Evolution of the Spineless Tagless G-Machine

Armin Bernstetter

Seminar Funktionale Programmierung
Julius-Maximilians-Universität, Würzburg

Abstract. The spineless tagless G-machine (STGM) is an abstract machine that is located at the core of the Glasgow Haskell Compiler GHC. Since its creation at the start of Haskell development in early 1990s it has undergone several significant changes. This work aims at showing the evolution of the STGM and overall at providing insight in the workings of the most widely-used Haskell compiler GHC.

1 Introduction

This work provides an insight in the compilation process of the lazy, pure functional programming language Haskell. For this we take a look inside the Glasgow Haskell Compiler, today the most used compiler for Haskell [citation needed]. Located at its core is the *Spineless Tagless G-Machine*, an abstract machine used as a bridge between high level code and machine code.

Described in detail in the 1992 paper *Implementing Lazy functional languages on stock hardware: the Spineless Tagless G-machine* [2], the STGM has undergone several significant changes since then. Two papers highlighting these changes are *Making a Fast Curry: Push/Enter vs. Eval/Apply for Higher-order Languages*[3] and *Faster Laziness Using Dynamic Pointer Tagging*[4]. The former introducing the switch from the *push/enter* evaluation method to the *eval/apply* (see section BLA), the latter introducing dynamic pointer tagging which revokes the "tagless" part in the name of the STGM.

2 Basics

This section introduces some basics about compilers, Haskell and functional programming in general.

2.1 Haskell

What is Haskell, where did it start what does it do? [1] Haskell is a pure functional programming language that emerged in the late 1980s and early 1990s. It was created with the goal of finding a common functional language to improve interactivity and exchange between programmers and researchers since, at the time, many lesser known functional programming languages existed. A committee consisting of (Hudak, SPJ, Wadler etc) was created and met several times until in 1991 the Haskell 1.0 Report was published CITE THE HASKELL 1.0 REPORT

2.2 Compilers

How do compilers work in general

2.3 Abstract machines

What is an Abstract machine?

3 GHC

The Glasgow Haskell Compiler, named after the city where it was initially developed

3.1 Core Language

The core language is a variant of Haskell in where all syntactic sugar is removed and resolved e.g. the do-notation or type aliases.

3.2 The STGM Language

The STG language is a pure functional programming language on its own. Reduced to the very basics it directly interacts with the stack and heap through the STG machine.[citation needed]

syntax details?

heap object layouts?

3.3 $C--$

$C--$ is a language developed by Simon Peyton-Jones as a portable backend-language for compilers. Its name references C and C++. Where C++ can be seen as an extension of the C language, C– is to be thought of as a reduction to a smaller core language. C– in general is made for being generated by compilers and not for being written by programmers.

Why is it needed? C and Stack stuff?

3.4 Backends

GHC supports the generation of Assembly machine code via several backends.

C Backend The C backend uses GCC but is deprecated

Native Code Generator The Native Code Generator is the default way used in GHC.

LLVM LLVM is a modern portable compiler toolchain that was developed as an alternative to the classic GCC toolchain.

Using LLVM in GHC results in similar compilation performance as the NCG but can lead to faster performing executables.

4 STGM in depth

Spineless Spineless refers to the way the code is represented on the machine.

STG programs are not represented as a tree but as a graph. Therefore, in memory a STG program is not a contiguous block of memory but smaller parts of the graph that reference each other.

Tagless The term tagless refers to the way the STG-machine evaluates a heap closure.

[EVERYTHING FROM THE INTRODUCTION OF THE POINTER TAGGING PAPER]

4.1 Evaluation/function calls

Function calls in a lazy functional languages with currying and partial application require special mechanisms in compilers. “Currying” is one of the core principles of lazy functional languages

[INSERT DETAILS ABOUT CURRYING]

HOW TO INCLUDE THE EVALUATION TRACE?

Push/Enter

Eval/Apply

4.2 Dynamic Pointer Tagging

In 2007, Marlow et al.[4] found, that Haskell programs compiled by GHC show mispredicted branches on modern processors. This led to a re-examination of the “tagless” aspect of the GHC. The result were significant performance improvements.

5 Conclusion

6 First Section

6.1 A Subsection Sample

Please note that the first paragraph of a section or subsection is not indented. The first paragraph that follows a table, figure, equation etc. does not need an indent, either.

Subsequent paragraphs, however, are indented.

Sample Heading (Third Level) Only two levels of headings should be numbered. Lower level headings remain unnumbered; they are formatted as run-in headings.

Sample Heading (Fourth Level) The contribution should contain no more than four levels of headings. Table 1 gives a summary of all heading levels.

Table 1. Table captions should be placed above the tables.

Heading level	Example	Font size and style
Title (centered)	**Lecture Notes**	14 point, bold
1st-level heading	**1 Introduction**	12 point, bold
2nd-level heading	**2.1 Printing Area**	10 point, bold
3rd-level heading	**Run-in Heading in Bold.** Text follows	10 point, bold
4th-level heading	*Lowest Level Heading.* Text follows	10 point, italic

Displayed equations are centered and set on a separate line.

$$x + y = z \tag{1}$$

Please try to avoid rasterized images for line-art diagrams and schemas. Whenever possible, use vector graphics instead (see Fig. 1).

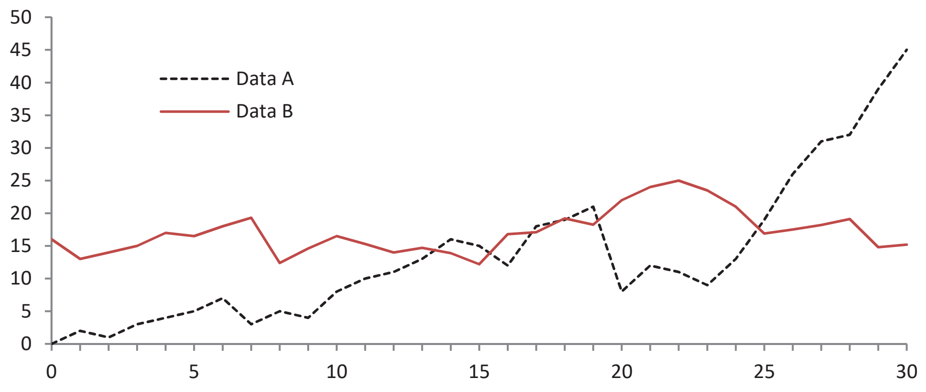

Fig. 1. A figure caption is always placed below the illustration. Please note that short captions are centered, while long ones are justified by the macro package automatically.

Theorem 1. *This is a sample theorem. The run-in heading is set in bold, while the following text appears in italics. Definitions, lemmas, propositions, and corollaries are styled the same way.*

Proof. Proofs, examples, and remarks have the initial word in italics, while the following text appears in normal font.

For citations of references, we prefer the use of square brackets and consecutive numbers. Citations using labels or the author/year convention are also acceptable. The following bibliography provides a sample reference list with entries for journal articles [**?**], an LNCS chapter [**?**], a book [**?**], proceedings without editors [**?**], and a homepage [**?**]. Multiple citations are grouped [**?**,**?**,**?**], [**?**,**?**,**?**,**?**].

References

1. Hudak, P., Hughes, J., Peyton Jones, S., Wadler, P.: A history of haskell: being lazy with class. In: Proceedings of the third ACM SIGPLAN conference on History of programming languages. pp. 12–1. ACM (2007)
2. Jones, S.L.P.: Implementing lazy functional languages on stock hardware: the spineless tagless g-machine. Journal of functional programming **2**(2), 127–202 (1992)
3. Marlow, S., Jones, S.P.: Making a fast curry: push/enter vs. eval/apply for higher-order languages. In: ACM SIGPLAN Notices. vol. 39, pp. 4–15. ACM (2004)
4. Marlow, S., Yakushev, A.R., Peyton Jones, S.: Faster laziness using dynamic pointer tagging. In: Acm sigplan notices. vol. 42, pp. 277–288. ACM (2007)