Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

The Past, Present and Future of the Programmer-friendly Helium Compiler

Jurriaan Hage

Department of Information and Computing Sciences, Universiteit Utrecht
J.Hage@uu.nl

September 6, 2014

1. Helium on Hackage

What is Helium?

- Original idea by Arjan van IJzendoorn (more credits later)
  - See the Haskell Workshop paper from 2003

What is Helium?

- Original idea by Arjan van IJzendoorn (more credits later)
  - See the Haskell Workshop paper from 2003
- Haskell 98 without class and instance definitions

What is Helium?

- Original idea by Arjan van IJzendoorn (more credits later)
  - See the Haskell Workshop paper from 2003
- Haskell 98 without class and instance definitions
- Particular attention paid to type error diagnosis
  - Serendipity in action
- Used in education in a few places (I know of)

What is Helium?

- Original idea by Arjan van IJzendoorn (more credits later)
  - See the Haskell Workshop paper from 2003
- Haskell 98 without class and instance definitions
- Particular attention paid to type error diagnosis
  - Serendipity in action
- Used in education in a few places (I know of)
- Support for ~~tailorable~~ domain-specific type error diagnosis

What is Helium?

- Original idea by Arjan van IJzendoorn (more credits later)
  - See the Haskell Workshop paper from 2003
- Haskell 98 without class and instance definitions
- Particular attention paid to type error diagnosis
  - Serendipity in action
- Used in education in a few places (I know of)
- Support for ~~tailorable~~ domain-specific type error diagnosis
- Maintained mostly by Bastiaan Heeren and myself
- Used by the AskElle system (Gerdes, Heeren, Jeuring,...)

Getting Helium 1.8

cabal install helium

-- installs helium, Top, and lvmlib

cabal install lvmrun

-- installs the runtime for running lvm files

texthint ...

runhelium ...

- The novice-friendly Helium compiler is now available from Hackage

Getting Helium 1.8

cabal install helium

-- installs helium, Top, and lvmlib

cabal install lvmrun

-- installs the runtime for running lvm files

texthint ...

runhelium ...

- The novice-friendly Helium compiler is now available from Hackage
- While you have that compiling, I'll get on with the talk...

What do you get?

- A compiler targeting novice programmer, aiming at better error messaging
- "Helium as a library"
- Haskell 98 minus a "few details"
  - no $n + k$ patterns
  - no class and instance definitions
  - simpler offside rule, simpler module system
- Built-in logging facility for storing student compiles
- Experiment with various type inference algorithms
- Supports domain-specific type error diagnosis
  - see Heeren, Hage, Swierstra ICFP 2003, and later in this talk

What do you get?

- A compiler targeting novice programmer, aiming at better error messaging
- "Helium as a library"
- Haskell 98 minus a "few details"
  - no $n + k$ patterns
  - no class and instance definitions
  - simpler offside rule, simpler module system
- Built-in logging facility for storing student compiles
- Experiment with various type inference algorithms
- Supports domain-specific type error diagnosis
  - see Heeren, Hage, Swierstra ICFP 2003, and later in this talk
- Motivated GHC to ask Simon for better error diagnosis
  - FP lecture notes

To overload or not to overload?

- Two language levels
  - `--no-overloading`: no type classes altogether
  - `--overloading`: Eq, Ord, $Show$, Num, $Enum$
  - with built-in instances, and deriving for Eq and $Show$
- Jeroen Weijers built parsing and type checking support, but code generation is still missing
- Type classes directives from PADL '05 paper not supported by Helium, the constraint solver (TOP) does.

Limitations and extensions?

- Supported language not enough for learning to program Haskell in our own FP course
- Speed comparable or better than Hugs
- But no code optimisations have been implemented, so I expect we are way behind GHC
  - For an educational tool this should not be a problem
- There are some bugs in there (see the website)
  - Hardy, but not debilitating
- Extensions too: polymorphic kinds have been in there for a loooong time

Getting Helium to Hackage

- Helium needs its own libraries (two variants)
- Cabal supports finding the libraries in a platform-independent way

Getting Helium to Hackage

- Helium needs its own libraries (two variants)
- Cabal supports finding the libraries in a platform-independent way
- Still, platform independence continues to be an effort consuming pain

Getting Helium to Hackage

- Helium needs its own libraries (two variants)
- Cabal supports finding the libraries in a platform-independent way
- Still, platform independence continues to be an effort consuming pain
- We used Cabal's Make build-type for building Daan Leijen's LVM interpreter (pure C++)
- Make build-type is very undocumented, and not very robust, but in the end it worked
  - Unsurprise: I am not aware of any other package on Hackage that has the same build-type
- Rolling out new versions of Helium is now much easier

Credits

- Main programmers of the original Helium:
  - Arjan van IJzendoorn: originator, implementer
  - Bastiaan Heeren: main implementer, type system research
  - Daan Leijen: Helium uses his LVM
  - Jurriaan Hage: type system research, integration, logging facility
  - Arie Middelkoop: Hint Java environment
- Many others have helped at some point: Rijk-Jan van Haaften, Arie Middelkoop, Arjan Oosting, Jeroen Weijers, Jurriën Stutterheim, Jeroen Fokker, Andres Löh, Arthur Baars, Remco Burema, Atze Dijkstra, Maarten van Gompel, Doaitse Swierstra, Martijn Lammerts, Martijn Schrage, Alejandro Serrano Mena, Alex Gerdes, and Stefan Holdermans

2. Improving Type Error Diagnosis

Error diagnosing heuristics in Helium

If an inconsistency is detected, heuristics help to choose which location is reported as type incorrect.

- Examples:
  - minimal number of type errors
  - count occurrences of clashing type constants ($3 \times \mathit{Int}$ versus $1 \times \mathit{Bool}$)
  - reporting an expression as type incorrect is preferred over reporting a pattern
  - wrong literal constant (4 versus 4.0)
  - not enough arguments are supplied for a function application
  - permute the elements of a tuple,
  - or arguments to a function
- All implemented in Helium
- NB. heuristics can also point in the wrong direction

Domain-unaware type error diagnosis


```
ERROR "BigTypeError.hs":1 - Type error in application
*** Expression     : sem_Expr_Lam <$ pKey "\\" <*> pFoldr1 (sem_LamIds_Cons,sem_
LamIds_Nil) pVarid <*> pKey "->"
*** Term           : sem_Expr_Lam <$ pKey "\\" <*> pFoldr1 (sem_LamIds_Cons,sem_
LamIds_Nil) pVarid
*** Type           : [Token] -> [((Type -> Int -> [([Char],(Type,Int,Int))] -> I
nt -> Int -> [(Int,(Bool,Int))] -> (PP_Doc,Type,a,b,[c] -> [Level],[S] -> [S]))
-> Type -> d -> [([Char],(Type,Int,Int))] -> Int -> Int -> e -> (PP_Doc,Type,a,b
,f -> f,[S] -> [S]),[Token])]
*** Does not match : [Token] -> [([Char] -> Type -> d -> [([Char],(Type,Int,Int)
)] -> Int -> Int -> e -> (PP_Doc,Type,a,b,f -> f,[S] -> [S]),[Token])]
```

from the innocuous lambda expression parser:

$$
\begin{aligned}
&pExpr = pAndPrioExpr \\
&\quad <|> sem\_Expr\_Lam \\
&\qquad <\$\ pKey\ \texttt{"\textbackslash\textbackslash"} \\
&\qquad <*> pFoldr1\ (sem\_LamIds\_Cons, sem\_LamIds\_Nil)\ pVarid \\
&\qquad <*> pKey\ \texttt{"->"} \\
&\qquad <*> pExpr
\end{aligned}
$$

A closer look at the message


```
ERROR "BigTypeError.hs":1 - Type error in application
*** Expression     : sem_Expr_Lam <$ pKey "\\" <*> pFoldr1 (sem_LamIds_Cons,sem_
LamIds_Nil) pVarid <*> pKey "->"
*** Term           : sem_Expr_Lam <$ pKey "\\" <*> pFoldr1 (sem_LamIds_Cons,sem_
LamIds_Nil) pVarid
*** Type           : [Token] -> [((Type -> Int -> [([Char],(Type,Int,Int))] -> I
nt -> Int -> [(Int,(Bool,Int))] -> (PP_Doc,Type,a,b,[c] -> [Level],[S] -> [S]))
-> Type -> d -> [([Char],(Type,Int,Int))] -> Int -> Int -> e -> (PP_Doc,Type,a,b
,f -> f,[S] -> [S]),[Token])]
*** Does not match : [Token] -> [([Char] -> Type -> d -> [([Char],(Type,Int,Int)
)] -> Int -> Int -> e -> (PP_Doc,Type,a,b,f -> f,[S] -> [S]),[Token])]
```

- Large and complicated
- Where did they hide the mismatch?
- No mention of “parsers” in the error message
- Common mistake, and easy to fix

The solution in a nutshell

1. Bring the type inference mechanism under control
   - by phrasing the type inference process as a constraint solving problem
2. Provide hooks in the compiler's type inference process to change the process for certain classes of expressions
   - specialize type error messages for a particular domain
   - control the order in which constraints are solved
   - drive heuristics that suggest fixes for often-made mistakes

The solution in a nutshell

1. Bring the type inference mechanism under control
   - by phrasing the type inference process as a constraint solving problem
2. Provide hooks in the compiler's type inference process to change the process for certain classes of expressions
   - specialize type error messages for a particular domain
   - control the order in which constraints are solved
   - drive heuristics that suggest fixes for often-made mistakes
- Changing the type system is forbidden!
  - Only the order of solving, and the provided messages can be changed

How is this organised?

- For a given source module Abc.hs, a DSL designer may supply a file Abc.type containing the directives
- The directives are automatically used when the module is imported
- The compiler will adapt the type error mechanism based on these type inference directives.
- The directives themselves are also a(n external) DSL!

Specialized type error messages


```
  x :: t1;   y :: t2;
--------------------
    x <$> y :: t3;

t1 == a1 -> b1      : left operand is not a function
t2 == Parser s1 a2  : right operand is not a parser
t3 == Parser s2 b2  : result type is not a parser
s1 == s2 : parser has an incorrect symbol type
a1 == a2 : function cannot be applied to parser's result
b1 == b2 : parser has an incorrect result type
```

- Supply an error message for each type constraint. This message is reported if the corresponding constraint cannot be satisfied.

Example

$$test :: Parser\ Char\ String$$
$$test = map\ toUpper <\$> \texttt{"hello, world!"}$$

This results in the following type error message:

```
Type error: right operand is not a parser
```

Example

$$test :: Parser\ Char\ String$$
$$test = map\ toUpper\ <\$>\ \texttt{"hello, world!"}$$

This results in the following type error message:

```
Type error: right operand is not a parser
```

Important context specific information is missing, for instance:

- Inferred types for (sub-)expressions, and intermediate type variables
- Pretty printed expressions from the program
- Position and range information

Error message attributes

A second attempt:

```
  x :: t1;   y :: t2;
---------------------
    x <$> y :: t3;
...
t2 == Parser s1 a2 :
 @expr.pos@: The right operand of <$> should be a
  expression          : @expr.pp@                parser
  right operand       : @y.pp@
    type              : @t2@
    does not match    : Parser @s1@ @a2@
...
```

Example

$test :: Parser\ Char\ String$
$test = map\ toUpper$ <$> "hello, world!"

This results in the following type error message (including the inserted error message attributes):

```
(2,21): The right operand of <$> should be a parser
 expression          : map toUpper <$> "hello, world!"
 right operand       : "hello, world!"
   type              : String
   does not match    : Parser Char String
```

Soundness and completeness

The soundness of a specialized type rule with respect to the default type rules is examined at compile time.

- Because a mistake is easily made
- Invalid type rules are rejected when a Haskell file is compiled
- Type safety can still be guaranteed at run-time
- The type rule may not be too restrictive, so we are also complete
  - This restriction may be dropped

Another directive: siblings

- Certain combinators are known to be easily confused:
  - cons $(:)$ and append $(+\!\!+)$
  - $<\$>$ and $<\$$
  - $(.)$ and $(+\!\!+)$ (PHP programmers)
  - $(+)$ and $(+\!\!+)$ (Java programmers)
- These combinations can be listed among the specialized type rules.

```
siblings   <$> , <$
siblings   ++ , +, .
```

- The siblings heuristic will try a sibling if an expression with such an operator fails to type check.

Back to the example

$$\begin{aligned}
&\mathbf{data}\ Expr = Lambda\ [String]\ Expr\\
&pExpr\\
&\quad = pAndPrioExpr\\
&<|>\ Lambda\ <\$\ pKey\ \texttt{"\textbackslash\textbackslash"}\\
&\qquad\qquad <\!\!*\!\!>\ many\ pVarid\\
&\qquad\qquad <\!\!*\ \ pKey\ \texttt{"->"}\\
&\qquad\qquad <\!\!*\ \ pExpr
\end{aligned}$$

Extremely concise:

```
(11,13): Type error in the operator <*
  probable fix: use <*> instead
```

Permuting function arguments example

$$test :: Parser\ Char\ String$$
$$test = option\ ""\ (token\ \texttt{"hello!"})$$

```
(2,8): Type error in application
 expression        : option "" (token "hello!")
 term              : option
   type            : Parser a b -> b -> Parser a b
   does not match  : String -> Parser Char String -> c
 probable fix      : flip the arguments
```

Helium tries not to unfold type synonyms!

3. To wrap up

Tasks for the near future

- What to do with Hint (Java based IDE for Helium)?
- What about the robust Java server for receiving logged programs?
- Fixings those pesky bugs
- In the next release, texthint will be a bit easier to use

The Helium dilemma

- Currently, Helium is too small a subset for our own FP
- SRC: Alejandro is working on extending domain-specific type error diagnosis to, say, Haskell 2010.
- But the current aim is to first build that into the Utrecht Haskell Compiler
  - Allowing us to concentrate on the type system, not on parsing, code generation
- On the other hand, UHC is not intended for student use

The Helium dilemma

- Currently, Helium is too small a subset for our own FP
- SRC: Alejandro is working on extending domain-specific type error diagnosis to, say, Haskell 2010.
- But the current aim is to first build that into the Utrecht Haskell Compiler
  - Allowing us to concentrate on the type system, not on parsing, code generation
- On the other hand, UHC is not intended for student use
- If successful, GHC should get its turn as well

If I have not discouraged you enough... §3

- Helium is open source
- if you want to contribute, I'll need to make that possible
- Mail me at J.Hage@uu.nl
- For now, also use that address for reporting any bugs you may find

If I have not discouraged you enough... §3

- Helium is open source
- if you want to contribute, I'll need to make that possible
- Mail me at J.Hage@uu.nl
- For now, also use that address for reporting any bugs you may find
- Reassuring to know: Helium is on Hackage
- And almost impossible to freeze

Thank you for your attention.

Have fun playing with Helium.

`www.cs.uu.nl/wiki/bin/view/Helium`