

Proving Compiler Correctness with Dependent Types

João Paulo Pizani Flor
Wout Elsinghorst

Department of Information and Computing Sciences
Utrecht University

Wednesday 16th April, 2014

Introduction

- Context/Terminology
- Compiler correctness
- Sharing
- Goals

Implementation (code)

- Basic correctness
- Lifting to sharing setting

Conclusions



Universiteit Utrecht

Table of Contents

Introduction

- Context/Terminology
- Compiler correctness
- Sharing
- Goals

Implementation (code)

- Basic correctness
- Lifting to sharing setting

Conclusions

Introduction

- Context/Terminology
- Compiler correctness
- Sharing
- Goals

Implementation (code)

- Basic correctness
- Lifting to sharing setting

Conclusions



Table of contents

Introduction

Context/Terminology

Compiler correctness

Sharing

Goals

Implementation (code)

Basic correctness

Lifting to sharing setting

Conclusions

Introduction

Context/Terminology

Compiler correctness

Sharing

Goals

Implementation (code)

Basic correctness

Lifting to sharing setting

Conclusions



Source language, Target language

- ▶ Example source code (expression language):

```
Add (Val 1) (Add (Val 1) (Val 3))
```

- ▶ Example target code (for a stack machine):

```
PUSH 1 >> PUSH 1 >> PUSH 3 >> ADD >> ADD
```

Introduction

Context/Terminology

- Compiler
- correctness
- Sharing
- Goals

Implementation (code)

- Basic correctness
- Lifting to sharing
- setting

Conclusions



Table of contents

Introduction

Context/Terminology

Compiler correctness

Sharing

Goals

Implementation (code)

Basic correctness

Lifting to sharing setting

Conclusions

Introduction

Context/Terminology

Compiler correctness

Sharing

Goals

Implementation (code)

Basic correctness

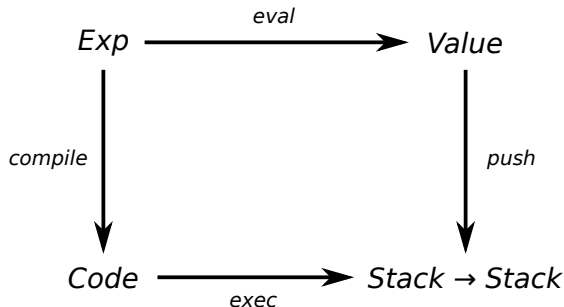
Lifting to sharing setting

Conclusions



What does "correct" mean?

- ▶ Both semantics (before and after compilation) should be "equivalent"
- ▶ Compiling then executing must give the same result as direct evaluation



Introduction

Context/Terminology

Compiler correctness

Sharing

Goals

Implementation (code)

Basic correctness

Lifting to sharing setting

Conclusions



Universiteit Utrecht

Reference paper

- ▶ "A type-correct, stack-safe, provably correct expression compiler in Epigram"
 - James McKinna, Joel Wright
- ▶ Basic ideas and proofs, which we extended. . .

Introduction

Context/Terminology

Compiler correctness

Sharing

Goals

Implementation (code)

Basic correctness

Lifting to sharing
setting

Conclusions



Universiteit Utrecht

Table of contents

Introduction

Context/Terminology

Compiler correctness

Sharing

Goals

Implementation (code)

Basic correctness

Lifting to sharing setting

Conclusions

Introduction

Context/Terminology

Compiler correctness

Sharing

Goals

Implementation (code)

Basic correctness

Lifting to sharing setting

Conclusions



Extending the source language

- ▶ More "realistic" languages have sharing constructs
- ▶ We wanted the "simplest possible" extension with sharing behaviour.

- Chosen extension: `if_then_else` + sequencing

```
if c then t else e >> common-suffix
```

- ▶ The "naïve" compile function will duplicate the suffix
- ▶ Having Bytecode defined as graph (structured graph) instead of tree would solve this problem
 - But proofs would be more complex

Introduction

Context/Terminology

Compiler correctness

Sharing

Goals

Implementation (code)

Table of contents

Introduction

Context/Terminology

Compiler correctness

Sharing

Goals

Implementation (code)

Basic correctness

Lifting to sharing setting

Conclusions

Introduction

Context/Terminology

Compiler correctness

Sharing

Goals

Implementation (code)

Basic correctness

Lifting to sharing setting

Conclusions



Reference paper

- ▶ "Proving Correctness of Compilers using Structured Graphs"
 - Patrick Bahr (visiting researcher)

Introduction

Context/Terminology

Compiler correctness

Sharing

Goals

Implementation (code)

Basic correctness

Lifting to sharing
setting

Conclusions



Universiteit Utrecht

Our project's goals

- ▶ Integrating the best of both “reference” papers
- ▶ Our contributions:
 - (Simplest possible) language extension showing sharing behaviour.
 - Proof of correctness for the **stack-safe** “naïve” compiler
 - The one that just duplicates code.
 - A way to lift this **stack-safe** “naïve” correctness proof
 - Into a proof concerning the more **efficient** compiler.

Introduction

Context/Terminology

Compiler correctness

Sharing

Goals

Implementation (code)

Basic correctness

Lifting to sharing
setting

Conclusions



Universiteit Utrecht

Table of contents

Introduction

- Context/Terminology
- Compiler correctness
- Sharing
- Goals

Implementation (code)

- Basic correctness
- Lifting to sharing setting

Conclusions

Introduction

- Context/Terminology
- Compiler correctness
- Sharing
- Goals

Implementation (code)

- Basic correctness**
- Lifting to sharing setting

Conclusions



Source

Source types:

`data Tys : Set where`

`Ns : Tys`

`Bs : Tys`

Source terms (snippet):

`data Src : (t : Tys) → (z : Sizes) → Set where`

`vs : ∀ {t} → (v : { t }) → Src t 1`

`_+_s_ : (e1 e2 : Src Ns 1) → Src Ns 1`

Denotational semantics (snippet):

$\llbracket _ \rrbracket : \{t : \text{Ty}_s\} \{z : \text{Size}_s\} \rightarrow (e : \text{Src } t \ z) \rightarrow \text{Vec } \{ t \} \ z$

$\llbracket v_s \ v \rrbracket = \llbracket v \rrbracket$

$\llbracket e_1 \ +_s \ e_2 \rrbracket = \llbracket e_1 \rrbracket + \llbracket e_2 \rrbracket$

Introduction

Context/Terminology

Compiler
correctness

Sharing

Goals

Implementation
(code)

Basic correctness

Lifting to sharing
setting

Conclusions



Universiteit Utrecht

Bytecode

Typed stack:

$\text{StackType} : \text{Set}$

$\text{StackType} = \text{List Ty}_s$

data $\text{Stack} : \text{StackType} \rightarrow \text{Set}$ **where**

$\epsilon : \text{Stack } []$

$_ \nabla _ : \forall \{t\ s'\} \rightarrow \{t\} \rightarrow \text{Stack } s' \rightarrow \text{Stack } (t :: s')$

Typed bytecode (snippet):

data $\text{Bytecode} : \text{StackType} \rightarrow \text{StackType} \rightarrow \text{Set}$ **where**

$\text{SKIP} : \forall \{s\} \rightarrow \text{Bytecode } s\ s$

$\text{PUSH} : \forall \{t\ s\} \rightarrow (x : \{t\}) \rightarrow \text{Bytecode } s\ (t :: s)$

$\text{ADD} : \forall \{s\} \rightarrow \text{Bytecode } (\mathbb{N}_s :: \mathbb{N}_s :: s)\ (\mathbb{N}_s :: s)$

Introduction

Context/Terminology

Compiler correctness

Sharing

Goals

Implementation
(code)

Basic correctness

Lifting to sharing
setting

Conclusions



Universiteit Utrecht

Compiler correctness

Introduction

- Context/Terminology
- Compiler correctness
- Sharing
- Goals

Implementation (code)

- Basic correctness**
- Lifting to sharing setting

Conclusions



Table of contents

Introduction

Context/Terminology

Compiler correctness

Sharing

Goals

Implementation (code)

Basic correctness

Lifting to sharing setting

Conclusions

Introduction

Context/Terminology

Compiler correctness

Sharing

Goals

Implementation (code)

Basic correctness

Lifting to sharing setting

Conclusions



Tree fixpoints

Introduction

- Context/Terminology
- Compiler correctness
- Sharing
- Goals

Implementation (code)

- Basic correctness
- Lifting to sharing setting**

Conclusions



Bytecode Tree Representation

Introduction

- Context/Terminology
- Compiler correctness
- Sharing
- Goals

Implementation (code)

- Basic correctness
- Lifting to sharing setting**

Conclusions



Correctness on Trees

Introduction

- Context/Terminology
- Compiler correctness
- Sharing
- Goals

Implementation (code)

- Basic correctness
- Lifting to sharing setting**

Conclusions



Graphs

Introduction

- Context/Terminology
- Compiler correctness
- Sharing
- Goals

Implementation (code)

- Basic correctness
- Lifting to sharing setting**

Conclusions



Bytecode Graph Representation

Introduction

Context/Terminology

Compiler
correctness

Sharing

Goals

Implementation (code)

Basic correctness

**Lifting to sharing
setting**

Conclusions



Universiteit Utrecht

Achieved

- ▶ Agda “framework” for deriving compiler correctness proofs
 - Compilers with **typed source** and **typed target**
 - Given correctness of a “naïve” compiler, derive correctness of “optimized” version
- ▶ Correctness proof for an expression language (with sequencing)
 - As “instance” of this framework

Introduction

Context/Terminology

Compiler correctness

Sharing Goals

Implementation (code)

Basic correctness

Lifting to sharing
setting

Conclusions



Universiteit Utrecht

Agda limitations we faced

- ▶ Strict positivity requirement
 - When defining fixed point type operators
- ▶ Totality checker
 - When defining folds
- ▶ Type checking with positivity check disabled made debugging hard
 - Stack overflow, memory consumption

Introduction

Context/Terminology

Compiler correctness

Sharing

Goals

Implementation (code)

Basic correctness

Lifting to sharing
setting

Conclusions



Yet to be done

- ▶ Sequence clause of “basic” (non-lifted) correctness proof
- ▶ Prove a final lemma to complete the lifting (fusion law)

Introduction

- Context/Terminology
- Compiler correctness
- Sharing
- Goals

Implementation (code)

- Basic correctness
- Lifting to sharing setting

Conclusions



Thank you!

Questions?

