Contents

Chapter 1

Common.BibTypes

```
module Common.BibTypes (
    BibTex(BibTex), Entry(Entry, entryType, reference, fields), Field(Field),
    getKey, getValue, maybegetKey, maybegetValue, EntryType, Reference,
    BibTexAlgebra, foldBibTex, lookupField, compareF
  ) where
```

The decision has been made to represent a BibTeX file as a list of entries, along with possibly some preamble.

```
data BibTex
```

`=  BibTex [String] [Entry]`

a bibtex file is just a list of entries, with a list of preamble-strings

```
instance Show BibTex
instance Tree BibTex
```

```
data Entry
```

`=`	`Entry`		
`entryType ::  EntryType`	book, thesis, etc.	`reference ::  Reference`	the name
			`fields :: [Field]`

a list of key/value pairs

a bibtex entry has a type, a reference (it's name), and a list of fields

```
instance Eq Entry
```

We implement equality on entries. When their names are the same, we consider them equal.

```
instance Show Entry
instance Tree Entry
```

```
data Field
     =  Field String String
```

a field is an attibute/value pair

```
instance Eq Field
```

Fields are considered equal when their keys are the same.

```
instance Ord Field
instance Show Field
instance Tree Field
```

```
getKey :: Field -> String
```

returns the key part, given a Field

```
getValue :: Field -> String
```

returns the value part, given a Field

```
maybegetKey :: Maybe Field -> Maybe String
```

sometimes we want to get the key from a Maybe Field.

```
maybegetValue :: Maybe Field -> Maybe String
```

...and sometimes we want to get the value from a Maybe Field.

```
type EntryType = String
```

the entry type is characterised by a string, for example `book`

```
type Reference = String
```

the reference is also just a string, such as `pierce02`

```
type BibTexAlgebra bibtex preamble entry = ([preamble]
                                            -> [entry] -> bibtex, String -> preamble, EntryType
                                                                                      -> Reference
                                                                                         -> [Field]
                                                                                            -> entry)
```

the type of the bibtex algebra. Used to fold over a bibtex library, like when we want to convert a BibTeX structure into HTML.

This seems the most natural way to define possible conversions from BibTex to other (possibly tree-like) formats, such as Html later on.

```
foldBibTex :: BibTexAlgebra bibtex preamble entry
              -> BibTex -> bibtex
```

How to fold over a BibTeX tree. Used when converting to Html in Bib2HTML.Tool.

```
lookupField :: String -> [Field] -> Maybe Field
```

Given a key, find the corresponding Field in a list of Fields. If it can't be found, Nothing is returned.

```
compareF :: Field -> Field -> Ordering
```

Our implementation of ordering on Fields. This is how we make sure that author, then title, then the other fields, and finally year, are displayed, regardless of how they are placed in the .bib file. This implementation allows us to simply run `sort` on a list of Fields.

Chapter 2

Common.HtmlTypes

```
module Common.HtmlTypes (
    Html(Html),  Head(Head),  Body,  BlockElem(A, Hr, Table, P),  Tr(Tr),  Td,
    Title,  HtmlAlgebra,  foldHtml
  ) where
```

This module contains our representation of an HTML document. Our simplistic version of HTML only knows anchors, horizontal rules, tables and paragraphs, but this is enough to display a BibTeX database.

```
data Html
```

```
     =  Html Head Body
```

an HTML document has a head and a body

```
instance Show Html
instance Tree Html
```

```
data Head
```

```
     =  Head Title
```

the head only contains the title.

```
instance Show Head
instance Tree Head
```

```
type Body = [BlockElem]
```

the body of an HTML document is a list of block elements (table, anchor, etc.)

```
data BlockElem
     =   A [Field] Reference
     |   Hr
     |   Table [Field] [Tr]
     |   P [Field] String
```

a block element can be (as stated) anchor, rule, table or a paragraph (a paragraph is also used to represent a plain string. In this case it's attribute list is empty, and when rendering, the p-tag is ommitted.

```
instance Show BlockElem
instance Tree BlockElem
```

```
data Tr
     =   Tr [Field] [Td]
```

a table row. It has attributes and a list of cells

```
instance Show Tr
instance Tree Tr
```

```
type Td = BlockElem
```

a table cell is a block element

```
type Title = String
```

the title of a document is simply a string.

```
type HtmlAlgebra html head body block tr = (head
                                            -> body -> html, (Title -> head, [block]
                                                                             -> body), ([Field]
                                                                                        -> Refer
                                                                                           -> bl
```

And once again we define an algebra for folding over an HTML document. This will prove useful when we want to pretty-print the html (or render it in any form), since the above type is only an abstract representation of an HTML document.

Since Html is tree-like, it seems logical to use a fold to convert it to some other (tree-like) format, in our case, this is Doc, the CCO pretty-print data structure. The actual fold for this is defined in PrettyPrintHTML.Tool.

```
foldHtml :: HtmlAlgebra html head body block tr -> Html -> html
```

The function which folds over an HTML tree, given an algebra as defined above.

Chapter 3

Common.TreeInstances

```
module Common.TreeInstances (
    bibfromTree,  bibtoTree,  entryfromTree,  entrytoTree,  fieldfromTree,
    fieldtoTree,  htmlfromTree,  htmltoTree,  headfromTree,  headtoTree,
    blockitemfromTree,  blockitemtoTree,  trfromTree,  trtoTree
  ) where
```

This module contains instances of Tree for various data structures. These are used for converting between ATerm and the given data structure. These functions aren't very complicated, they just allow `flattening` a datastructure into a portable format, and converting it back again.

`bibfromTree :: BibTex -> ATerm`

Converts from BibTex to ATerm

`bibtoTree :: ATerm -> Feedback BibTex`

Converts an ATerm back into a BibTex tree

`entryfromTree :: Entry -> ATerm`

Converts from bibtex Entry to ATerm

`entrytoTree :: ATerm -> Feedback Entry`

Converts an ATerm back into a BibTex entry

`fieldfromTree :: Field -> ATerm`

Converts from Field (key/value pair) to ATerm

`fieldtoTree :: ATerm -> Feedback Field`

Converts an ATerm back into a key/value pair

`htmlfromTree :: Html -> ATerm`

Converts from HTML document to ATerm

`htmltoTree :: ATerm -> Feedback Html`

Converts an ATerm back into an HTML document

`headfromTree :: Head -> ATerm`

Converts from HTML head element to ATerm

`headtoTree :: ATerm -> Feedback Head`

Converts an ATerm back into an html head element

`blockitemfromTree :: BlockElem -> ATerm`

Converts from HTML entity to ATerm

`blockitemtoTree :: ATerm -> Feedback BlockElem`

Converts an ATerm back into an html element

`trfromTree :: Tr -> ATerm`

Converts from HTML horizontal rule to ATerm

`trtoTree :: ATerm -> Feedback Tr`

Converts an ATerm back into a horizontal rule (html element)

Chapter 4

Main

```
module Main (

  ) where
```

Chapter 5

PrettyPrintHTML.Tool

```
module PrettyPrintHTML.Tool (
    mainFunc, pipeline, aTerm2Html, html2String, html2ppAlgebra,
    tagWithFields, printField
  ) where
```

This module contains the pp-html tool's code. It is a rather simple tool: it reads an Html ATerm, folds over the Html to convert it into Doc (the data structure provided by the CCO library for pretty-printing), and finally makes use of CCO's rendering function to output the Html tree as a string.

```
mainFunc :: IO ()
```

```
pipeline :: Component String String
```

the pipeline for pp-html. Read input, fold over resulting Html tree, and output with pretty-print built-in functions.

```
aTerm2Html :: Component ATerm Html
```

read an ATerm and return Html (abstract tree)

```
html2String :: Component Html String
```

pretty print the Html tree. Do this using html2ppAlgebra, an algebra to fold over the Html with.

`html2ppAlgebra :: HtmlAlgebra Doc Doc Doc Doc Doc`

the meat of this module. Defines an algebra that when used to fold over an Html structure, yields a Doc, which can be pretty-printed. The way this is done, is that for each possibly type found in Html, we define how to pretty-print it.

`tagWithFields`

`::`	`[Field]`	attribute list, pretty-printed with printField
`->`	`String`	the tag name, A for example
`->`	`Doc`	output pretty-print CCO format

helper function to create an html tag given the element name and a list of attributes.

`printField :: Field -> Doc`

used to print an attribute of an Html element. Simple, key=`value`.