Summer of Haskell Project

Add primops to expand the (boxed) array API.

Vilem-Benjamin Liepelt

Abstract

GHC is an industry-strength compiler for Haskell. I am proposing to add primitives that allow more efficient construction of boxed `Array#`s, the underlying data structure for many popular data structures. We expect for example to speed up array concatenation by a factor of up to 2 by removing an unnecessary `memset` operation.

Contents

Project Title

Add primops to expand the (boxed) array API.

Personal Details

Please refer to my CV which is appended to this document.

Deliverables

The deliverables are outlined in a proposal that I co-authored with Andreas Klebinger, which can be found here: summer.haskell.org/ideas.html#array-

primops. I have also included it verbatim below. In the following I give a draft timeline.

Array concatenation primitive (5 weeks, required)

- Get to know the codebase.
- Add a new operation `concatArray#s` to `GHC.Exts`.
- Add the necessary cmm primops.

Testing and benchmarking (1 week, required)

- Benchmark simple programs.
- Test with downstream users, e.g. `vector`.
- Ensure that documentation is up to date.

API for modifying array sizes (3 weeks, required)

At this point I expect to be able to build on my learnings from above, hence making this stage faster to implement.

- Add `sliceArray#` and `growArray#` primops which combine the copy and initialization step.
- Extend tests, documentation and benchmarks.

API for creating arrays from known elements (3 weeks, optional)

If I manage to keep to the above (ambitious) timeline, then I will attempt the following.

- Provide Array literals which allow giving the size and contents of an array as a single construct. E.g. `arrayFrom# (# a, b, c #)`
- Ensure we are constant-folding the above.
- Extend tests, documentation and benchmarks.

Project writeup (interleaved with above)

I will keep a log of what I am doing, the issues I encounter and how I feel about my progress.

My PhD

I am doing a PhD at the School of Computing, University of Kent. I am a founding member of and contributor to the Granule project, where we develop and implement a linear functional language with graded modal types, a generalisation of Linear Logic:

Project page	granule-project.github.io/
Source repository	github.com/granule-project/granule
ICFP 2019 paper	kar.kent.ac.uk/74450/
ICFP 2019 talk	youtu.be/JikTzq6kdjE

I am currently working on a paper that explores array interfaces which allow safe and convenient stateful array manipulation, enabled by Granule's linear type system. I also show that Granule is a viable laboratory for the linear types community, e.g. by implementing Ahmed et al.'s *L^3: A Linear Language with Locations*.

The Granule type checker and interpreter are implemented in Haskell. I have presented our work at various conferences and institutions, not least to SPJ himself at Microsoft Research, Cambridge.

For my BSc thesis I added GADTs to Granule for which I was awarded "best final year project".

My Haskell Experience

In addition to my use of Haskell in my PhD, I am currently using Haskell professionally at Future Finance. I manage our collaboration with a renowned Haskell consultancy. I have refactored crucial bits of our Haskell infrastructure and ported it to AWS Lambda.

In total I have 4 years of experience with Haskell, 2 years of which I have been using it in an industrial setting.

Project Proposal

(Copied verbatim from summer.haskell.org/ideas.html#array-primops)

Arrays are the bedrock on which popular data structures like Vector are implemented. However the API provided by GHC is quite limited. This project would expand the API to fill some of these gaps.

All examples are given for `Array#` but apply to `SmallArray#` as well.

Create boxed arrays from existing ones

Motivation: Creating (`Small`)`Array#`s efficiently in GHC Haskell stands in tension with the garbage collector as we must never encounter uninitialised slots during a collection. Thus, for safety, array creation primitives have to conservatively intialise array slots at the cost of performance.

Currently to make a new `Array#` `zs` out of existing `Array#`s `xs` and `ys`, we have to call the following operations (analogously for `SmallArray#`):

1. `let zs = newArray# ...`
2. `copyArray# xs ... zs ...`
3. `copyArray# ys ... zs ...`

Proposed idea: We propose to add a new array primitive that allows copying existing arrays into a new array while bypassing any unnecessary initialisation step.

1. `let zs = concatArray#s xs ys`

Provide an API for modifying array sizes

Motivation: Creating a new array from an subset of the element inside an array currently requires us to first create a new array, and then copy the elements over.

Proposed idea: Provide `sliceArray#` and `growArray#` primops which combine the copy and initialization step.

These could be used for example in the implementation of `grow` from the *vector* package.

Create boxed arrays from known elements.

Motivation: Currently creating an array of fully known contents consists of two steps. Creating an array initializing it with default values and then filling in the actual contents.

Proposed idea: Provide Array literals which allow giving the size and contents of an array as a single construct. E.g. `arrayFrom# (# a, b, c #)`

This would allow us to completely eliminate the initialization with default values completely.

Vilem-Benjamin Liepelt

Software Engineer & Researcher

I believe in

- formal & semi-formal methods
- teamwork & good communication
- open-mindedness & prototyping

Personal Details

Address Dakar Street 11, Ashdod, Israel

Phone
- +44 7490 128688
- +972 53 3544609

Email
- mail@vilem.net
- vl200@kent.ac.uk

www
- github.com/buggymcbugfix
- twitter.com/buggymcbugfix
- cs.kent.ac.uk/people/rpg/vl200/

References Available on request

Education

since 2018 **PhD at School of Computing, University of Kent (Ongoing):** My research is in graded modal and linear type systems for enforcing static guarantees about resources and security in computer programs. I am a contributor to the Granule programming language and its research implementation.

2015–2018 **BSc (Hons.) Computer Science and Artificial Intelligence, University of Kent:** Classification: First-class honours. I also received the following awards:

- School of Computing prize for highest overall performance
- Faculty of Sciences Rotary Award for distinguished performance
- Computer Science Project Prize for distinguished final year project

2011–2014 **French and German Law, Universität zu Köln & Université Paris I (Panthéon-Sorbonne):** Completed until foundation stage (Grundstudium). Classification: 8.9/18 ("Above average").

Work

since 11/2019 **Future Finance:** Senior Software Engineer at a financial startup with over 100 employees. I maintain the credit policy and loan scheduling service, written in Haskell.

- Saved the company £0.5M annual costs within 3 months in this role.
- Managing our external contractors who work with us via a renowned Haskell consultancy.
- Rewrote a critical legacy application that even non-engineers now feel comfortable interacting with.
- Wrote a library for deploying Haskell on AWS Lambda and integrating with Amazon API Gateway.
- Moved the deployment of a critical Haskell application from an in-house server to AWS Lambda.
- Received the Future Finance Pillar Award in February 2020 for *"Acting like an owner"*.
- Integrating and refining a fraud detection model.
- Building up a Haskell team. Referring and interviewing potential applicants.
- Replacing a data caching microservice written in Ruby.

12/2018–10/2019 **Adjoint Inc.:** Part-time Research Engineer at a fintech startup. I worked on the type checker and interpreter (written in Haskell) for the now open source project FCL, a DSL for describing business/financial workflows in the context of distributed ledgers.

- Extended the language with records.
- UX improvements, such as more helpful error messages.
- Fixed soundness issues in the operational semantics.
- Implemented non-lossy arithmetic with dependently typed operations, ensuring accurate rounding.
- Extensive use of generative testing using QuickCheck.
- Supervised a successful summer internship.

1–4/2019 **University of Kent:** TAing the *Functional and Concurrent Programming* BSc/MSc class. I tutored two classes of ca. 30 students each, helping them understand functional programming and the actor model via Erlang.

5–9/2018 **Adjoint Inc.:** I worked mainly on extending the FCL type checker and interpreter (written in Haskell) as part of an internship. I extended the DSL to allow writing workflows with concurrent actions. Other improvements include additions of roles (permissioning) and temporal preconditions.

7–8/2018 **Freelance App Developer:** I implemented a React Native-based mobile app for ESL accent training research for the linguistics department at the University of Kent.

7–8/2017 **Google:** Software engineering internship. I extended end user preferences of the AdSense frontend, written in Java. AdSense is the world's most popular advertising network. I learnt how to fillet fish.

5–6/2017 **Cambridge Fortran Infrastructure:** Research internship supervised by Dominic Orchard. I ported the constraint solving for the static analysis of scientific Fortran code to an external SMT solver (Z3) using the Haskell library SBV.

2013–2018 **Freelance Consultant:** Training and consultancy for parametric and procedural pattern drafting with the Grafis CAD suite of software solutions for the fashion industry. My main customer base was in France and the UK.

A Selection of Publications & Talks

ICFP 2019 **Paper & talk:** *Quantitative program reasoning with graded modal types*

ICFP 2019 **Tutorial:** *Fine-grained program reasoning using linear and graded modal types*

TypeLevel 2019 **Talk:** *Taking Resources to the Type Level*

LSS Cambridge **Talk:** *Resource-oriented programming with graded modal types*

POPL 2019 **Tutorial:** *Linear and graded modal types for fine-grained program reasoning*

Unpublished **Paper:** *Scrap Your Reprinter: a datatype generic algorithm for layout-preserving refactoring*

SREPLS 7 **Talk:** *Granule: A language for fine-grained program reasoning with graded modal types*

TLLA 2017 **Extended abstract:** *Gram: A linear functional language with graded modal types*

More about me

Languages Deutsch ∧ English ∧ Français ∧ Português ∧ קצת עברית