# Overflow-checking Elimination in MLton

Daman Morris

djm5830@rit.edu • (412) 586-8168

29 Highland Avenue, Rochester NY, 14623

November 4, 2018

**Abstract**

Typically, when numbers are represented in computers, they are stored with a fixed amount of space. As such, when adding numbers it is possible for the result to exceed the space allotted, a condition called overflow. This is nearly always an error, so most modern programming languages check for this condition and report an appropriate error to the programmer. However, these checks can be costly — hence, much research in the field of compiler optimization has gone into finding new ways to recognize and eliminate superfluous checks. In this project, I intend to perform an in-depth literature review of this research area and select one or more to implement in MLton, a compiler for the SML programming language. This is a project I have previous experience in, and I feel that my undergraduate education in the field of Programming Language theory (PL) has adequately prepared me to produce an in-depth review and a concrete implementation of a significant, and possibly novel, compiler optimization.

## 1 Introduction

Computers usually store numbers with a fixed number of digits (in this context, referred to as *bits*, or binary digits), which is an efficient representation and sufficient for most purposes – depending on the architecture, the "default" amount of precision is either 32 or 64 bits. However, if the result of an arithmetic operation exceeds these bounds, an *overflow* can occur, which, due to the nature of the hardware implementing these operations, typically results in "wrap-around", where a large positive number becomes a large negative number. For example, if `MAX_INT` represents the largest possible positive number, `MAX_INT+1` will result in the largest possible *negative* number.

An overflow typically represents an underlying error in the program. As such, in most programming languages, the language automatically checks for whether an overflow has occurred and, if so, exits with an error. However, this is a somewhat expensive operation, and so the running time of the program can be affected. As such, most programming languages also implement some kind of overflow-checking elimination, where superfluous overflow checks are eliminated. For example, if somewhere in a program we have $x_1 = y + 1$, and later in the program we have $x_2 = y + 1$, and the value of $y$ has not changed, then either $x_1$ and $x_2$ both overflow or both do not, so we do not need to check for overflow in the $x_2$ case.

There are many such potential optimizations — in this project, I aim to perform an in-depth literature review of the optimizations which have been published in the field of compiler optimization. This fits into the category of an advanced study, and will be a focused investigation of this particular subfield. In addition, I will produce a concrete implementation of one such optimization, which will be a specific research end-goal, and will also provide a benefit to the programming community as a whole. I came up with the idea for this project as a result of previous work done on the MLton compiler, which already has several optimizations in this category. As such, this project will further my education objectives by exposing me to advanced research in this field and giving me valuable additional experience working with compilers and compiler optimizations.

## 2 Schedule & Methodology

The bulk of the project will consist of two parts: a literature review in the form of an academic paper summarizing the current state of research in this field, and a concrete implementation of one optimization for the program MLton. Table 1 shows a tentative ten-week schedule, including specific goals for when an initial draft, a final draft, a working implementation, and the final product (paper and implementation). This table shows targets for identifying the papers to review, advisor meetings, and paper reviews. In addition, once a working implementation has been completed, I intend to perform a benchmarking analysis to identify potential performance improvements; these results will ideally be integrated into the final paper.

| **Week** | **Deliverable** | **Description** |
|---|---|---|
| 1 | — | Preliminary literature review, identification of papers to be included. |
| 2 | — | Meeting with advisor, start draft of final paper, identify optimization to be implemented. |
| 3 | — | Continue draft, begin implementation of optimization. |
| 4 | Intitial draft | Finish first draft, submit draft to be reviewed by advisor. |
| 5 | — | Revise draft based on advisor feedback, continue implementation of optimization. |
| 6 | — | Continuing draft revision, implementation. |
| 7 | Final draft | Finish final draft of paper, submit to advisor for review. |
| 8 | Working impl. | Produce working version of implementation, revise final draft based on advisor feedback. |
| 9 | — | Fine-tune implementation, begin benchmarking analysis of results. |
| 10 | Final product | Integrate analysis into paper and produce final paper/optimization. |

Table 1: Tentative ten-week schedule.

## 3 Materials & Budget

The materials required will primarily be published research, and use of university library services to obtain it. No travel will be required. Most papers will be identified over the course of the project; an example of a paper that will likely be included is Exception Analysis in MLton, which details an optimization already specific to MLton [3].

The budget for this project will consist primarily of my housing and income, as well as a small sum that may be used for access to any testing equipment (servers etc.) that turns out to be required; I break this down as follows:

- Housing — \$2,000
- Equipment — \$500
- Income — \$9,500

## 4 Biographical Background

The advisor for this project will be Matthew Fluet, who is a professor at RIT specializing in the field of programming language theory. Professor Fluet is also the lead author of the MLton compiler. He has published several papers in this area, on subjects such as parallel language design and novel linear languages [2, 1].

Professor Fluet, being the principal maintainer of MLton, is highly qualified to direct this work given its involvement with that program. In addition, he has many years of experience in the field of programming language theory, which is of obvious relevance since all of the papers under review will be from that field.

I am currently in my third year at RIT, and will soon (next academic year) be entering a dual BS/MS degree program as a graduate student. I have taken a number of courses in programming language theory and design, and have independent experience in compiler construction, having worked on MLton in the past. I am interested in theoretical computer science in general, including the subfield of programming language theory; I also have academic interests in linguistics (esp. computational linguistics), the history and philosophy of science, and theoretical mathematics. I expect to graduate the next academic year (Spring 2020).

## 5 Conclusion

This project will produce an independent review of the current status of overflow-checking compiler optimizations. This is of immediate benefit to the community of programmers and to the field of programming language theory, as this class of optimizations can greatly improve the performance of generated code. In addition, the process of actually implementing one or more possibly theoretical optimizations serves as a kind of peer review for these papers, and may also uncover novel tactics for the implementation of compiler optimizations, which will be included in the final product. Further, it will provide valuable experience and produce an actual improvement in the quality of the MLton compiler, which is used by university professors and industrial programmers alike to compile the SML programming language.

## References

[1] Amal Ahmed, Matthew Fluet, and Greg Morrisett. “L3: A Linear Language with Locations”. In: *Fundam. Inf.* 77.4 (Dec. 2007), pp. 397–449. ISSN: 0169-2968. URL: `http://dl.acm.org/citation.cfm?id=1365997.1366003`.

[2] Matthew Fluet et al. “Manticore: A Heterogeneous Parallel Language”. In: *Proceedings of the 2007 Workshop on Declarative Aspects of Multicore Programming*. DAMP ’07. Nice, France: ACM, 2007, pp. 37–44. ISBN: 978-1-59593-690-5. DOI: `10.1145/1248648.1248656`. URL: `http://doi.acm.org/10.1145/1248648.1248656`.

[3] Alexander Bjerremand Hansen. Masters. University of Aarhus, 2006.