Agda Backends: A survey and a UHC backend prototype

Author: Philipp Hausmann
<p.hausmann@students.uu.nl>

Supervisors: Wouter Swierstra
<w.s.swierstra@uu.nl>
Atze Dijkstra
<atze@uu.nl>

Department of Information and Computing Sciences
Utrecht University

Wednesday 26th November, 2014

Universiteit Utrecht

Table of Contents

Agda Introduction

- Why dependent types?

Universiteit Utrecht

Agda Introduction

- Why dependent types?
- ```
  head :: forall a . List a -> a
  head (x:xs) = x
  head [] = error "something went wrong..."
  ```


Universiteit Utrecht
  ```

Agda Introduction

- Why dependent types?
- ```
  head :: forall a . List a -> a
  head (x:xs) = x
  head [] = error "something went wrong..."
  ```
- Runtime crashes are possible in Haskell!

Universiteit Utrecht
  ```

Agda Introduction

- How to make sure at compile time that this doesn't happen?
- We need to encode the length of lists in the type

Agda Introduction

- How to make sure at compile time that this doesn't happen?
- We need to encode the length of lists in the type
  - ```
    data Nat : Set where
      zero : Nat
      succ : Nat → Nat
    ```


Universiteit Utrecht
    ```

Agda Introduction

- How to make sure at compile time that this doesn't happen?
- We need to encode the length of lists in the type
  - ```
    data Nat : Set where
      zero : Nat
      succ : Nat → Nat
    ```
  - ```
    data Vec : (A : Set) → (n : Nat) → Set where
      nil  : ∀ {A} → Vec A zero
      cons : ∀ {A n} → A → Vec A n → Vec A (succ n)
    ```


Universiteit Utrecht

Cont.

- We can now write the head function in Agda
  - $\text{head1} : \forall \{A\ n\} \to \text{Vec}\ A\ n \to A$
    $\text{head1}\ (\text{cons}\ x\ xs) = x$
    $\text{head1}\ \text{nil} = ????$

Universiteit Utrecht

Cont.

- We can now write the head function in Agda
  - $\text{head1} : \forall \{A\ n\} \to \text{Vec}\ A\ n \to A$
    $\text{head1}\ (\text{cons}\ x\ xs) = x$
    $\text{head1}\ \text{nil} = ????$

- This will not type check!

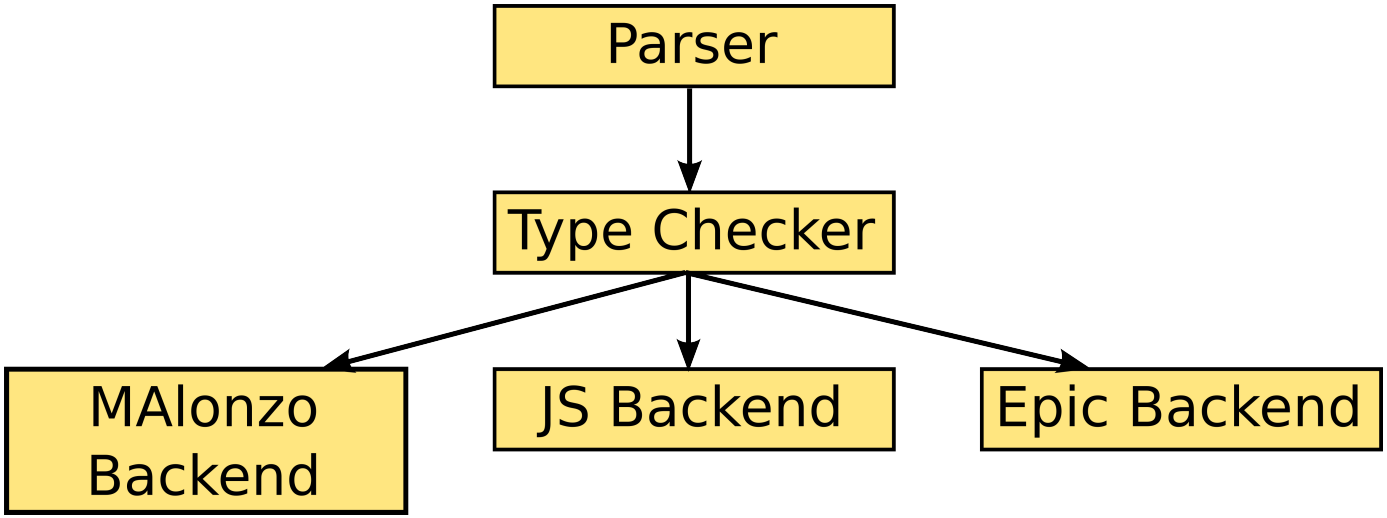
Cont.

- We can now write the head function in Agda
  - $\mathsf{head1} : \forall\ \{A\ n\} \to \mathsf{Vec}\ A\ n \to A$
    $\mathsf{head1}\ (\mathsf{cons}\ x\ xs) = x$
    $\mathsf{head1}\ \mathsf{nil} = ????$
- This will not type check!
  - $\mathsf{head2} : \forall\ \{A\ n\} \to \mathsf{Vec}\ A\ (\mathsf{succ}\ n) \to A$
    $\mathsf{head2}\ (\mathsf{cons}\ x\ xs) = x$
- The typechecker now knows that the nil-case cannot happen!

Agda Summary

- Values can be used as types
- Types cannot influence value of an expression
- Functions need to be total

Agda Architecture



Universiteit Utrecht

MAlonzo backend

MAlonzo backend

- Targets Haskell
- Maintained
- Relies on GHC for optimizations

MAlonzo - FFI

- Provides simple FFI to haskell
- Very limited
  - No class support
  - Can't export Agda datatypes
  - Not automatic

MAlonzo - FFI

```
{-# IMPORT Data.List #-}

data List : (A : Set) -> Set where
  nil : ∀ {A} → List A
  cons : ∀ {A} → A → List A → List A
{-# COMPILED_DATA List Data.List nil cons #-}

postulate
  head : ∀ {A} → List A -> A
{-# COMPILED head Data.List.head #-}
```


MAlonzo - Code Generation

```
vecToStr : ∀ {A m} → (A → String)
    → Vec A m → String
vecToStr f [] = ""
vecToStr f (x :: xs) = ", " ++ ((f x)
    ++ (vecToStr f xs))
```


Universiteit Utrecht

MAlonzo - Code Generation

```
d55 v0 v1 v2 v3
  = MAlonzo.RTE.mazCoerce
      (d_1_55 (MAlonzo.RTE.mazCoerce v0)
        (MAlonzo.RTE.mazCoerce v1)
        (MAlonzo.RTE.mazCoerce v2)
        (MAlonzo.RTE.mazCoerce v3))
  where d_1_55 v0 v1 v2 (C51 v3 v4 v5)
          = MAlonzo.RTE.mazCoerce
              (d33 (MAlonzo.RTE.mazCoerce ", ")
                (MAlonzo.RTE.mazCoerce
  (d33 (MAlonzo.RTE.mazCoerce (v2 (MAlonzo.RTE.mazCoerce v4)))
    (MAlonzo.RTE.mazCoerce
      (d55 (MAlonzo.RTE.mazCoerce v0) (MAlonzo.RTE.mazCoerce v3)
        (MAlonzo.RTE.mazCoerce v2)
        (MAlonzo.RTE.mazCoerce v5))))))
      d_1_55 v0 v1 v2 v3 = MAlonzo.RTE.mazIncompleteMatch name55
```


Universiteit Utrecht

MAlonzo - Summary

- Produces 'strange' haskell code
- Can lead to size blow-up
  - 84 lines Agda - 250'000 lines Haskell - 300 Mb executable (CITE)

JS backend

Universiteit Utrecht

JS backend

- Targets Javascript
- Not maintained
- Very similar to MAlonzo

Epic backend

Epic backend

- Targets Epic
- Not maintained

Universiteit Utrecht

Epic

- Untyped-lambda calculus with some extensions
- Intended as building block for compilers
- Also not maintained

Epic Language

Epic Language		
t	$::=$	x
	$\mid$	$t\ \vec{t}$
	$\mid$	$\lambda x \to t$
	$\mid$	Con $i\ \vec{t}$
	$\mid$	if t then t else t
	$\mid$	case t of $\vec{alt}$
	$\mid$	let $x = t$ in t
	$\mid$	lazy t
	$\mid$	t ! i
	$\mid$	i

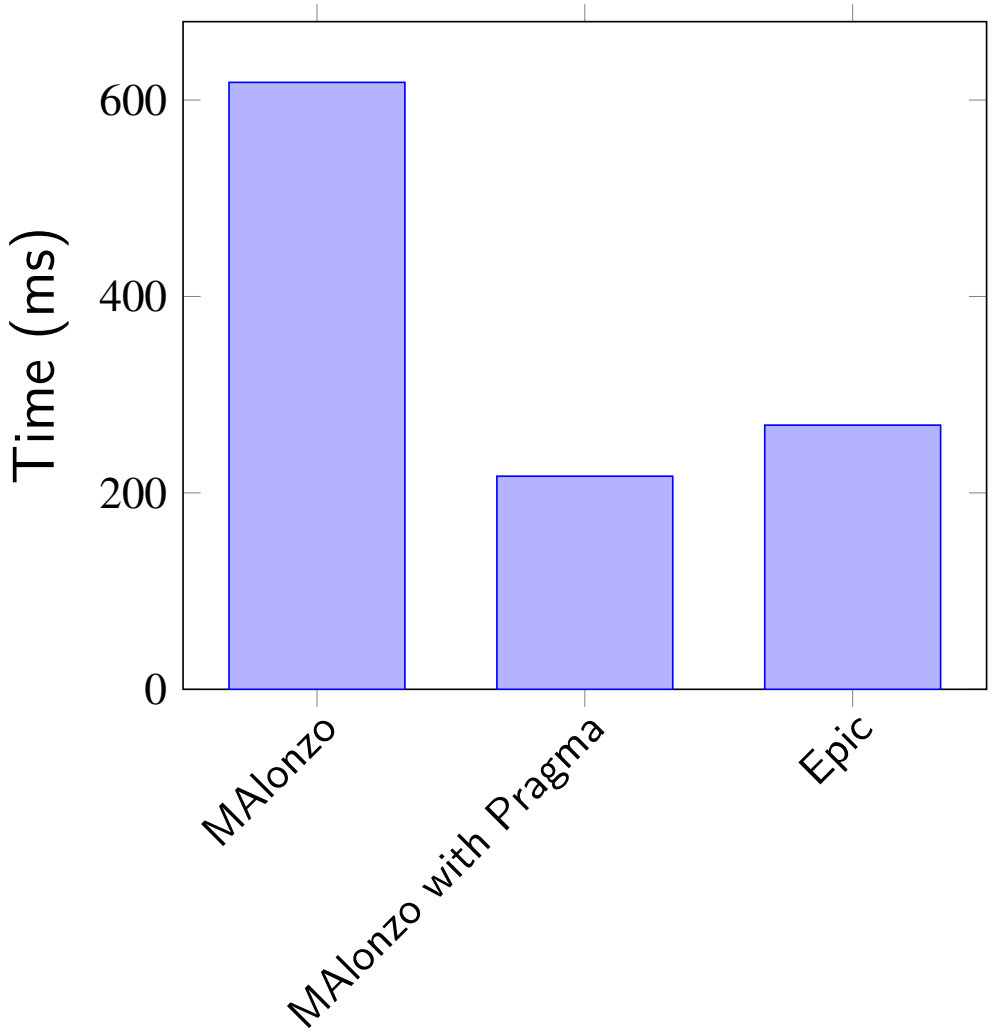
Universiteit Utrecht

Epic - Nat Optimizations

```
data Nat : Set where
  zero : Nat
  succ : Nat -> Nat
{-# BUILTIN NATURAL Nat #-}
```

- Naive translation is horribly slow
- Can be transformed into arbitrary precision Integers
- Automatic detection of Nat-like datatypes

Nat Performance

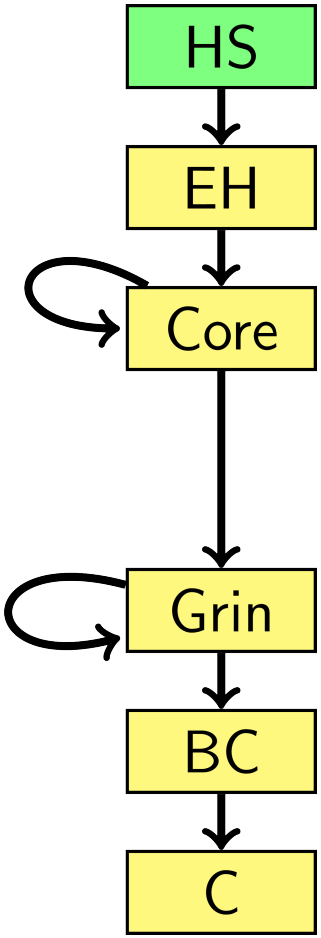


Universiteit Utrecht

Comparison

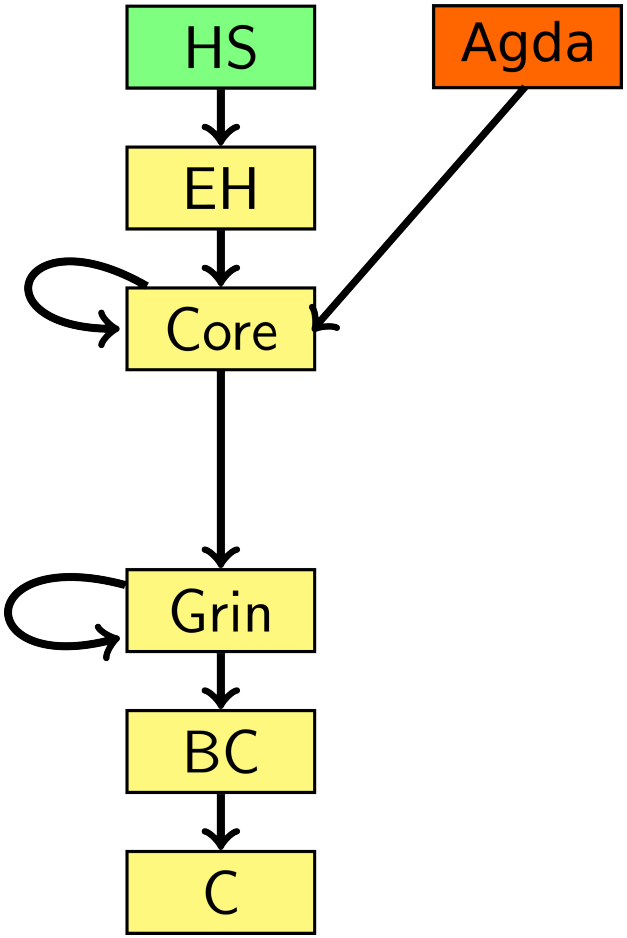
Universiteit Utrecht

UHC Compiler



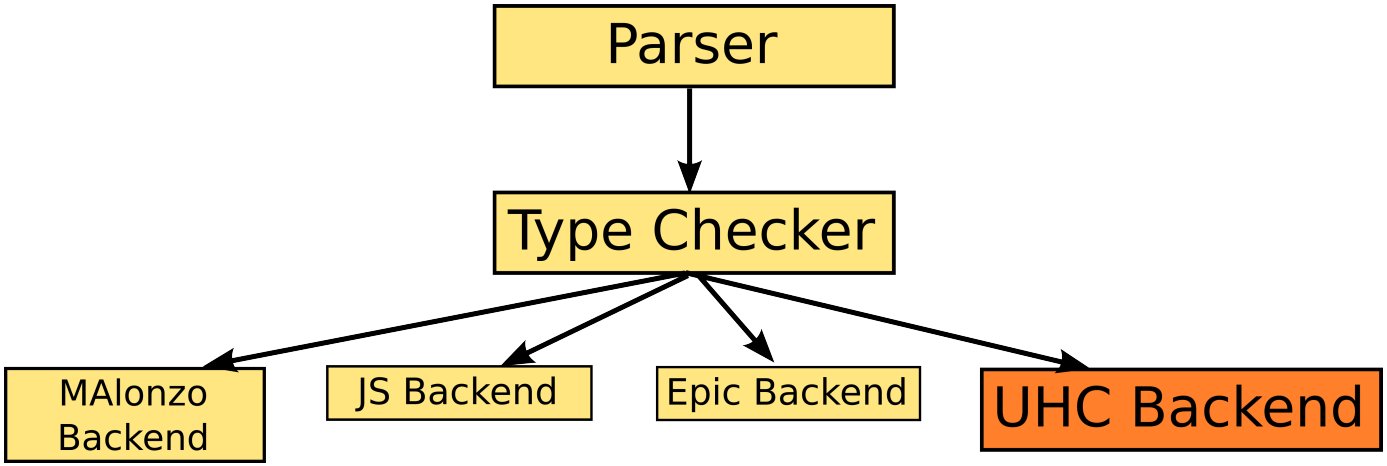
Universiteit Utrecht

UHC Compiler



Universiteit Utrecht

UHC Backend



Universiteit Utrecht

Epic vs UHC Core

Epic Language			UHC Core		
t	$::=$	x	t	$::=$	x
	$\mid$	$t\ \vec{t}$		$\mid$	$t\ t$
	$\mid$	$\lambda x \to t$		$\mid$	$\lambda x \to t$
	$\mid$	Con $i\ \vec{t}$		$\mid$	Con $i\ \vec{t}$
	$\mid$	if t then t else t			
	$\mid$	case t of $\vec{alt}$		$\mid$	case t of $\vec{alt}$
	$\mid$	let $x = t$ in t		$\mid$	let $x = t$ in t
				$\mid$	let! $x = t$ in t
	$\mid$	lazy t			
	$\mid$	i		$\mid$	i

Universiteit Utrecht

UHC Backend - Challenges

- Agda is a moving target
- UHC Core was not intended as public API
- Undocumented assumptions inside UHC

UHC Backend - Challenges

- Agda is a moving target
- UHC Core was not intended as public API
- Undocumented assumptions inside UHC

```
case x of
  []      -> a
  (x:xs)  -> b
```

is not the same as

```
case x of
  (x:xs)  -> b
  []      -> a
```


UHC Backend - What works?

- (Dependent) dataypes, functions
- Compiling single Agda modules
- Agda - Haskell FFI, but involves manual work

Demonstration

UHC Backend - Future work

- Support whole Agda language
  - Multiple modules
  - Complete IO bindings
  - Agda Standard Library
- Optimizations
- Improve Agda - Haskell FFI
- Agda support for Cabal
- Contracts for FFI

Thank you!

Questions?

Universiteit Utrecht

References I

Universiteit Utrecht