Ghostbuster:
A Tool for Simplifying and Converting GADTs

Trevor L. McDonell\textsuperscript{3-1}
Timothy A. K. Zakian\textsuperscript{3-2}
Matteo Cimini\textsuperscript{3}
Ryan R. Newton\textsuperscript{3}

\textsuperscript{1}University of New South Wales \hspace{1cm} \textsuperscript{2}University of Oxford \hspace{1cm} \textsuperscript{3}Indiana University
we should teach our students parallelism from the outset!

end of Moore’s rule, blah blah blah blah...
Embedded language for high-performance array computations
maybe they can hack on Accelerate?
however...

as a research project explores extensive use of type-indexed datatypes
(do { GHC.Read.expectP (Text.Read.Lex.Ident "Scanr'"));
a1 ← Text.ParserCombinators.ReadPrec.step GHC.Read.readPrec;
a2 ← Text.ParserCombinators.ReadPrec.step GHC.Read.readPrec;
a3 ← Text.ParserCombinators.ReadPrec.step GHC.Read.readPrec;
    .... })
Text.ParserCombinators.ReadPrec.+++
(Text.ParserCombinators.ReadPrec.prec 10
(do { GHC.Read.expectP (Text.Read.Lex.Ident "Scanr1");
a1 ← Text.ParserCombinators.ReadPrec.step GHC.Read.readPrec;
a2 ← Text.ParserCombinators.ReadPrec.step GHC.Read.readPrec;
    return (Scanr1 a1 a2) })
Text.ParserCombinators.ReadPrec.+++
(Text.ParserCombinators.ReadPrec.prec 10
(do { GHC.Read.expectP (Text.Read.Lex.Ident "Permute");
a1 ← Text.ParserCombinators.ReadPrec.step GHC.Read.readPrec;
a2 ← Text.ParserCombinators.ReadPrec.step GHC.Read.readPrec;
a3 ← Text.ParserCombinators.ReadPrec.step GHC.Read.readPrec;
    .... })
Text.ParserCombinators.ReadPrec.+++
(Text.ParserCombinators.ReadPrec.prec 10
(do { GHC.Read.expectP (Text.Read.Lex.Ident "Backpermute");
a1 ← Text.ParserCombinators.ReadPrec.step GHC.Read.readPrec;
a2 ← Text.ParserCombinators.ReadPrec.step GHC.Read.readPrec;
a3 ← Text.ParserCombinators.ReadPrec.step GHC.Read.readPrec;
    .... })
Text.ParserCombinators.ReadPrec.+++
(Text.ParserCombinators.ReadPrec.prec 10
(do { GHC.Read.expectP (Text.Read.Lex.Ident "Stencil");
a1 ← Text.ParserCombinators.ReadPrec.step GHC.Read.readPrec;
a2 ← Text.ParserCombinators.ReadPrec.step GHC.Read.readPrec;
a3 ← Text.ParserCombinators.ReadPrec.step GHC.Read.readPrec;
    .... })
Text.ParserCombinators.ReadPrec.+++
(Text.ParserCombinators.ReadPrec.prec 10
(do { GHC.Read.expectP (Text.Read.Lex.Ident "Stencil2");
a1 ← Text.ParserCombinators.ReadPrec.step GHC.Read.readPrec;
a2 ← Text.ParserCombinators.ReadPrec.step GHC.Read.readPrec;
a3 ← Text.ParserCombinators.ReadPrec.step GHC.Read.readPrec;
    .... })
Text.ParserCombinators.ReadPrec.+++
Text.ParserCombinators.ReadPrec.prec 10
(do { GHC.Read.expectP (Text.Read.Lex.Ident "Collect");
a1 ← Text.ParserCombinators.ReadPrec.step GHC.Read.readPrec;
    return (Collect a1 )})))))

When typechecking the code for 'GHC.Read.readPrec'
in a derived instance for 'Read (PreOpenAcc acc aenv a)':
data List a where
   Nil :: List a
   Cons :: a → List a → List a

simply typed ADTs
data List a where
  Nil :: List a
  Cons :: a → List a → List a

simply typed ADTs

type-indexed GADTs

data Vec n a where
  VNil :: Vec Zero a
  VCons :: a → Vec n a → Vec (Succ n) a

head :: List a → a
head Nil = ):

vhead :: Vec (Succ n) a → a
vhead VNil ^_^
type-indexed GADTs

new feature?
Difficulties…
- rapid prototyping
- missing compiler features
- … error messages

type-indexed GADTs
simply typed ADTs

- type-indexed GADTs
new feature

simply typed ADTs

type-indexed GADTs
new feature

remove type invariants

simply typed ADTs

type-indexed GADTs
new feature

remove type invariants

reestablish invariants

simply typed ADTs

type-indexed GADTs
focus of this work

remove type invariants

new feature

reestablish invariants

simply typed ADTs

type-indexed GADTs
#1: do it manually
#1: do it manually

#2: runtime eval

Example in the wild:  https://hackage.haskell.org/package/hakaru
### Evaluation

#### 8. Runtime Performance

Constructing a down-converted implementation of Ghostbuster, we need to import data definitions from, a target host language. Because our prototype implemeting Ghostbuster, we need to import data definitions from, languages that incorporate GADTs. To build a practical tool im-

#### 7. Implementing Ghostbuster for Haskell

As mentioned in Section 2.1, we as arguments to other type constructors, for example.

#### Advanced type system features

As we saw in Section 2.4, there is not a single, clearly defined behaviour. Future work may allow a user to specify how Ghostbuster should traverse under type constructors to continue the erasure and conversion processes.

#### 6.1 Current Limitations

There are many GADTs exist "in the wild" which might benefit from the many GADTs exist "in the wild" which might benefit from the automated up- and down-conversions explored in this work. We conclude our experimental evaluation by testing our prototype tool.

#### 6.2 Implementation

Erased datatypes as type parameters includes type families [7, 20] and type classes [11, 19].

#### 6.3 Erasure

Data types generated by Ghostbuster. Benchmarks were conducted on a machine with two 12-core Xeon E5-2670 CPUs (64-bit, 2.3GHz, 32GB RAM) running GNU/Linux (Ubuntu 14.04 LTS), using GHC.

#### 6.4 Type Erasure

A regular Haskell data type is erasure variant with embedded package.

### Table 1: Benchmarks

<table>
<thead>
<tr>
<th>Contender</th>
<th>SLOC</th>
<th>Tokens</th>
<th>Binary size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ghostbuster</td>
<td>198</td>
<td>1426</td>
<td>1MB</td>
</tr>
<tr>
<td>#1: Manually written</td>
<td>122</td>
<td>1011</td>
<td>1MB</td>
</tr>
<tr>
<td>#2: Runtime eval</td>
<td>78</td>
<td>451</td>
<td>45MB</td>
</tr>
</tbody>
</table>

#### #1: do it manually

#### #2: runtime eval

https://hackage.haskell.org/package/hint

Example in the wild: https://hackage.haskell.org/package/hakaru
We generated large random programs that included all of the we could lean on a we do not model explicitly in our core language. This currently supports Haskell but could be extended to other languages that incorporate GADTs. To build a practical tool im-

The Ghostbuster prototype tool is a source-to-source translator, version 7.10.1 at report the time to convert programs containing that number of terms.

important cases of up- and down- conversion (32GB RAM) running GNU/Linux (Ubuntu 14.04 LTS), using GHC generated by Ghostbuster. Benchmarks were conducted on a

8.1 Runtime Performance

constructors to continue the erasure and conversion processes.

allow a user to specify how Ghostbuster should traverse under type

types, which is not a single, clearly defined behaviour. Future work may

many GADTs exist “in the wild” which might benefit from the

includes type families [7, 20] and type classes [11, 19].

Ghostbuster does not allow datatypes undergoing erasure to be used

port indirectly by allowing them in the “opaque” regions of the

Advanced type system features

results in Haskell’98 datatypes, we add

package,

from our core-language into Haskell using the

for the most part, code generation is a straightforward translation

date certain Haskell features of data definitions such as bang patterns.

require type-indexed

Runtime type representation

are supported.

7. Implementing Ghostbuster for Haskell

in GHC-8.2. In the meantime, we use our own representation of

in the alternatives; (2) the conversion must live in the

Execution Time

This work: Ghostbuster

#1: do it manually

#2: runtime eval

#1: Manually written

#2: Runtime eval

<table>
<thead>
<tr>
<th>Contender</th>
<th>SLOC</th>
<th>Tokens</th>
<th>Binary size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ghostbuster</td>
<td>198</td>
<td>1426</td>
<td>1MB</td>
</tr>
<tr>
<td>#1: Manually written</td>
<td>122</td>
<td>1011</td>
<td>1MB</td>
</tr>
<tr>
<td>#2: Runtime eval</td>
<td>78</td>
<td>451</td>
<td>45MB</td>
</tr>
</tbody>
</table>

Example in the wild:  https://hackage.haskell.org/package/hakaru

ally

yal

https://hackage.haskell.org/package/hint
This section analyzes the performance of the conversion routines for our simple expression language (Section 3.1).

7.1 Current Limitations

Erased datatypes as type parameters

The Ghostbuster prototype tool is a source-to-source translator, which currently supports Haskell but could be extended to other languages. The tool is designed to automatically generate Haskell code from a given input, allowing a user to specify how Ghostbuster should traverse under type classes. However, there is not a single, clearly defined behaviour. Future work may require type-indexed information to allow a user to specify how Ghostbuster should traverse under type classes.

8. Evaluation

7.10.1 at http://hackage.haskell.org/package/criterion build. For the most part, code generation is a straightforward translation when working with type classes and datatypes.

Example in the wild: https://hackage.haskell.org/package/hakaru

<table>
<thead>
<tr>
<th>Contender</th>
<th>SLOC</th>
<th>Tokens</th>
<th>Binary size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ghostbuster</td>
<td>198</td>
<td>1426</td>
<td>1MB</td>
</tr>
<tr>
<td>#1: Manually written</td>
<td>122</td>
<td>1011</td>
<td>1MB</td>
</tr>
<tr>
<td>#2: Runtime eval</td>
<td>78</td>
<td>451</td>
<td>45MB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contender</th>
<th>SLOC</th>
<th>Tokens</th>
<th>Binary size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ghostbuster</td>
<td>198</td>
<td>1426</td>
<td>1MB</td>
</tr>
<tr>
<td>#1: Manually written</td>
<td>122</td>
<td>1011</td>
<td>1MB</td>
</tr>
<tr>
<td>#2: Runtime eval</td>
<td>78</td>
<td>451</td>
<td>45MB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contender</th>
<th>SLOC</th>
<th>Tokens</th>
<th>Binary size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ghostbuster</td>
<td>198</td>
<td>1426</td>
<td>1MB</td>
</tr>
<tr>
<td>#1: Manually written</td>
<td>122</td>
<td>1011</td>
<td>1MB</td>
</tr>
<tr>
<td>#2: Runtime eval</td>
<td>78</td>
<td>451</td>
<td>45MB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contender</th>
<th>SLOC</th>
<th>Tokens</th>
<th>Binary size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ghostbuster</td>
<td>198</td>
<td>1426</td>
<td>1MB</td>
</tr>
<tr>
<td>#1: Manually written</td>
<td>122</td>
<td>1011</td>
<td>1MB</td>
</tr>
<tr>
<td>#2: Runtime eval</td>
<td>78</td>
<td>451</td>
<td>45MB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contender</th>
<th>SLOC</th>
<th>Tokens</th>
<th>Binary size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ghostbuster</td>
<td>198</td>
<td>1426</td>
<td>1MB</td>
</tr>
<tr>
<td>#1: Manually written</td>
<td>122</td>
<td>1011</td>
<td>1MB</td>
</tr>
<tr>
<td>#2: Runtime eval</td>
<td>78</td>
<td>451</td>
<td>45MB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contender</th>
<th>SLOC</th>
<th>Tokens</th>
<th>Binary size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ghostbuster</td>
<td>198</td>
<td>1426</td>
<td>1MB</td>
</tr>
<tr>
<td>#1: Manually written</td>
<td>122</td>
<td>1011</td>
<td>1MB</td>
</tr>
<tr>
<td>#2: Runtime eval</td>
<td>78</td>
<td>451</td>
<td>45MB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contender</th>
<th>SLOC</th>
<th>Tokens</th>
<th>Binary size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ghostbuster</td>
<td>198</td>
<td>1426</td>
<td>1MB</td>
</tr>
<tr>
<td>#1: Manually written</td>
<td>122</td>
<td>1011</td>
<td>1MB</td>
</tr>
<tr>
<td>#2: Runtime eval</td>
<td>78</td>
<td>451</td>
<td>45MB</td>
</tr>
</tbody>
</table>
data List a where
  Nil   :: List a
  Cons :: a → List a → List a
data List a where
  Nil :: List a
  Cons :: a → List a → List a

data Vec n a where
  VNil :: Vec Zero a
  VCons :: a → Vec n a → Vec (Succ n) a
data List a where
  Nil  :: List a
  Cons :: a → List a → List a

( Ornaments, McBride 2010 )
  ( Dagand, ICFP 2016 )

data Vec n a where
  VNil  :: Vec Zero a
  VCons :: a → Vec n a → Vec (Succ n) a
data Vec n a where
  VNil :: Vec Zero a
  VCons :: a → Vec n a → Vec (Succ n) a
{-# Ghostbuster: synthesize n #-}

data Vec n a where
  VNil  :: Vec Zero a
  VCons :: a → Vec n a → Vec (Succ n) a
data Vec' a where
  VNil' :: Vec' a
  VCons' :: a → Vec' a → Vec' a

{-# Ghostbuster: synthesize n #-}
data Vec n a where
  VNil :: Vec Zero a
  VCons :: a → Vec n a → Vec (Succ n) a
instance ... ⇒ Read (Vec n a) where
readsPrec i s =
read simply-typed ADT

instance ... ⇒ Read (Vec n a) where
  readsPrec i s =
  [ (v,r) | (v',r) ← readsPrec i s ]
instance ... ⇒ Read (Vec n a) where

readsPrec i s =
    [ (v,r) | (v',r) ← readsPrec i s,
      let Just v = downVec v' ]
{-# Ghostbuster: synthesize a #-}

data List a where
  Nil  :: List a
  Cons :: a -> List a -> List a
{-# Ghostbuster: synthesize a #-}
data List a where
    Nil    :: List a
    Cons   :: a → List a → List a
{-# Ghostbuster: check a #-}

data List a where
  Nil  ::  List a
  Cons :: a → List a → List a
data List' where
  Nil' :: List'
  Cons' :: ∃ a. TypeRep a → a → List'
         → List'

{-# Ghostbuster: check a #->
data List a where
  Nil  :: List a
  Cons :: a → List a → List a
```haskell
{-# Ghostbuster: check a #} data List a where
  Nil   :: List a
  Cons  :: a -> List a -> List a
```

```haskell
data List' where
  Nil'  :: List'
  Cons' :: ∃ a. TypeRep a -> a -> List'
```

runtime type checks
data List' where
  Nil' :: List'
  Cons' :: ∃ a. TypeRep a → a → List'
           → List'

upList ( ) downList

{-# Ghostbuster: check a #} data List a where
  Nil  :: List a
  Cons :: a → List a → List a
checked vs. synthesised

an information-flow criterion for how erased type information can be recovered
checked vs. synthesised

{-# synthesize n #-}
downVec :: Vec' a
    → Maybe (Vec n a)
checked vs. synthesised

```
{-# synthesize n #-}
downVec :: Vec' a
    → Maybe (Vec n a)
```

output: determined by structure of the datatype
checked vs. synthesised

downVec = openVecS . downVecS
checked vs. synthesized output: determined by structure of the datatype

```
downVec = openVecS . downVecS
```

keep synthesized type existential
checked vs. synt.

keep synthesized type existential

downVec = openVecS \cdot downVecS

data SVec a where
    SVec :: \exists n. Vec n a \to SVec a

downVecS :: Vec' a \to SVec a
checked vs. synthesised output:
determined by structure of the datatype

```
downVec = openVecS . downVecS

data SVec a where
  SVec :: ∃ n. Vec n a → SVec a

downVecS :: Vec' a → SVec a
openVecS :: SVec a → Maybe (Vec n a)
```
checked vs. synthesised output: determined by structure of the datatype

```
---

downVec = openVecS . downVecS

data SVec a where
  SVec :: ∃ n. Vec n a → SVec a

downVecS :: Vec' a → SVec a
openVecS :: SVec a → Maybe (Vec n a)
withVecS :: SVec a → (∀ n. Vec n a → b) → b
---
```
checked vs. synthesised

{-# synchronize n #-}
downVec :: Vec' a → Maybe (Vec n a)

output: determined by structure of the datatype
checked vs. synthesised

```haskell
{-# synthesise n #-}
downVec :: Vec' a
   → Maybe (Vec n a)
```

output: determined by structure of the datatype

```haskell
{-# check a #-}
downList :: List'
   → Maybe (List a)
```
checked vs. synthesised

```haskell
{-# synthesize n #-}
downVec :: Vec' a 
       → Maybe (Vec n a)
```

output: determined by structure of the datatype

input: must check the type of each element

```haskell
{-# check a #-}
downList :: List' 
       → Maybe (List a)
```
In the paper...
In the paper...

```haskell
{-# Ghostbuster: check env, synthesize ans #-}

data Exp env ans where
  Con :: Int -> Exp e Int
  Add :: Exp e Int -> Exp e Int -> Exp e Int
  Var :: Idx e a -> Exp e a
  Abs :: Typ a -> Exp (e, a) b -> Exp e (a -> b)
  App :: Exp e (a -> b) -> Exp e a -> Exp e b

data Idx env t where
  ZeroIdx :: Idx (env, t) t
  SuccIdx :: Idx env t -> Idx (env, s) t
```
In the paper...

![Programs and datatype declarations](image)

---

**Ghostbuster: check env, synthesize ans #—**

```
data Exp env ans where
  Con :: Int → Exp e Int
  Add :: Exp e Int → Exp e Int → Exp e Int
  Var :: Idx e a → Exp e a
  Abs :: Typ a → Exp (e, a) b → Exp e (a → b)
  App :: Exp e (a → b) → Exp e a → Exp e b
```

```
data Idx env t where
  ZeroIdx :: Idx (env, t) t
  SuccIdx :: Idx env t → Idx (env, s) t
```

---

**Figure 3.** The core language manipulated by Ghostbuster
In the paper...

Programs and datatype declarations

\[
\begin{align*}
  \text{prog} &::= \text{dd}_1 \ldots \text{dd}_n; \text{vd}_1 \ldots \text{vd}_m; e \\
  \text{dd} &::= \text{data } T \overline{\tau_1} \overline{\tau_2} \ldots \overline{\tau_m} \text{ where} \\
                 &\quad \text{K :: } \forall \overline{k, c, s, b} \\
                 &\quad \tau_1 \rightarrow \cdots \rightarrow \tau_p \rightarrow T \overline{\tau_k} \overline{\tau_e} \overline{\tau_s} \\
  \text{vd} &::= x :: \sigma; x = e
\end{align*}
\]

Data constructors \(K\)
Type constructors \(T, S\)
Type variables \(a, b, k, c, s\)
Monotypes \(\tau \ ::= a \mid \tau \rightarrow \tau \mid T \overline{\tau}\)
Type Schemes \(\sigma ::= \tau \mid \forall \pi. \tau\)

Term variables \(x, y, z\)
Constraints \(C, D ::= \epsilon \mid \tau \sim \tau \mid C \& C\)
Substitutions \(\phi ::= \emptyset \mid \phi, \{a := \tau\}\)

Terms \(e ::= K \mid x \mid \lambda x : \tau. e \mid e e \mid \text{let } x : \sigma = e \text{ in } e \mid \text{case}\[\tau\] e \mid (\text{typerep } T) x_1 \ldots x_n \rightarrow e \mid \_ \rightarrow e \mid \text{if } e \equiv \tau \text{ then } e \text{ else } e\)

Patterns \(p ::= K \mid x_1 \ldots x_n\)
Type names \(T ::= T \mid \text{ArrowTy} \mid \text{Existential}\)

\[\text{Figure 3. The core language manipulated by Ghostbuster}\]

\[
\begin{align*}
  &\text{C, } \Gamma \vdash \text{typerep } T : \text{TypeRep } a^n \rightarrow \text{TypeRep } (T \overline{\tau^n}) \quad C, \Gamma \vdash e : \text{TypeRep } a_0 \\
  &\quad \frac{C \land (a_0 \sim T \overline{\tau_0}^n), \Gamma \cup \{t_1 : \text{TypeRep } a_1, \ldots, x_n : \text{TypeRep } a_n\} \vdash e' : \tau}{C, \Gamma \vdash \text{typecase}\[\tau\] e \mid (\text{typerep } T) x_1 \ldots x_n \rightarrow e' \mid \_ \rightarrow e'' : \tau} \\
  &\quad \frac{C, \Gamma \vdash e : \text{TypeRep } \tau_1 \quad C, \Gamma \vdash e' : \tau}{C, \Gamma \vdash \text{if } e \geq \tau \text{ then } e' \text{ else } e' : \tau} \\
\end{align*}
\]

\[\text{Figure 4. Typing rules for type representations and operations on them}\]
In the paper...

Theorem 1 (Gradual erasure guarantee). For a given datatype with erasure settings \( \bar{\delta}, \bar{\tau} = \bar{\tau}_1 \bar{\tau}_2 \) and \( \bar{\pi} = \bar{\pi}_1 \bar{\pi}_2 \), then erasure settings \( \bar{\delta} = (\bar{\tau}_1 \bar{\tau}_2) \), \( \bar{\tau} = \bar{\pi}_1 \bar{\pi}_2 \) will also be valid.

Theorem 2 (Reachability of type representations). All searches by bind for a path to \( v \) in \( \phi \) succeed.

Theorem 3. Round-trip. Let \( \text{prog} \) be a program, and let \( \mathbf{T} = \{ (T_1, k_1, c_1, s_1), \ldots, (T_n, k_n, c_n, s_n) \} \) be the set of all datatypes in \( \text{prog} \) that have variable erasures. Let \( \mathbf{D} = \{ D_1, \ldots, D_n \} \) be a set of dictionaries such that \( D_i = (D_i, s_i, c_i) \) contains all needed \( \text{typeReps} \) for the synthesized and checked types of \( T_i \). We then have that if for each \( (T_i, k_i, c_i, s_i) \in \mathbf{D} \) that \( T_i \) passes the ambiguity criteria, then Ghostbuster will generate a new program \( \text{prog}' \) with busted datatypes \( \mathbf{T}' = \{ (T'_1, k_1), \ldots, (T'_n, k_n) \} \), and functions up\( T_i \) and down\( T_i \) such that

\[
\forall e \in \text{prog}. \exists e' \in \text{prog}' \quad \text{and} \quad e' \models (\text{up}(T_i, D_i, e) : T'_i k_i), \text{where } (T'_i, k_i) \in \mathbf{T}'
\]

and

\[
\forall e \in \text{prog}. \exists e'' \in \text{prog}' \quad \text{and} \quad e'' \models (\text{down}(T_i, D_i, e) : T_i k_i)
\]

The full proof including supporting lemmas can be found in the Supplemental Material (Section C). We provide a brief proof-sketch here.
### Package Survey

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total # packages</td>
<td>9026</td>
</tr>
<tr>
<td>Total # source files</td>
<td>94,611</td>
</tr>
<tr>
<td>Total # SLOC</td>
<td>16,183,864</td>
</tr>
<tr>
<td>Total # datatypes using ADT syntax</td>
<td>9261</td>
</tr>
<tr>
<td>Total # datatypes using GADT syntax</td>
<td>18,004</td>
</tr>
<tr>
<td>Total # connected components</td>
<td>15,409</td>
</tr>
<tr>
<td>ADTs with type variable(s)</td>
<td>1341</td>
</tr>
<tr>
<td>GADTs with type variable(s)</td>
<td>11,213</td>
</tr>
<tr>
<td>GADTs with type indexed variable(s)</td>
<td>8773</td>
</tr>
<tr>
<td>Actual search space</td>
<td>185,056,322,576,712</td>
</tr>
<tr>
<td>Explored search space</td>
<td>9,589,356</td>
</tr>
<tr>
<td>Ghostbuster succeeded</td>
<td>2,582,572</td>
</tr>
<tr>
<td>GADTs turned into ADTs</td>
<td>5525</td>
</tr>
<tr>
<td>Ambiguity check failure</td>
<td>5,374,628</td>
</tr>
<tr>
<td>Unimplemented feature in Ghostbuster</td>
<td>1,632,156</td>
</tr>
</tbody>
</table>

**Table 1.** Summary of package survey
Performance

expression language AST (checked + synthesized)
type-indexed $\rightarrow$ simply typed
Performance

expression language AST (checked + synthesized)
simply typed $\rightarrow$ type-indexed
Summary

Ghostbuster is a tool for converting between simply and type-indexed datatypes, in order to incrementalise engineering costs.

Thank you!
Summary

Ghostbuster is a tool for converting
between simply and type-indexed datatypes,
in order to incrementalise engineering costs

Thank you!