IMPACT

OPTIMISING COMPUTER PROGRAMS

www.cs.nott.ac.uk/~pszgmh/

EPSRC
Engineering and Physical Sciences Research Council

University of Nottingham
UK | CHINA | MALAYSIA

Closing the gap between correctness and efficiency

*A team at the **Functional Programming Laboratory** at the **University of Nottingham** in the UK, is addressing the technically challenging problem of finding formal ways of optimising computer programs, in the **Mind the Gap: Unified Reasoning about Program Correctness and Efficiency** project*

Graham Hutton co-leads the Functional Programming Laboratory at the University of Nottingham in the UK and is project coordinator for the Mind the Gap project, aimed at developing a unified theory of computer program improvement. He explains: 'While most people know software is written using one of a number of computer languages, few are aware that the resulting programs go through a process of transformation and optimisation. Much research has been devoted to developing ways of optimising programs and ensuring they are correct.' He adds: 'While there are now well-developed methods for formally verifying that a program does what it is supposed to, efficiency remains hard to formally assess.' This problem is exacerbated in programs written in high-level functional languages, such as Haskell, which are increasingly used for today's complex multi-authored applications.

The Functional Programming Laboratory has assembled a formidable team to address this neglected area and develop a widely applicable unified theory of reasoning about both correctness and efficiency. Hutton is a leading researcher in functional programming and Jennifer Hackett's PhD set the stage for this project and has been nominated for a number of awards. Martin Handley is a second year PhD student who graduated as the top student on his course with the best individual dissertation. The group is collaborating with five international experts in this field, from both industry and academia. Senior advisors from DARPA and facebook are also on board to provide ideas and guidance. The five-year project is funded by the UK Engineering and Physical Sciences Research Council and is due for completion in May 2021.

TRANSFORMING FUNCTIONAL PROGRAMS

There are many different computer-programming languages, which fall roughly into different groups or paradigms, based on their features. A computer language is a means of instructing a computer to undertake a series of operations and calculations in response to inputs through the software's user interface. Different types of languages suit different software applications. The main programming paradigms are imperative, sometimes called procedural, and functional. The former lists commands on a step-by-step basis and can be long-winded and difficult to manage for large applications. Functional languages on the other hand, use high-level declarative programming, whereby the logic of a function is described, rather than being detailed on a line-by-line basis. In this way, the low-level complexity of the program is effectively hidden, allowing the programmer to concentrate on the architecture and to more readily collaborate with other programmers. This approach in which only the highest levels of the program are shown, is called abstraction.

Once a program has been written, it must be tested and assessed to ensure it works correctly and efficiently. Correctness looks at whether the program behaves as intended, and is usually addressed at the same high level as the program itself, using automated off-the-shelf tools or by handwritten or machine-checked proofs. Efficiency measures look at the different aspects of resource usage including memory, disk space and time to run. Since the complexity of a program is not evident to the programmer, there is no obvious route to calculate efficiency, and by going back to the low-level detail, the advantages of using a high-level language in the first place, are somewhat negated. In 'lazy' languages such as Haskell, expressions are only calculated when needed, which also causes difficulties when exploring efficiency.

UNIFIED IMPROVEMENT THEORY

Hutton says: 'Some high-level techniques for reasoning about the performance of functional programs have been explored, but the resulting techniques are quite different in character to those used for reasoning about correctness. To bridge the divide, we are working on unified theories and techniques that allow both correctness and efficiency to be considered within the same general high-level framework.' Hackett continues: 'We believe improvement theory can be developed to provide this framework. Outside improvement theory, most work on the efficiency of lazy functional languages has been empirical or observational, but we seek a generic approach based on programming

> *Our ultimate goal is to fully automate our improvement theory, allowing proofs of improvements to be derived by the computer without any help from the user*

principles.' Earlier forms of improvement theory went back to basics, looking at how programs are executed on an operational level, whereas Handley says: 'We are taking a different approach, using mathematical models of what programs mean.' In this approach, known as denotational semantics, mathematical objects are used to represent or denote the different parts of the program.

To apply denotational semantics to questions of efficiency, an abstract way of measuring resource usage is needed. Hutton's team has chosen to use metric spaces, which are a flexible concept that can be applied to multiple dimensions of resource usage, such as disk space and time to run. He says: 'We have also developed a resource-aware notion of parametricity, which is a way of looking at the relationships between programs based on the type of data they manipulate.' The team has also already shown that their metric space-based framework enables simpler and easier reasoning about correctness and efficiency. However, there is much more to do. For instance, Hackett explains: 'It is not enough to consider resource usage in one particular situation, we must have a theory that works for all the possible contexts in which a program will be used.' In addition, while the group has developed a generic theory that shows when one program is better than another, the next step is to quantify the improvement by showing how much better one program is over another version.

AUTOMATED TOOLS

As well as developing and refining a unified and generic theory of improvement, an important objective of the project is to create tools to help practitioners apply the theory to their own programs. These will take the form of automated optimisation tools and educational materials in the form of a comprehensive 'Handbook of Improvement', a series of educational videos and a workshop. Handley also hopes the work will: 'prompt further research in this area, and filter through to educators who can include concepts of efficiency in their teaching materials'.

Two important tools have already been produced. The University of Nottingham Improvement Engine (UNIE) is a transformation tool based on existing theory, which helps programmers to determine if their transformations are actually improvements. As Hutton explains: 'The UNIE system has shown that a semi-automated reasoning assistant can help with the detail of proofs, freeing the user to focus on high-level issues.' The AutoBench system is an empirical-based tool, which compares different versions or transformations of programs in terms of their running times. Hutton says: 'To display graphical representations of our comparisons, we had to use statistical techniques and it was surprising to us that there was no standard approach for solving these problems.'

AutoBench is a useful tool, which as well as assisting programmers, is helping the project team study the resource usage profiles of different programming techniques and components. Although memory usage is a much harder concept to generalise, the project aims to produce tools that compare memory usage and processing needs, as well as running times between programs. These tools will also quantify the improvements made in each area and graphically illustrate these through the user interface. Hackett concludes: 'Our ultimate goal is to fully automate our improvement theory, allowing proofs of improvements to be derived by the computer without any help from the user.' The team's work promises to not only open up channels to more reliable, secure and efficient programs, but to also make programming more accessible to non-experts.

Project Insights

FUNDING

This project is funded by the Engineering and Physical Sciences Research Council (EPSRC) under grant reference EP/P00587X/1.

COLLABORATORS

Jennifer Hackett and Martin Handley – University of Nottingham, UK

CONTACT

Graham Hutton
Project Coordinator

T: +44 1159514220
E: graham.hutton@nottingham.ac.uk
W: www.cs.nott.ac.uk/~pszgmh/

PROJECT COORDINATOR BIO

Graham Hutton is Professor of Computer Science at the University of Nottingham, UK, where he co-leads the Functional Programming Lab. His research interests are in developing simple but powerful techniques for writing and reasoning about programs, by recognising and exploiting their underlying mathematical structure. He has served as an editor of the *Journal of Functional Programming*, as Chair of the International Conference on Functional Programming, as Vice-Chair of the ACM Special Interest Group on Programming Languages, and he is an ACM Distinguished Scientist. The second edition of his book *Programming in Haskell* was published by Cambridge University Press in 2016.

EPSRC
Engineering and Physical Sciences Research Council

University of Nottingham
UK | CHINA | MALAYSIA

Impact Objectives

- Demonstrate to the programming world that high-level formal reasoning about program efficiency is a feasible and practical approach
- Develop unified theories and techniques that allow both correctness and efficiency to be considered within the same general high-level framework
- Provide educational and practical tools to assist programmers and researchers in these tasks

Optimising computer programs

***Graham Hutton, Jennifer Hackett** and **Martin Handley** are addressing the long-standing problem of assessing the efficiency of computer programs written in high-level functional languages*


Graham Hutton


Jennifer Hackett


Martin Handley

What problem are you aiming to solve with this project?

GH: Our focus is on reasoning about the performance of computer programs written in high-level functional languages. Most optimisations for functional languages take the form of program transformations, where a program that fits a particular pattern is automatically transformed into an equivalent but more efficient form. However, existing research on program transformation is mostly focused on proving the correctness of transformed programs. Efficiency, by which we mean resource usage, is usually treated in an empirical manner rather than in accordance with an underlying theory.

JH: Our primary objectives are to demonstrate that high-level formal reasoning about program efficiency is both feasible and practical, and to provide educational and practical tools to assist programmers and researchers in these tasks. One of the key benefits of functional languages is their close link with mathematics, giving us the ability to apply reasoning about programs in a formal manner. However, while the high-level nature of functional programming simplifies reasoning about program correctness, it makes it more difficult to evaluate program efficiency. This reasoning gap is particularly pronounced in 'lazy' languages, where the on-demand nature of computations makes this issue particularly challenging.

MH: In a sense, this is a problem of the programming community's own making. By design, high-level languages abstract from low-level details, but efficiency is fundamentally related to this low-level detail. Whereas experienced programmers may circumvent this issue by reasoning about efficiency at the low level, our approach is to enable programmers to address efficiency at a high level, in the same way that correctness is assessed. We are applying the mathematical framework of improvement theory to develop a unified approach to reasoning that allows both correctness and efficiency to be considered within the same general setting.

What is your background and how did you become involved in this particular project?

GH: I became interested in programming at an early age when I began writing computer games for the Sinclair Spectrum. At university I began working on high-level programming languages, which eventually led me to functional languages such as Haskell, which I have worked on ever since. My research work has mainly focused on formal reasoning surrounding program correctness, but I have always maintained an interest in program efficiency. This project combines these two interests.

JH: I fell in love with functional programming as an undergraduate, inspired by how it can enable real brevity and clarity of thought in programs. For me, programming is a collaborative effort between human and computer, and this project came directly out of that idea. A strong theoretical understanding of program optimisation means we can use the computer to do the hard work of making things efficient, freeing up the programmer to consider high-level architectural issues. By reducing the amount of technical knowledge needed to write programs, we can make programming more widely accessible.

MH: When I started programming, I was writing a lot of erroneous code, which was frustrating. Later, when I was taught a mathematical approach to reasoning about program correctness and shown how program efficiency could also be formally tackled using a comparable method, it was a revelation to me! This project offers me the opportunity to further research and develop such reasoning approaches.