Contents

Chapter 1

AbstractBib2HTML.Tool

```
module AbstractBib2HTML.Tool (
    mainFunc, pipeline, checkRequired, empty, sortFields, checkOptionals,
    checkEntry, sorter, sortGen, checkDups, bibTex2HTML, bib2htmlAlg,
    separate, generateIndex, generateTableRows, flattenEntry, formatFields
  ) where
```

---

this module implements the bib2html tool. It contains the complete pipeline for this part of the suite, converting from a BibTex ATerm into an Html ATerm. This is also where the main validation is done on the BibTeX database, to make sure that there are no duplicates, etc.

`mainFunc :: IO ()`

> the default main function. Just wrap a pipeline in IO

`pipeline :: Component String String`

> The pipeline for bib2html.

`checkRequired :: Component BibTex BibTex`

> checks a number of things. Make sure all required fields are in all entries, and that there are no fields which are unrecognised (such as misspelled fields). Also sorts the fields (author, title, the rest, finally year). Finally checks that all the other fields are at least optional, since we don't want disallowed fields in entries.

`empty :: Entry -> Bool`

this function returns whether a given entry has 0 fields. We don't want those.

`sortFields :: Entry -> Feedback Entry`

Sort fields, and nub duplicates. Here we just need to call `sort`, since the Ord class is implemented on Fields.

`checkOptionals :: Entry -> Feedback Entry`

Checks that all fields in an entry are at least optional. If they are not, issue a warning and empty them. These will later be pruned.

`checkEntry :: Entry -> Feedback Entry`

Check that all required fields are present, depending on the entry's type. If they aren't issue an error and stop. This is the only condition on which the bib2html program fails.

`sorter :: Component BibTex BibTex`

Sort entries by author, year, title. Makes use of the usual lexical sort on strings.

`sortGen`

`::`	`String`	the key to sort on
`->`	`Entry`	entry 1
`->`	`Entry`	entry 2
`->`	`Ordering`	

helper function which compares two entries and returns an ordering, based on the key requested.

`checkDups :: Component BibTex BibTex`

This function eliminates entries with the same name. The first found entry with a certain name is retained. A warning is also issued.

`bibTex2HTML :: Component BibTex Html`

Convert a BibTex tree to Html by folding with the bib2htmlAlg algebra.

`bib2htmlAlg :: BibTexAlgebra Html BlockElem ([BlockElem], Tr)`

this algebra converts a BibTex tree into an Html representation, including a list of hyperlinks at the top, maybe preamble blocks, and the table with the entries.

`separate :: [BlockElem] -> [BlockElem]`

used to place a pipe character between the list of hyperlinks

`generateIndex :: Reference -> [BlockElem]`

turns a reference into a hyperlink

`generateTableRows :: EntryType -> Reference -> [Field] -> Tr`

turns an Entry (or more accurately, given a type, a name, and a list of attributes) into a table row

`flattenEntry :: EntryType -> [Field] -> String`

given a type and a list of attributes, flattens an entry into a string, for placement in a table cell

`formatFields :: Field -> String`

format a field into a string. Possibly do special formatting things, depending on if it's a title, for example.

Chapter 2

Common.ATermUtils

```
module Common.ATermUtils (
    aTerm2String,  aTerm2BibTex,  html2Aterm
  ) where
```

This module contains some generally useful functions for manipulating ATerms

`aTerm2String :: Component ATerm String`

> Render a given ATerm as a string. Used by all the tools that need to output ATerms to the terminal.

`aTerm2BibTex :: Component ATerm BibTex`

> Parse an ATerm into a BibTex data structure.

`html2Aterm :: Component Html ATerm`

> This function takes an abstract HTML tree and converts is into an ATerm

Chapter 3

Common.AllowedFields

```
module Common.AllowedFields (
    allowedTable, FieldTable, RequiredKey, OptionalKey, keysBook,
    keysBooklet, keysConference, keysInbook, keysIncollection,
    keysInproceedings, keysManual, keysMastersthesis, keysMisc,
    keysPhdthesis, keysProceedings, keysTechreport, keysUnpublished,
    keysArticle
  ) where
```

This file is a place to store a table containing allowed BibTeX entry types, and the required and optional fields for each. Adapted from `http://en.wikipedia.org/wiki/BibTeX`

`allowedTable :: [FieldTable]`

> a table containing all the allowed BibTeX types and their attributes

`type FieldTable = (EntryType, ([RequiredKey], [OptionalKey]))`

> An entry mapping from the BibTeX entry type to the list of required and optional keys

`type RequiredKey = String`

> A required key is simply defined by the string-representation of it's attribute name.

```
type OptionalKey = String
```

Like required keys, optional keys are strings.

```
keysBook :: FieldTable

keysBooklet :: FieldTable

keysConference :: FieldTable

keysInbook :: FieldTable

keysIncollection :: FieldTable

keysInproceedings :: FieldTable

keysManual :: FieldTable

keysMastersthesis :: FieldTable

keysMisc :: FieldTable

keysPhdthesis :: FieldTable

keysProceedings :: FieldTable

keysTechreport :: FieldTable

keysUnpublished :: FieldTable

keysArticle :: FieldTable
```

Chapter 4

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 5

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 6

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 7

Main

```
module Main (

  ) where
```
