Ihs2TEX

Andres Löh

Dutch HUG meeting – October 12, 2009
What \texttt{lhs2\LaTeX} is . . .

A preprocessor.

- Input: a \LaTeX\ document containing directives and Haskell-like code.
- Output: a \LaTeX\ document where the code is formatted as \LaTeX\ as well; or Haskell code that can be run.
What \texttt{lhs2T\TeX} is . . .

A preprocessor.

- **Input:** a \LaTeX{} document containing directives and Haskell-like code.
- **Output:** a \LaTeX{} document where the code is formatted as \LaTeX{} as well; or Haskell code that can be run.

Useful for:

- \LaTeX{} documents containing Haskell code – papers, documentation, presentations, . . .
- \LaTeX{} documents containing other kinds of aligned code
- many things you might want a Haskell preprocessor for
- managing different versions of a document
What lhs2TeX is not . . .

- The conversion is *not* fully automatic.
- You have lots of freedom, but you have to make some choices.
Hello world

Input
\documentclass{article}
%include polycode.fmt
\begin{document}

> main = putStrLn "Hello world"

\end{document}

Output
main = putStrLn "Hello world"
History

- Created by Ralf Hinze in 1997. Most of the functionality is due to Ralf.
- I picked up development in 2002. New features:
  - better code alignment,
  - using \texttt{lhs2TEX} as a preprocessor also to generate code,
  - improved possibilities of calling \texttt{GHC} from within a document.
Inline code

Inline code is surrounded by vertical bars.

**Input**

The function \( |\text{map}| \) takes two arguments, a function \( |f :: a \to b| \) and a value \( |x| \) of type \( |a| \).

**Output**

The function \( \text{map} \) takes two arguments, a function \( f :: a \to b \) and a value \( x \) of type \( a \).
Vertically bars occurring in inline code have to be escaped.

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>The function ( | \text{or} | ) can be defined using ( | \text{foldr} | ), namely as ( | \text{foldr} (,</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>The function \textit{or} can be defined using \textit{foldr}, namely as \textit{foldr} ((\lor)) \textit{True}.</td>
</tr>
</tbody>
</table>

As can be seen, some operators are by default formatted as symbols.
The parser is very liberal. It only approximates the Haskell syntax. Generally, Haskell constructs should be typeset nicely.

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>let x = 2 in x * x</td>
</tr>
<tr>
<td>case x of Foo -&gt; Bar</td>
</tr>
<tr>
<td>[ x * x</td>
</tr>
<tr>
<td>( x \rightarrow x )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>let ( x = 2 ) in ( x \times x )</td>
</tr>
<tr>
<td>case ( x ) of Foo ( \rightarrow ) Bar</td>
</tr>
<tr>
<td>[ ( x \times x )</td>
</tr>
<tr>
<td>( \lambda x \rightarrow x \times x )</td>
</tr>
</tbody>
</table>
Much as inline Haskell, we can also produce verbatim code by surrounding it in @s. Again, escaping other @s is necessary.

**Input**

Typing @foo@ yields |foo|. Here’s an escaped @@.

**Output**

Typing foo yields foo. Here’s an escaped @.
Directives are lines starting with a % immediately followed by a recognized \texttt{lhs2TeX} command. The directive \texttt{\%format} can be used to change the appearance of tokens.

**Input**

\%
format True = "\top "
\%
format foldr = "\{\color{blue}\textbf{foldr}\} "
|foldr (|||) True|

**Output**

\texttt{foldr} (\texttt{\lor}) \top
Formatting directives can also be used to undo predefined formattings. The default formatting of variables and constructors makes use of \Varid{not} and \Conid{not}, respectively.

**Input**

|not x|

\%format not = "\Varid{not}"  
|not x|  
\let\Varid\mathbf  
|not x|

**Output**

¬ x, \textit{not} x, \textbf{not} x
Implicit formatting

For indices there are special cases where no right hand side has to be given. The directive itself is still required.

**Input**

|a1|, |a_1|, |a_not|

%format a1
%format a_1
%format a_not
|a1|, |a_1|, |a_not|

**Output**

a1, a_1, a_not
a_1, a_1, a_not
Parameterized formatting

Input

%format <> = "\diamond 
%format Instr x = "\{\texttt Conid\texttt x \}"
%format eval x = "\llbracket \texttt x \rrbracket "

> eval (Add x y) = eval x <> eval y <> [Instr ADD]

Output

\( [(Add \times y)] = [x] \diamond [y] \diamond [ADD] \)

Too many parentheses!
Parameterized formatting and parentheses

Input

```plaintext
%format eval (x) = "\llbracket " x "\rrbracket "

> (eval (Add x y))

%format (eval (x)) = "\llbracket " x "\rrbracket "

> (eval (Add x y))
```

Output

```plaintext
([Add x y])
[[Add x y]]
```
Blocks of code can be typeset using a code-environment or by prefixing every line with a `>`:

**Input**

This is a `let` expression:

```
> let x = 2
> in x * x
```

**Output**

This is a `let` expression:

```
let x = 2
in x * x
```
Blocks of code – contd.

Input

This is a \texttt{|let|} expression:
\begin{verbatim}
let x = 2
in x * x
\end{verbatim}

Output

This is a \textbf{let} expression:

\begin{verbatim}
let x = 2
in x * x
\end{verbatim}
Code starting with `<` or in a `spec`-environment is also typeset – for code that should be included in the output, but not run.

**Input**

This is a `let` expression:

\begin{spec}
let x = 2
in x * x
\end{spec}

**Output**

This is a **let** expression:

```plaintext
let x = 2
in x * x
```
Comments are typeset as text. Use `<` or `spec` for larger blocks of commented code that should be shown.

**Input**

> 0 :: Num a => a -- not of type `Int`, but overloaded

**Output**

```
0 :: Num a => a -- not of type `Int`, but overloaded
```
Alignment is lhs2TEX's strong point: a token that is prefixed by two or more spaces is aligned with other tokens on the same column.

**Input**

```
> map f [] = []
> map f (x:xs) = f x : map f xs
```

**Output**

```
map f [] = []
map f (x : xs) = f x : map f xs
```
Indentation is with respect to aligned columns.

**Input**

```haskell
%format ... = "\dots "

> instance (Ord a) => Ord [a] where
> ...
> compare (x:xs) (y:ys) = case compare x y of
>   EQ -> compare xs ys
>   other -> other
```

**Output**

```
instance (Ord a) ⇒ Ord [a] where
...
  compare (x : xs) (y : ys) = case compare x y of
    EQ    → compare xs ys
    other → other
```
Alignment – contd.

Alignment does not have to affect subsequent lines.

**Input**

```haskell
> consTree a (Deep s (Two b c) m sf) =
>   Deep (size a + s) (Three a b c) m sf
> consTree a (Deep s (One b) m sf) =
>   Deep (size a + s) (Two a b) m sf
```

**Output**

```haskell
consTree a (Deep s (Two b c) m sf) =
   Deep (size a + s) (Three a b c) m sf
consTree a (Deep s (One b) m sf) =
   Deep (size a + s) (Two a b) m sf
```

Watch out that code is not aligned by accident!
Alignment is computed by \LaTeX, using the polytable package that was written specifically for lhs2\TeX.

### Input

```
%format i = "\Varid{iiiiiiiiiiiiii}" \\
> xxx  yyy  zzz \\
> aaaaa  bbbbb \\
> i  jjjjjjjjj \\
> c  dddddd
```

### Output

```
xxx     yyy  zzz
aaaaa  bbbbb
iiiiiiiiiiiiii  jjjjjjjjjj
   c       dddddd
```
Reusing alignment

Alignment information can be shared for multiple code blocks.

Input

\texttt{\textbackslash savecolumns}

\begin{verbatim}
> eval (Const n) = n
> eval (Neg x) = - (eval x)
\end{verbatim}

And now addition: \texttt{\textbackslash restorecolumns}

\begin{verbatim}
> eval (Add x y) = eval x + eval y
\end{verbatim}

Output

\begin{verbatim}
[Const n] = n
[Neg x] = - [x]
\end{verbatim}

And now addition:

\begin{verbatim}
[Add x y] = [x] + [y]
\end{verbatim}
Including files

Using an \%include directive, a file can be included. This is used for .fmt files that contain lhs2\TeX\ libraries, but can be used for parts of the document instead of \LaTeX\ commands.

\%include polycode.fmt

There are a number of useful files shipped with lhs2\TeX\.
Libraries

- polycode.fmt – standard library
- colorcode.fmt – some code styles using colored backgrounds
- greek.fmt – format greek identifiers
- forall.fmt – universal quantifier magic
- spacing.fmt – spacing hacks
Using the lhs2TEX standard library, you can easily adapt the look and feel of lhs2TEX.

**Input**

\renewcommand\hscodestyle{\small\rmfamily}

\begin{verbatim}
> foldr \texttt{op} e [] = []
> foldr \texttt{op} e (x:xs) = x \texttt{'}op\texttt{'} foldr \texttt{op} e xs
\end{verbatim}

**Output**

foldr \texttt{op} e [] = []
foldr \texttt{op} e (x:xs) = x \texttt{'}op\texttt{'} foldr \texttt{op} e xs
Input

\texttt{\textbackslash framedhs}

\[ > \text{foldr } \text{op } e \ [\] \ = \ [\] \]
\[ > \text{foldr } \text{op } e \ (x:xs) \ = \ x \ 'op' \ \text{foldr } \text{op } e \ xs \]

Output

\texttt{foldr op e \ [\] \ = \ [\]}
\texttt{foldr op e \ (x : xs) \ = \ x \ 'op' \ foldr op e \ xs}
Sometimes you want to have code as part of the module and still show it inline.

**Input**

We therefore define

\[
\text{\textbackslash inlinehs}
\]

\[
> \text{mapM } f = \text{sequence } \cdot \text{map } f
\]

and are done.

**Output**

We therefore define \( \text{mapM } f = \text{sequence } \circ \text{map } f \) and are done.
Libraries – Greek identifiers

Input

```
%include greek.fmt

> gamma = alpha + beta
```

Output

$$\gamma = \alpha + \beta$$
If you use Haskell code with explicit quantifiers, you probably want to include `forall.fmt`:

**Input**

```haskell
%include forall.fmt

> mapM :: forall m. (Monad m) => (a -> m b) -> [a] -> m [b]
> mapM f = sequence . map f
```

**Output**

```haskell
mapM :: ∀m.(Monad m) ⇒ (a → m b) → [a] → m [b]

mapM f = sequence ∘ map f
```

Note the different formatting of the periods.
There are directives `%if`, `%else`, `%elif` and `%endif` that can be used to process parts of the document conditionally.

- Documentation, paper, presentation from the same sources.
- Process differently depending on mode.

Using `%let` or command line flags, we can set variables to boolean or integer values.
You can annotate your code and still run it.
By using formatting directives and conditionals, you can typecheck your documents.

Lhs2TEX is not limited to displaying Haskell code. Using formatting directives, you can use it to display a wide range of languages.
How to get it

- Current version is 1.14.
- Available from Hackage (i.e., cabal install lhs2tex).
- Version 1.15 should appear soon (mainly interesting for Windows users).
- Let me know if you’re doing something cool with lhs2\TeX. 