xBib : The language design and implementation of a transformation language

PEPIJN VISSER, University of Twente, The Netherlands

BibTeX is one of the standards for managing lists of references for academic papers. But due to different publishers generating their BibTeX files, there is no universal standard and inconsistencies between similar items can emerge, leading to confusion and mistakes. In this paper, we propose a transformation language for BibTeX called xBib, that allows the user to create a bibliographic list that looks more professional and uniform. Using this language we design a tool that can be used to transform .bib files in a clear and simple manner.

Additional Key Words and Phrases: transformation, language design, bibtex

1 INTRODUCTION

When researchers are writing an academic paper, they will use many different sources and reference them inside the paper. It is not unusual for a paper to have dozens of references in its bibliography list. One of the universal standards for storing these references is called BibTeX. When an author wants to reference a paper, they would get the .bib data from a library source, and store it in their bibliographic list. These sources are mainly from two types of article providers.

The first is from academic conferences. Examples of these are the IEEE/ACM international conferences. These conferences publish a so-called *conference proceeding* which is a collection of academic papers, typically made by the researchers at the conference. Conferences include lots of information in their .bib files like the number of the conference in the series, the location and date of the conference, and other, in most cases, useless information.

On the other hand, we have services that collect and search articles and papers in a vast collection of libraries; think of services like Google Scholar. Because these services do not belong to a specific conference or library, they often fall short of the specifics. Often the generalisation of all the various libraries results in abbreviation inconsistencies, where some libraries say ACM, others say Association for Computing Machinery, and in some cases even include the specific year and series.

Currently there is no good method of handling these bibliography files, which can grow to quite large amounts of data. Researchers tend to do some of these transformations manually and this can get cumbersome and slow. In other cases the bibliography file will get messy and can look unprofessional.

1.1 Problem Statement

We can categorise the problem into three categories: Information overload, inconsistencies and duplicates.

TScIT 38, Februari, 2023, Enschede, The Netherlands

© 2022 University of Twente, Faculty of Electrical Engineering, Mathematics and Computer Science.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

1.1.1 Information overload. One of the problems with the various ways in which BibTeX files are being put out, is that there is a varied level of detail in the description of different papers. This can cause issues as the reference list can quickly become overloaded with information that can be seen as unimportant to the reader. This is especially an issue when there is a maximum number of allowed pages in a paper. A single line of unimportant data can grow to become quite a lump of text if there are tens of references included. An example of information overload can be seen in the published articles of conferences like IEEE/ACM. These files often include the exact date, time and location of the conference, which is of unimportance to a normal reader of the article.

1.1.2 Inconsistencies. Another issue arises when big services like Google Scholar and DBLP use web crawlers to get information about the articles. Due to the massive amount of articles and authors, there is a chance for inconsistencies to arise. Due to the nature of BibTeX, mistakes in the capitalisation of article/book titles can arise. For some titles, it is of great importance for the article's clarity that the capitalisation is correct. Some publishers convert all text in a .bib file to lowercase, which can create errors.

Another inconsistency is in the way that BibTeX treats capitalisation. **Example:** *"This is a TITLE"* and *{This is a TITLE}* will both be accepted, as BibTeX treats the characters "" and {} as the same. But in some cases the capitalisation of this title will get lost, as BibTeX will convert every character except for the first one into lowercase. A way to fix this is to add another pair of curly brackets ({}) around the case-sensitive words. **Example:** *{This is a TITLE}* might output: *This is a title* or even *This is a Title*, whereas *{This is a {TITLE}}* will only output: *This is a TITLE*.

1.1.3 Duplicates. A smaller issue with getting references from different sources or publishers is that there is a chance for duplicates to emerge. These duplicates would be easy to spot if they had the same format, but due to the problem described above, they could look quite different to each other. This would form a problem and an unnecessarily large bibliography list.

1.2 Research Questions

To solve the problem described in 1.1 we will design the transformation language xBib and implement it using a support tool. The following research questions follow from the problem statement:

- **RQ1:** What kind of transformations are needed in the domain of BibTeX transformation to solve the inconsistent BibTeX format?
- **RQ2:** How do we define a transformation language that can grow and cover the domain of the problem, while being simple to understand?
- **RQ3:** How do we design a parser and support tool to implement xBib into a working program?

2 RELATED WORK

In this section we will go over some related work in language design and transformation languages.

There are multiple reference management software that also have some features that xBib needs. Mendeley [3] and Endnote [2] both have smart solutions for the abbreviation of conferences and journals, among other things. These software both allow the user to export their bibliography files, but this does mean that the user has to download and use a big piece of software. The advantage of a smaller language like xBib is that it does not need a large download, and can be used as a small tool or plugin.

When designing a program, coding conventions are an important topic to discuss, as they define the best practices to write a program. In 2016, Goncharenko and Zaytsev [6] wrote a paper about a language on top of CSS, that can express coding conventions. Here they describe the process of language design for a language on top of CSS, and an interpreter of this language that can automatically detect violations in the coding conventions. There are many overlapping aspects between CssCoCo (the language they proposed) and xBib, as they both created a language on top of an existing language. CssCoCo is a language that detects violations, and warns the user about them, whereas xBib has a more functional application. Allamanis et al. [4] wrote a paper about NATURALIZE, which can study a code base, and suggest improvements to the code. Prause and Jarke [8] propose gamification as a way to promote coding conventions to developers.

BibSLEIGH [11] is an ongoing project by Zaytsev that lies in the same domain as xBib. In a general sense, xBib could be the successor of BibSLEIGH. BibSLEIGH uses a JSON library to crosscheck and store BibTeX files and has some basic functionality for the removal of URLs and abbreviating or elongating some well-established journals. The main difference is that BibSLEIGH is not a transformation language, but a big database with some data management tools.

Much research has been done on the process of defining a language. Steele [9] wrote a paper that describes the importance of a growing language. The advice that Steele gave has helped the process of designing xBib tremendously. Other types of advice for language design are the paper by Wijngaarden [10] about orthogonal design and the paper by Erwig and Walkingshaw [5] about semantics first. Moody [7] wrote a paper discussing the design flaws and improvements to be made in visual notations in software engineering. Zaytsev [12] wrote about a way to approach the design of a DSL and how not to make mistakes that have crumbled other languages. The paper proposes a DSL toolkit to be used whilst designing a language.

3 DOMAIN ANALYSIS

Before we can analyse the domain, we first have to learn the syntax of BibTeX, as the terms will be prevalent in the analysis. The structure shown in 1 is part of the BiblaTeX cheat sheet [1] and shows what some of the terms mean in the context of a BibTeX file.

```
@<entrytype>{<key>,  e.g.
   <field> = <value>,
   <field> = <value>,
   <field> = <value>,
   ...}
```

```
@book{tolkien-hobbit,
  author = {Tolkien, J. R. R.},
  title  = {The Hobbit},
  date   = {YYYY-MM-DD},
  ...}
```

Fig. 1. The structure of BibTeX

The problem raised in section 1.1 has to be broken down into transformations that the user can call upon. To do that we have broken down the problem into four main categories. These categories then each have different transformations that as a whole, cover the entire domain of the problem.

3.1 Formatting

The first category covers all of the problems that have to do with formatting. The focus for this category is on how the output will look, and make the whole bibliography list a uniform document. These transformations will not change any information, they merely alter the document's look.

Table 1. Formatting transformations

Name	Description	Parameters
Indentation	Choose what, and how much, indentation is used for every field	spaces / tabs
String symbol	Choose what symbol is used to define a string	curly brackets / quotes
Last comma	Select if the last row in each entry should have a comma, or not	-
Line wrap	Set a threshold to enable line wrap after a specified length	-

3.2 Order

This category focuses on transformations that have to do with the order and output of the program. This covers any problems that have to do with specific entries but is not about altering any information.

Table 2. Order transformations

Name	Description	Parameters
Sort	Sort the entries by key	-
Filter	The output will only show the entries with the key provided by the filter	List of keys
Smart filter	The output will be filtered to show only cited entries	The references file

3.3 Content

The third category solves the problems that have to do with the generic information of an entry. The transformations will not change the field data, but rather the general entry data.

Table 3. Content transformations

Name	Description	Parameters
Remove duplicates	If enabled, remove any duplicate entries and combine the information	-
Prefer uri	Set the preferred type of uri, and change the other types to match it	doi / url
Change type	Every entry of the given type will change to the other type	Pair of keys
Rename key	The specified key will be renamed.	Pair of keys
Blind	The given entry will be made completely anonymous	Key of entry
Validate	Validates any entry if the necessary fields are present	-

3.4 Field specifics

The last category solves the remaining issues, focused on the field specifics. These transformations are not specific to any type of field but are rather more generalised actions that can be called for any field, and it is up to the user to use the correct ones. These actions can be expanded on in the future and are by no means finished, but some of these actions are;

Abbreviate, flip name/surname, declutter, remove.

4 LANGUAGE DESIGN

With the domain of the problem analysed we had a good idea of what was necessary for the language syntax. We wanted to design a language that was simple but also had room to grow and evolve. If the language had no room to grow then that could lead to any later implementations not fitting within the scope of the designed language, and this was something we wanted to avoid. The language should be as simple as possible as xBib is intended to be used by researchers of any field, that could have no prior experience with programming. Since the problems described are applicable to any academic research, this was imperative.

Table 4. A simplified grammar of xBib

```
bib       : input command* output
input     : filename
command   : 'go' category item*
          | 'field' argument argument
category  : ('format' | 'order' | 'content')
output    : filename
item      : 'set' argument*
          | 'action' argument*
          | ('enable' | 'disable') argument
argument  : '(' argument+ ')'
          | primitive
primitive : (integer | unquoted | quoted)
```

As the domain analysis showed, we can categorise the problem into four categories: formatting, order, content and field actions. This categorisation is something we wanted to feature in the language syntax since this made it clear to the user what has to be written at any location in the code. We can further categorise the first three categories from the latter, as field actions are contextually different from the rest. Since field actions are detached from any big umbrella, they can differ extremely from each other, whereas the other three categories all have functionality that fits under their umbrella category. **For example:** The field actions *abbreviate* and *flip name/surname* do not share a similar context, whereas the formatting transformations *indentation* and *line wrap* all have to do with the formatting of the output.

The process of designing the xBib language was an iterative process, where we analysed each iteration based on intuitiveness for the user and simplicity for further expansion. There is much room for creativity in the designing process of a language, and many choices are neither good nor bad, but merely the decision that the creator made. A simplified grammar of xBib can be seen in table 4, as a result of the iterations.

For simplicity, we assumed that only one input and output file were allowed as this made the implementation process much easier. The xBib language accepts multiple commands, which can be either a category or a field action. For any of the three categories, we can fit all the possible transformations within three types: *set*, *action* and *enable or disable*.

The *set* type corresponds to any transformation that changes a variable, an example is given below:

```
go format (
    set indentation to (tab, 1),
    set string_identifier to quotes,
    set line_wrap to 25
);
```

Some of these variables will have a default value set and can be overwritten, whereas others will only be called when there is a value set by the user.

Action corresponds to a transformation that behaves like a function; There are one or multiple input variables and there is some

expected result. Unlike the other types, this type does not have any default values, and the action will only be performed when called upon by the user. An example of the usage can be seen below:

```
go order(
    action filter ('exampleKey', 'exampleKey2')
);
go content(
    action change_type ('book', 'article'),
    action rename_key ('tolkien-hobbit', 'hobbit'),
    action blind 'exampleKey'
);
```

The last type is a form of flag, where the user can either enable or disable it. The type of function that the user flags differs for each category, and so does the use case. A user can flag a setting, e.g. *last comma* can be enabled to change the formatting. A flag can also enable a function that has no variables, e.g. *remove duplicates* can be enabled to allow duplicates to be removed automatically. Example:

```
go format (
    enable last_comma
);
go order(
    enable sort,
    enable smart_filter
);
```

Field actions have two arguments, the first is the fields for which the actions will be called on, the second argument is the actions to be taken. Since multiple fields can have the same actions called, and multiple actions can be applied to the same field, we assumed that both the first and second parameter can have multiple arguments paired between brackets. This allows the user to have cleaner and simpler code. The following example shows a paired and non paired field action call:

```
field ('author','editor') (abbreviate, flip_name);
field 'title' capitalise;
```

5 TOOL SUPPORT

With the domain analysed and the language syntax created, the next step is to create a tool to support the language. This tool can then be used by the end user to transform .bib files. The tool is written in Java, using ANTRL4 to parse and execute the code. The simplified version of the grammar shown in table 4 is extended and can be used to parse a .xbib file which contains all the transformations.

The program will consist of two parts. First, it should parse the .xbib file and store all the commands in a single class. Secondly, it should pass these commands to a second parser that transforms the bibliography file based on the aforementioned stored commands. The grammar that describes xBib generates a listener, which is used to take actions when specific nodes are reached in the parsing process. We extended the base listener to store the data in a special class that stores all the generated commands. After the listener has completed and visited the entire parse tree, the class has stored all the commands, which can be passed on to the second part of the process.

```
public void exitMain(xbibParser.MainContext ctx) {
    // Import file from the path given
    String in = getFileContents(ctx.in.getText());

    // Parse the .bib file
    CharStream chars = CharStreams.fromString(in);
    Lexer lexer = new simpleBibTeXLexer(chars);
    simpleBibTeXParser parser =
    new simpleBibTeXParser(new CommonTokenStream(lexer));
    ParseTree tree = parser.database();

    // Run the second parseTree
    BibTeXListener listener = new BibTeXListener();
    listener.run(tree, commands);

    // The parser is finished, get the results
    String res = listener.getResult();

    Path out = getPath(ctx.out.getText());

    FileWriter fileWriter = null;
    try {
        fileWriter = new FileWriter(out.toString());
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
    // Write the output to a new .bib file
    PrintWriter printWriter = new PrintWriter(fileWriter);
    printWriter.print(res);
    printWriter.close();
}
```

The code above shows a snippet from the entire listener code, namely the final node that the listener reaches, called *exitMain*. This is the final step of the first part of the program. At this point in the program, the listener has walked the entire tree, and we can assume that the command class has been filled accordingly. The next step would be to find the specified BibTeX file that was named as input in the xBib file. A second simple grammar is used to parse the .bib file, and this parser also has a listener class. The entire process of the second part is conceptually similar to the first part, but instead of gathering data during parsing, we alter the parsed data at specific locations.

The grammar to parse the BibTeX file is based on the structure given by the BiblaTeX Cheat Sheet [1]. The domain analysis discussed in 3 shows what transformations have to be achieved. Each transformation has a unique process to be implemented, and their location in the listener process is different for each of them. The following snippet of Java code shows the node *exitStringValue*.

```
public void exitStringValue(StringValueContext ctx) {
  String s = ctx.String().toString();

  // String identifier
  Item i = getCommand(commands.getFormat(),
           Item.Call.set, "string_identifier");
  if (i != null) {
      switch (i.getArguments().toArray()[0].toString()) {
          case "quotes":
              s = String.format("\"%s\"", s.substring(1,
                  s.length()-1));
              break;
          case "curly":
              s = String.format("{%s}", s.substring(1,
                s.length()-1));
              break;
          default:
              addError(ctx, "The argument %s can only be
                       of type (quotes, curly)",
                       i.getArguments().toArray()[0]);
              break;
      }
  }
  result.append(s);
}
```

The code shows the implementation of the formatting transformation *string identifier*. The *getCommand* function gets the command with the name *string_identifier* from the command class. If this object exists, then the xBib file had a call made for the string identifier to be changed, if not, then there was no mention of the string identifier, and the transformation won't happen. The code then finds the provided argument, which in this case has to be either *quotes* or *curly*, and transforms the string accordingly. Every transformation has a similar process as *string identifier* and is not worth explaining in this paper. The code can be found at https://github.com/visperr/xbib.

6 CONCLUSION & DISCUSSION

We have analysed the domain of the problem that currently exists with BibTeX bibliographies. To solve this solution we proposed three research questions that lead to this end goal. We have made a domain analysis, discussing the problems and what transformations can solve them. Based on the analysis we were able to define a language syntax for xBib that is simple and efficient and finally implement the xBib language into a support tool. The support tool can read an xBib file, grab a BibTeX file, transform it based on the specified commands and output it to the desired file.

There are still aspects that can be researched further. The parser used for parsing BibTeX files is a simple parser that has some flaws, mainly that it is unable to parse strings of multiple lines. No research has been done with respect to the speed of different parsing methods or using a different language for the support tool. The support tool can be seen as a prototype to implement xBib, and therefore these flaws fall out of the scope of the research.

With most of the features implemented, the support tool has all the functionality to solve the problems described in section 1.1. The problem of information overload can be solved by allowing the user to remove unnecessary fields. The problem of duplicates and inconsistencies is solved by letting the user have control over the look of the output, creating a unified document.

REFERENCES

[1] 2017. BiblaTeX cheat sheet. http://tug.ctan.org/info/biblatex-cheatsheet/biblatex-cheatsheet.pdf [Online; accessed 21. Jan. 2023].

[2] 2022. EndNote | The best reference management tool. https://endnote.com [Online; accessed 23. Nov. 2022].

[3] 2022. Mendeley - Reference Management Software. https://www.mendeley.com [Online; accessed 23. Nov. 2022].

[4] Miltiadis Allamanis, Earl T Barr, Christian Bird, and Charles Sutton. 2014. Learning natural coding conventions. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 281–293.

[5] Martin Erwig and Eric Walkingshaw. 2012. Semantics first. In *Proceedings of the Fourth International Conference on Software Language Engineering, SLE*, Vol. 11. Springer, 243–262.

[6] Boryana Goncharenko and Vadim Zaytsev. 2016. Language design and implementation for the domain of coding conventions. In *SLE 2016: Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*. Association for Computing Machinery, New York, NY, USA, 90–104. https://doi.org/10.1145/2997364.2997386

[7] Daniel L. Moody. 2009. The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering* 35 (2009).

[8] Christian R Prause and Matthias Jarke. 2015. Gamification for enforcing coding conventions. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering*. 649–660.

[9] Guy L Steele Jr. 1998. Growing a language. In *Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*.

[10] Adriaan van Wijngaarden. 1965. Orthogonal design and description of a formal language. (Jan. 1965).

[11] Vadim Zaytsev. 2017. BibSLEIGH: Bibliography of Software (Language) Engineering in Generated Hypertext. In *Post-proceedings of the Eighth Seminar in Series on Advanced Techniques and Tools for Software Evolution (SATToSE 2015) (CEUR Workshop Proceedings, Vol. 1820)*, Anya Helene Bagge, Tom Mens, and Haidar Osman (Eds.). CEUR-WS.org, 54–64. http://ceur-ws.org/Vol-1820/paper-06.pdf

[12] Vadim Zaytsev. 2017. Language Design with Intent. In *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 45–52. https://doi.org/10.1109/MODELS.2017.16