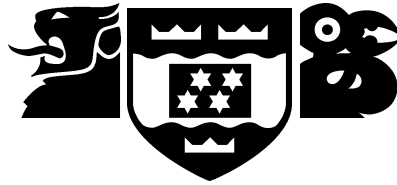


VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



School of Mathematics, Statistics and Computer
Science
Te Kura Tatau

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@mcs.vuw.ac.nz

Project proposal

Andrew Walbran

Supervisor: Associate Professor Lindsay Groves

March 15, 2008

Submitted in partial fulfilment of the requirements for
Bachelor of Science with Honours in Computer Science.

Abstract

This document describes a proposed research project to better optimise Java programs by annotating certain methods as being 'pure' functions with no side-effects. The optimisations will be implemented in the JKit compiler.

Chapter 1

Research project proposal

1.1 Objectives

The aim of this project is to implement Java compiler optimisations based on annotating appropriate methods as being ‘pure’ — i.e. having no side-effects. This will involve:

- Investigating possible optimisations based on annotating functions as not having side-effects.
- Implementing some or all of these optimisations with the JKit compiler being developed here at VUW.
- If time allows, implementing a JKit compiler stage to check that these annotations are correct, and to automatically derive them for unmodified Java programs.

1.2 A title

This project shall be known as

“Optimising Java programs with pure functions”

1.3 Why: background

Java compilers are limited in how they can optimise the use of methods in Java programs by the fact that methods can have side effects. For example a method that returns a value may also change the value of one of the object’s attributes, or even some other state in an apparently unrelated part of the program.

In practice, however, many methods are *pure* functions which only return a value and do not alter any state. If programmers were able to annotate these methods as such, the compiler would be able to make optimisations which would not otherwise be possible. Such annotations would also be helpful in making the programmer’s intent and assumptions clear, resulting in more readable code. If the compiler were able to check that the annotations were valid (i.e. that methods annotated as being pure do not in fact have any side-effects) then certain errors could be caught at compile-time, making debugging easier.

In the following example class, the `mult` method is pure but not the `set` method (as it alters the state of the object). `multk` would also not be considered pure for my purposes, as it reads a static field which could change and so may give a different result even if called with the same arguments and without the object having changed.

```

public class Example {
    private int x;
    public static int k;

    public int mult(int y) {
        return x * y;
    }

    public set(int a) {
        x = a;
    }

    public int multk(int y) {
        return x * y * k;
    }
}

```

Similar features already exist in other programming languages. For example, C++ allows member functions to be marked as `const` if they will not modify the object they are called on, and so can be called on a constant instance of their class [1].

GCC allows C functions to be marked `__attribute__((pure))` if they have no side-effects and depend only on their arguments and global variables, or `__attribute__((const))` if they have no side-effects and depend only on their arguments [2] [3]. It uses these attributes to allow common subexpression elimination and loop optimization, similar to what I plan to do in this project.

Fortran 95 (from the earlier High Performance Fortran) allows pure functions to be marked with the `PURE` keyword, to allow optimisation of `FORALL` loops on parallel systems [4].

In pure functional languages such as Haskell no functions have side-effects, and so the compiler can always make optimisations that assume functions to be pure.

There has been some work done on pure functions in Java, but for use in specifications rather than optimisation. These studies have considered different levels of purity, and it will be important for this project to determine exactly what level is required for the compiler to be able to safely make optimisations. Darvas and Müller [5] describe a notion of *weak purity* that allows methods to construct new objects and even return them as long as they do not modify pre-existing objects. This is too weak for my purposes, as it means that a method could return references to different objects when called with the same arguments, and this can result in quite different behaviour later even though both objects will contain the same values. Naumann [6] gives a definition of *observational purity* which may be more appropriate, though possibly complicated to verify.

1.4 How and when: method and plans

I plan to:

1. Investigate how other compilers optimise pure / const functions, and what research has already been done. Produce a bibliography. (2 weeks)
2. Extend JKit parser to support necessary annotations (1 week)
3. Implement optimisation stages for JKit using these annotations (10 weeks)

4. Run some tests with small programs constructed for the purpose and appropriately annotated
5. Implement checking of const / pure annotations (4 weeks, this could be omitted if time is tight)
6. Write report (3 weeks)

If time allows, I could also:

- Implement functionality to derive pure and const annotations for existing code
- Run tests on a variety of programs (either existing large Java programs or created for this project) using automatically derived annotations to determine how often the optimisations implemented in this project can be applied and how much they affect speed.

By milestone 1 (2008-5-11) I plan to have completed the first two objectives and be well into the third, having implemented at least an initial version of one optimisation.

By milestone 2 (2008-7-20) I plan to have mostly completed the optimisations in JKit, and be ready to start testing it with some sample programs.

Bibliography

- [1] M. Cline, “[18] const correctness, c++ faq lite.” <http://www.parashift.com/c++-faq-lite/const-correctness.html#faq-18.10>.
- [2] “Function attributes - using the gnu compiler collection (gcc).” <http://gcc.gnu.org/onlinedocs/gcc-4.0.0/gcc/Function-Attributes.html>.
- [3] R. Love, “With a little help from your compiler.” <http://blog.rlove.org/2005/10/with-little-help-from-your-compiler.html>, Oct. 2005.
- [4] M. Metcalf, “Fortran 95,” *SIGPLAN Fortran Forum*, vol. 15, pp. 19–22, 1996.
- [5] m Darvas and P. Muller, “Reasoning about method calls in interface specifications,” *Journal of Object Technology*, vol. 05, pp. 59–85, June 2006.
- [6] D. A. Naumann, “Observational purity and encapsulation,” 2005.