specify spacial layout of data structures, which is used to improve data locality in a randomized work-stealing algorithm by guiding related data and computations to the same processing unit.

Summer 2000

INTERN, HEWLETT PACKARD

Palo Alto, CA

Worked in a marketing group in the IT department, maintaining and improving web access to a central marketing database.

Summer 1999

INTERN, HEWLETT PACKARD

Palo Alto, CA

Worked with an HP Labs group optimizing an MP3 decoder for the ARM processor. Modified MP3 decoding algorithm to use an efficient FFT instead of the standard transformation.