HaTeX: Generación de código LaTeX

Programación declarativa

Universidad de Málaga

24 de mayo de 2.011

¿Qué es LaTeX?

LaTeX es un lenguaje que nos permite programar documentos (e incluso imágenes) con una alta calidad y con características que le hacen apropiado para la redacción de textos técnicos y científicos, aunque puede ser utilizado para casi cualquier objetivo.

`http://www.latex-project.org/`

¿Qué es LaTeX?

LaTeX es un lenguaje que nos permite programar documentos (e incluso imágenes) con una alta calidad y con características que le hacen apropiado para la redacción de textos técnicos y científicos, aunque puede ser utilizado para casi cualquier objetivo.

`http://www.latex-project.org/`

La motivación de este proyecto fue acercar la programación de documentos LaTeX a Haskell, construyendo una librería que los fabricase.

Nuestros objetivos

Los objetivos en mente para diseñar esta librería son:

1. Diseñar una estructura tipificada de la sintaxis de LaTeX.
2. Producir, a partir de datos de este tipo, una salida correspondiente a código LaTeX.
3. Conseguir una librería con una interfaz lo más semejante posible a la original, mejorándola en todo lo posible.

Esto permitirá una integración cómoda de programación en LaTeX dentro de un guión escrito en Haskell.

Describiendo la sintaxis

Estructuración en tipos

Una sintaxis viene dada por una serie de reglas de construcción.

Describiendo la sintaxis

Estructuración en tipos

Una sintaxis viene dada por una serie de reglas de construcción.

Por ejemplo, podríamos decir que la sintaxis de una suma viene dada por el siguiente par de reglas:

Describiendo la sintaxis

Estructuración en tipos

Una sintaxis viene dada por una serie de reglas de construcción.

Por ejemplo, podríamos decir que la sintaxis de una suma viene dada por el siguiente par de reglas:

1. Un número es una suma.

Describiendo la sintaxis

Estructuración en tipos

Una sintaxis viene dada por una serie de reglas de construcción.

Por ejemplo, podríamos decir que la sintaxis de una suma viene dada por el siguiente par de reglas:

1. Un número es una suma.
2. Una suma, seguida de un símbolo $+$, seguido de otra suma, es, a su vez, una suma.

Describiendo la sintaxis

Estructuración en tipos

Una sintaxis viene dada por una serie de reglas de construcción.

Por ejemplo, podríamos decir que la sintaxis de una suma viene dada por el siguiente par de reglas:

1. Un número es una suma.
2. Una suma, seguida de un símbolo +, seguido de otra suma, es, a su vez, una suma.

Luego un tipo coherente a estas reglas que represente una suma podría ser:

Tipo Suma

```
data Suma = Numero Int | Sumandos Suma Suma
```

Describiendo la sintaxis

Sintaxis de LaTeX (I)

Pasemos ahora a definir una sintaxis más compleja, de interés para nuestro cometido: la sintaxis de LaTeX.

Describiendo la sintaxis

Sintaxis de LaTeX (I)

Pasemos ahora a definir una sintaxis más compleja, de interés para nuestro cometido: la sintaxis de LaTeX.

Por regla general, un comando en LaTeX posee el siguiente formato:

```
\nombre[Argumento Opcional]{Argumento Obligatorio}
```

Describiendo la sintaxis

Sintaxis de LaTeX (I)

Pasemos ahora a definir una sintaxis más compleja, de interés para nuestro cometido: la sintaxis de LaTeX.

Por regla general, un comando en LaTeX posee el siguiente formato:

```
\nombre[Argumento Opcional]{Argumento Obligatorio}
```

Es decir, un comando válido está formado por:

1. Un símbolo \.

Describiendo la sintaxis

Sintaxis de LaTeX (I)

Pasemos ahora a definir una sintaxis más compleja, de interés para nuestro cometido: la sintaxis de LaTeX.

Por regla general, un comando en LaTeX posee el siguiente formato:

```
\nombre[Argumento Opcional]{Argumento Obligatorio}
```

Es decir, un comando válido está formado por:

1. Un símbolo \.
2. El nombre del comando.

Describiendo la sintaxis

Sintaxis de LaTeX (I)

Pasemos ahora a definir una sintaxis más compleja, de interés para nuestro cometido: la sintaxis de LaTeX.

Por regla general, un comando en LaTeX posee el siguiente formato:

```
\nombre[Argumento Opcional]{Argumento Obligatorio}
```

Es decir, un comando válido está formado por:

1. Un símbolo \.
2. El nombre del comando.
3. Una serie de argumentos, opcionales o necesarios. Argumentos opcionales estarán encerrados entre corchetes y los necesarios entre llaves.

Describiendo la sintaxis

Sintaxis de LaTeX (II)

A la hora de representar esta sintaxis *concreta* con una sintaxis *abstracta*, los detalles como el símbolo \ o los corchetes (o llaves) que encierran a los argumentos quedan camuflados, reemplazados por los constructores de tipo (como lo hacía el constructor `Sumandos` en el tipo `Suma`).

Describiendo la sintaxis

Sintaxis de LaTeX (II)

A la hora de representar esta sintaxis *concreta* con una sintaxis *abstracta*, los detalles como el símbolo \ o los corchetes (o llaves) que encierran a los argumentos quedan camuflados, reemplazados por los constructores de tipo (como lo hacía el constructor `Sumandos` en el tipo `Suma`).

Así, el constructor apropiado para un comando queda descrito del siguiente modo:

Constructor `TeXComm`

```
TeXComm String [TeXArg]
```

donde `TeXArg` corresponde al tipo que representa un argumento en LaTeX.

Describiendo la sintaxis

Tipo LaTeX

La descripción completa de la sintaxis de LaTeX queda descrita con la siguiente declaración de tipo:

Tipo LaTeX

```
data LaTeX =
      TeXRaw Text
    | TeXComm String [TeXArg]
    | TeXCommS String
    | TeXEnv String [TeXArg] LaTeX
    | TeXMath LaTeX
    | TeXNewLine Bool
    | TeXOp String LaTeX LaTeX
    | TeXSeq LaTeX LaTeX
    | TeXEmpty
      deriving Show
```

Describiendo la sintaxis

Tipo TeXArg

La sintaxis de los argumentos puede describirse:

Tipo TeXArg

```
data TeXArg =
      OptArg LaTeX
    | FixArg LaTeX
    | MOptArg [LaTeX]
    | SymArg LaTeX
    | MSymArg [LaTeX]
      deriving Show
```

Quedando así definida la sintaxis total del código LaTeX mediante estructuras de tipo. Nótese la recursividad, tanto en un mismo tipo como entre ellos.

Creando la interfaz

Para que el usuario de la librería pueda crear código LaTeX de un modo similar al código original, se ha exportado el tipo `LaTeX` como tipo abstracto de datos, y se proporcionan una serie de funciones que representan los comandos y entornos existentes en LaTeX. Por ejemplo:

```
textbf ::  LaTeX -> LaTeX
textbf l = TeXComm ``textbf'' [FixArg l]

document ::  LaTeX -> LaTeX
document l = TeXEnv ``document''[]
```

Cadenas sobrecargadas

Motivación

Para crear una expresión `LaTeX` con texto plano a partir de un `String`, hemos de utilizar la función `fromString` seguida del constructor `TeXRaw`.

Cadenas sobrecargadas

Motivación

Para crear una expresión `LaTeX` con texto plano a partir de un `String`, hemos de utilizar la función `fromString` seguida del constructor `TeXRaw`.

Ésta será una operación frecuente. Sin embargo, lo ideal sería poder introducir texto sin necesidad de acudir a ella.

Cadenas sobrecargadas

Motivación

Para crear una expresión LaTeX con texto plano a partir de un `String`, hemos de utilizar la función `fromString` seguida del constructor `TeXRaw`.

Ésta será una operación frecuente. Sin embargo, lo ideal sería poder introducir texto sin necesidad de acudir a ella.

La función `fromString` está *sobrecargada*, puesto que es miembro de la clase `IsString`.

Clase `IsString`

```
class IsString a where
 fromString ::  String -> a
```

Cadenas sobrecargadas

Resolución

Una vez hemos hecho `LaTeX` instancia de la clase `IsString`, podemos utilizar el método `fromString` para crear a partir de cadenas de caracteres datos del tipo `LaTeX`.

Cadenas sobrecargadas

Resolución

Una vez hemos hecho `LaTeX` instancia de la clase `IsString`, podemos utilizar el método `fromString` para crear a partir de cadenas de caracteres datos del tipo `LaTeX`.

Para mayor comodidad, existe la extensión del lenguaje *Overloaded Strings* (o *Cadenas Sobrecargadas*), que sustituye toda aparición de una cadena en el guión de nuestro programa por una aplicación de la función `fromString` sobre la misma cadena. Por ejemplo:

```
``Texto aquí.'' = fromString ``Texto aquí.''
```

Generando la salida

La función `render`

Una vez hemos creado un dato que representa el fichero que queremos exportar, necesitamos una función que lo transcriba a un fichero real.

Generando la salida

La función `render`

Una vez hemos creado un dato que representa el fichero que queremos exportar, necesitamos una función que lo transcriba a un fichero real.

Ésta función estará definida por patrones e irá, caso por caso, transcribiendo a sintaxis concreta de LaTeX cada una de las expresiones.

Generando la salida

La función `render`

Una vez hemos creado un dato que representa el fichero que queremos exportar, necesitamos una función que lo transcriba a un fichero real.

Ésta función estará definida por patrones e irá, caso por caso, transcribiendo a sintaxis concreta de LaTeX cada una de las expresiones.

Éste es el paso contrario al que realizamos cuando describimos la sintaxis.

Generando la salida

La función `render`

Una vez hemos creado un dato que representa el fichero que queremos exportar, necesitamos una función que lo transcriba a un fichero real.

Ésta función estará definida por patrones e irá, caso por caso, transcribiendo a sintaxis concreta de LaTeX cada una de las expresiones.

Éste es el paso contrario al que realizamos cuando describimos la sintaxis.

```
render ::  LaTeX -> Text
```

Generando la salida

Exportando a un fichero

Ya estamos preparados para exportar el texto a un fichero. Utilizando la función `writeFile` para datos de tipo `Text`, escribiremos en un fichero el resultado, dispuesto a ser compilado con nuestro compilador TEX usual.

Haciendo uso de una mónada

Mejorando la interfaz

Haciendo uso de la mónada escritora y de la estructura de monoide de `LaTeX`, podemos mejorar la interfaz gracias a la notación **do**.

Haciendo uso de una mónada

Mejorando la interfaz

Haciendo uso de la mónada escritora y de la estructura de monoide de `LaTeX`, podemos mejorar la interfaz gracias a la notación **do**.

Sin entrar en detalle, podemos ver un ejemplo de definición de un preámbulo utilizando la notación **do**.

Haciendo uso de una mónada

Mejorando la interfaz

Haciendo uso de la mónada escritora y de la estructura de monoide de `LaTeX`, podemos mejorar la interfaz gracias a la notación **do**.

Sin entrar en detalle, podemos ver un ejemplo de definición de un preámbulo utilizando la notación **do**.

Ejemplo en notación **do**

```
pream = do
 documentclass [] article
 usepackage [utf8] inputenc
 pagestyle headings
 title "Ejemplo"
 author "Daniel"
```

Turno de ejemplos

Ejemplos

Bibliografía

Referencias de LaTeX

- *The not so short introduction to LaTeX 2ε* [1995-2011] **Tobias Oetiker**, Huber Partl, Irene Hyna, Elisabeth Schlegl.

  `ftp://ftp.di.uminho.pt/pub/ctan/info/lshort/english/lshort.pdf`

- *The Beamer Class* [2003-2010]. **Till Tantau, Joseph Wright, Vedran Miletić**.

  `mirror.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf`

- *Hypertext marks in LaTeX: a manual for hyperref*. **Sebastian Rahtz, Heiko Oberdiek**.

  `latex.tugraz.at/_media/docs/hyperref.pdf`

Bibliografía

Referencias de Haskell

- *HaTeX, a monadic perspective of LaTeX* [2010]. **Daniel Díaz**.

  `http://hackage.haskell.org/package/HaTeX`

- *GHC standard libraries*. `Data.Monoid`, `Data.List`, `Data.String`, `System.Cmd`.

  `http://www.haskell.org/ghc/docs/latest/html/libraries`

- *Text package* [2008-2011]. **Bryan O'Sullivan**, Tom Harper, Duncan Coutts.

  `http://hackage.haskell.org/package/text`

- *ByteString package* [2007-2011]. **Bryan O'Sullivan**, Simon Marlow, David Roundy, Don Stewart.

  `http://hackage.haskell.org/package/bytestring`

Fin de la presentación

Turno de dudas y sugerencias.