博士研究生学位论文

题目：测试文档

姓　　名：某某

学　　号：0123456789

院　　系：某某学院

专　　业：某某专业

研究方向：某某方向

导　　师：某某教授

某年某月

版权声明

摘要

pkuthss 文档模版最常见问题:

在最终打印和提交论文之前，请将 pkuthss 文档类选项中的 `colorlinks` 改为 `nocolorlinks`，因为图书馆要求电子版论文的目录必须为黑色，且某些教务要求打印版论文的文字部分为纯黑色而非灰度打印。

`\cite`、`\parencite` 和 `\supercite` 三个命令分别产生未格式化的、带方括号的和上标且带方括号的引用标记：**test-en**，[**test-zh**]、[test-en, test-zh]。

若要避免章末空白页，请在调用 pkuthss 文档类时加入 openany 选项。

如果编译时不出参考文献，请参考 `texdoc pkuthss`“问题及其解决”一章“其它可能存在的问题”一节中关于 biber 的说明。

关键词：其一，其二

Test Document

Test (Some Major)
Directed by Prof. Somebody

Abstract

Test of the English abstract.

Keywords: First, Second

目录

序言

pkuthss 文档模版最常见问题:

在最终打印和提交论文之前，请将 pkuthss 文档类选项中的 `colorlinks` 改为 `nocolorlinks`，因为图书馆要求电子版论文的目录必须为黑色，且某些教务要求打印版论文的文字部分为纯黑色而非灰度打印。

`\cite`、`\parencite` 和 `\supercite` 三个命令分别产生未格式化的、带方括号的和上标且带方括号的引用标记：**test-en**，[**test-zh**]、[**test-en**, **test-zh**]。

若要避免章末空白页，请在调用 pkuthss 文档类时加入 openany 选项。

如果编译时不出参考文献，请参考 `texdoc pkuthss`“问题及其解决”一章“其它可能存在的问题”一节中关于 biber 的说明。

0.1 Introduction

现在，许多程序分析工具都涉及代码修改功能。在这些工具中，有许多都是代码修复工具 [30, 29, 49, 44]。通常来说，修复工具的输入是一段代码和一组测试，并不断修改代码直至代码能通过测试。另一些程序分析工具是API升级工具 [32, 48, 42]。当API升级时出现了不兼容情况时，这些工具可以自动更新相应的API调用让程序与API契合。我们把这类直接修改代码的工具称作程序编辑工具。

A lot of program analysis tools involve direct modification of source code. A notable category of such tools is program-repair tools [30, 29, 49, 44]. These tools take a program and a set of tests as input, and modify the program until all tests pass. Another category is API evolution tools [32, 48, 42]. When an API is upgraded with incompatible changes, these tools automatically change the API invocations to comply with the new API. We call such tools that directly modify the source code *program-editing tools*.

另一方面，许多程序语言的实现都带有预处理器 [9, 27, 31]。最常见的预处理器是C预处理器（C++）。许多程序语言也接受C++，包括C，C++和Objective-C。同时，程序员也时常使用C++来写一些零散的小工具。这时就会使用到预处理器。例如Korpela [28]曾在文章中描述过用C++写一个HTML编辑工具：这个工具会把页面间相同的HTML代码转换成C的宏，而不是直接生成HTML页面。然后页面再利用这些宏最终生成HTML文件。

On the other hand, many programming languages are provided with preprocessors [9, 27, 31]. The most widely used preprocessor is the C preprocessor (CPP). Many programming languages adopt CPP, notable C, C++, and Objective-C. Furthermore, CPP is often used by programmers in casual situations as a general purpose tool, where the preprocessor is added as a building step for any language used. For example, Korpela [28] describes the use CPP as an HTML authoring tool: instead of writing HTML pages directly, the shared code pieces between HTML pages are first defined as C macros, and HTML pages using these macros are preprocessed into final HTML files.

程序编辑工具通常不会去修改程序的预处理指令。但是，只有能够把修改反映射到预处理之前的代码的工具才算有用。只在预处理后的代码中修复错误会导致原有程序再次编译的时候错误依然存在，这样毫无意义。这个问题具有挑战性，因为工具必须同时能理解预处理命令和目标程序语言，同时保证修改在两边能保持一致。事实上，现有的程序编辑工具往往无法正确处理预处理指令、或是直接不处理预处理指令，例如现有的C语言工具：GenProg [30, 29]，RSRepair [49]，和SemFix [44]。这三个工具都只在预处理后的代码上工作。用户需要手动检查预处理后的代码变化，并自行修改源代码——而这又增加了新bug的可能。

Program editing tools usually do not directly change preprocessor directives. However the tools must be able to map results back to the unpreprocessed source to be useful. There is no point of fixing a bug in the preprocessed code and only to have it overwritten when the unchanged source is compiled again. This is challenging, as the tools must be able to understand both the preprocessor directives and the target programming languages, and make sure whatever changes made on both levels are consistent with each other. As a matter of fact, existing program-editing tools often fail to produce correct results under the presence of preprocessor directives, or give up dealing with them entirely. We have investigated the implementations of three influential bug-fixing tools on the C programming language: GenProg [30, 29], RSRepair [49], and SemFix [44]. All the three implementations work only on preprocessed code. Users have to manually inspect the preprocessed code, and copy the changes to the original source code—risking of introducing new bugs in the process.

代码重构是一个密切相关的领域 [41, 17]。在代码重构中，程序编辑工具有时需要直接修改预处理指令。比如：用户有时想重命名一个宏，或者需要提取一个宏作为重构的一部分。在这种情况下，工具开发者别无选择，只好修改预处理指令。典型情况中，工具开发者会定义一种新的C语法使得原有的C语法和预处理指令能兼容。但是，如果我们考虑更一般的程序编辑工具，这种方法就捉襟见肘了。首先，工具开发者需要在真正设计工具之前把精力花费在学习语言的细节上。其次，学习新语言的努力并不能在其他语言中复用。

A closely related area is refactoring [41, 17], where tools are expected to directly manipulate preprocessor directives. For example, one may well want to rename a macro or extract a macro as part of the refactoring. In this case, tool builders have no choice but to bite the bullet and confront the preprocessor directly. Typically a new C grammar is designed such that it incorporates both the original C grammar and the preprocessor directives. However, when applied to a more wider range of code editing tools, such almost brute force approaches exhibit obvious shortcomings. First, tool developers using such a grammar basically have to start from scratch: they have to learn the new grammar and leave behind the existing tool chains on C. Second, the effort spend on the new grammars is specialized and cannot be reused for other languages, which basically rules out casual uses of CPP.

本文中我们提出了一个轻量级的支持C++的程序编辑工具实现方法。该系统时一个双向的C++预处理器：原有的预处理过程可以背看作一个正向变换，在此基础上我们添加一个能够把预处理后代码上的修改反响映射回去的反向变换。于是，程序编辑工具现在可以关注于预处理后的代码，并把映射修改的交给我们的自动工具[1]。

[1] 这个过程并不是全自动的。因为我们的工具现在支持程序编辑的基本操作。尽管这些步骤可以用通用代码差分方法实现 [11]，但是如果工具能直接提供基本编辑操作可以有最好的效果

In this paper we propose a lightweight approach to support CPP in program-editing tools. Our system acts as a bidirectional CPP: the original preprocessing can be considered as a forward program transformation, and we add to it a corresponding backward transformation that maps changes on the preprocessed code back into the unpreprocessed source. As a result, program-editing tools can now focus on preprocessed programs, and have results (almost) automatically reflected to the source[2].

在这里我们列举一些例子：（1）上文中所提到的三个学术届认可的错误修复系统现在可以处理预处理前的代码；（2）API升级软件现在可以在有预处理代码的情况下更好地实现；（3）所有并不需要关心预处理过程的程序编辑工具都能够被改善。

We list a few examples here: (1) as mentioned above the implementations of the three state-of-the-art bug-fixing approaches only deal with preprocessed code; (2) the API evolution tools mentioned previously can also be implemented more conveniently by only dealing with preprocessed code; (3) all program-editing tools on languages that do not formally rely on CPP naturally fall into this category because the programs may be put under casual uses of CPP.

虽然现在存在着几种双向变换的技术 [40, 57, 58]，但是它们都是为数据的转换设计的。给定一个源数据集`s`，一个变换程序`p`，和一个目标数据集`t = p(s)`，这些方法试图将`t`上的变化描述成`s`的变化。然而，C的预处理器与数据的转换不同，因为C的源代码不仅包含了作为数据的代码，还包含了座位变换程序的预处理指令。这就要求反向变换的机制能处理更复杂的情况：当目标数据发生变化时，我们可能要变化源数据、转换程序、或者是二者都变化。

Although there exist several bidirectionalization techniques [40, 57, 58], they are designed for data transformation: given a source data set `s`, a transformation program `p`, and a target data set `t = p(s)`, these approaches try to reflect the changes on `t` to `s`. The C preprocessor differs from this data transformation scenario in the way that the unpreprocessed source program actually contains both the data set and the transformation program. This added complication amplifies the variance of the backward transformation: when

[2] It is not entirely automatic because the tools need to support the extraction of the changes on the preprocessed programs. Although this step can be performed by generic code differencing [11], extracting the changes directly from the tools gives the best result.

the target data is changed, we may change either the source data set, the transformation program, or both.

A key design novelty in our approach that serves to control this variance is to allow, but at the same time minimize, changes to the transformation programs. First, our approach never introduces new macro definitions or modifies existing macro definitions, effectively confining the impact of the reflected changes to a local scope. Second, our approach only removes existing macro invocations but never invent new ones. Furthermore, we will consider removing macro invocations only when necessary. In this way, we maintain the existing structure of the original program source as much as possible.

Implementing this design is also challenging. Typical approaches to bidirectionalization [40, 57, 39] decompose the transformation along the abstract syntax tree of the program, where each subtree corresponds to a small bidirectional transformation that collectively forms the final transformation. However, a CPP program cannot be easily parsed into a tree structure. For example, in the following piece of code,

```
#define inc(x) 1+x
#define double(x) 2*x
inc(double) 2
inc(double) (x)
```

the first `inc(double)` independently expands to a new segment, but the second `inc(double)` is only part of an expansion, as the expanded `double` will form a new macro invocation with `(x)` to be recursively invoked. As a result, we cannot treat `inc(double)` as an independent unit and derive a backward transformation from it. To overcome this difficulty, we propose a new model for interpreting CPP programs. Instead of parsing a CPP program into an abstract syntax tree, we view the preprocessing as applying a set of rewriting rules to the code. This interpretation enables the bidirectionalization of a CPP program as bidirectionalization of each rewriting rule.

Furthermore, our approach is proved to be correct, in the sense of the following round-trip laws (1) if the preprocessed program is not changed, the unpreprocessed program will not be changed, and (2) preprocessing the unpreprocessed program with the reflected changes will produce exactly the same changed preprocessed program. These two properties are known as GETPUT and PUTGET [13] in the bidirectional transformation literature.

To sum up, our paper makes the following contributions:

- We propose a lightweight approach to handling the C preprocessor in program-editing tools based bidirectional transformations. We analyze different design alternatives and propose five requirements for defining the behavior of the backward transformation, including GETPUT and PUTGET (Section **??**).
- We propose an algorithm that meets the five requirements. This algorithms is based on an interpretation of CPP as a set of rewriting rules, which structurally decomposes the bidirectionalization of CPP into the bidirectionalizatio of each rule
- We evaluate our approach on the Linux kernel and compare our approach with two baseline approaches: one reflecting changes by copying back the entire changed file and one reflecting changes by copying back the changed lines. The evaluation results show that our approach breaks much less macro invocations, and always produces correct results while the other two do not

Finally, we discuss related work in the paper in

参考文献

[1] B. Adams *et al.* "*Can We Refactor Conditional Compilation into Aspects?*" In: *Proceedings of the 8th ACM International Conference on Aspect-oriented Software Development*. Charlottesville, Virginia, USA: ACM, **2009**: 243–254. http://doi.acm.org/10.1145/1509239.1509274.

[2] F. Bancilhon and N. Spyratos. "*Update Semantics of Relational Views*". *ACM Trans. Database Syst.* **1981**, *6*(4): 557–575.

[3] I. D. Baxter and M. Mehlich. "*Preprocessor Conditional Removal by Simple Partial Evaluation*". In: *Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01)*. Washington, DC, USA: IEEE Computer Society, **2001**: 281–. http://dl.acm.org/citation.cfm?id=832308.837146.

[4] Ed. by G. C. Necula and P. Wadler. "*Boomerang: resourceful lenses for string data*". In: *POPL*. ACM, **2008**: 407–419.

[5] Q. Boucher *et al.* "*Tag and Prune: A Pragmatic Approach to Software Product Line Implementation*". In: *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*. Antwerp, Belgium: ACM, **2010**: 333–336. http://doi.acm.org/10.1145/1858996.1859064.

[6] Y. Cui, J. Widom and J. L. Wiener. "*Tracing the Lineage of View Data in a Warehousing Environment*". *ACM Trans. Database Syst.* 2000-06: 179–227. http://doi.acm.org/10.1145/357775.357777.

[7] U. Dayal and P. A. Bernstein. "*On the Correct Translation of Update Operations on Relational Views*". *ACM Trans. Database Syst.* **1982**, *7*(3): 381–416.

[8] Ed. by J. Whittle, T. Clark and T. Kühne. "*From State- to Delta-Based Bidirectional Model Transformations: The Symmetric Case*". In: *Model Driven Engineering Languages and Systems*. Springer Berlin Heidelberg, **2011**: 304–318. http://dx.doi.org/10.1007/978-3-642-24485-8_22.

[9] M. D. Ernst, G. J. Badros and D. Notkin. "*An empirical analysis of C preprocessor use*". *Software Engineering, IEEE Transactions on*, **2002**, *28*(12): 1146–1170.

[10] L. Fegaras. "*Propagating updates through XML views using lineage tracing*". In: *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, 2010-03: 309–320.

[11] B. Fluri *et al.* “*Change distilling: Tree differencing for fine-grained source code change extraction*”. *Software Engineering, IEEE Transactions on*, **2007**, *33*(11): 725–743.

[12] Ed. by J. Hook and P. Thiemann. “*Quotient lenses*”. In: *ICFP*. ACM, **2008**: 383–396.

[13] J. N. Foster *et al.* “*Combinators for Bidirectional Tree Transformations: A Linguistic Approach to the View-update Problem*”. *ACM Trans. Program. Lang. Syst.* 2007-05.

[14] Ed. by J. Gibbons. “*Three Complementary Approaches to Bidirectional Programming*”. In: *SSGIP*. Springer, **2010**: 1–46.

[15] A. Garrido and R. Johnson. “*Analyzing Multiple Configurations of a C Program*”. In: *Proceedings of the 21st IEEE International Conference on Software Maintenance*. Washington, DC, USA: IEEE Computer Society, **2005**: 379–388. http://dx.doi.org/10.1109/ICSM.2005.23.

[16] A. Garrido and R. Johnson. “*Challenges of Refactoring C Programs*”. In: *Proceedings of the International Workshop on Principles of Software Evolution*. Orlando, Florida: ACM, **2002**: 6–14. http://doi.acm.org/10.1145/512035.512039.

[17] A. Garrido and R. Johnson. “*Embracing the C preprocessor during refactoring*”. *Journal of Software: Evolution and Process*, **2013**, *25*(12): 1285–1304. http://dx.doi.org/10.1002/smr.1603.

[18] P. Gazzillo and R. Grimm. “*SuperC: Parsing All of C by Taming the Preprocessor*”. In: *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*. Beijing, China: ACM, **2012**: 323–334. http://doi.acm.org/10.1145/2254064.2254103.

[19] Ed. by S. Abiteboul and P. C. Kanellakis. “*Foundations of Canonical Update Support for Closed Database Views*”. In: *ICDT*. Springer, **1990**: 422–436.

[20] Ed. by P. Hudak and S. Weirich. “*Bidirectionalizing graph transformations*”. In: *ICFP*. ACM, **2010**: 205–216.

[21] S. Hidaka *et al.* “*GRoundTram: An Integrated Framework for Developing Well-behaved Bidirectional Model Transformations*”. In: *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. Washington, DC, USA: IEEE Computer Society, **2011**: 480–483. http://dx.doi.org/10.1109/ASE.2011.6100104.

[22] M. Hofmann, B. Pierce and D. Wagner. “*Edit Lenses*”. In: *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. Philadelphia, PA, USA: ACM, **2012**: 495–508. http://doi.acm.org/10.1145/2103656.2103715.

[23] Ed. by N. Heintze and P. Sestoft. “*A programmable editor for developing structured documents based on bidirectional transformations*”. In: *PEPM*. ACM, **2004**: 178–189.

[24] ISO/IEC JTC 1, SC 22, WG 14. *Programming languages – C*, 2011-04.

[25] C. Kästner *et al.* “*Variability-aware Parsing in the Presence of Lexical Macros and Conditional Compilation*”. In: *Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications*. Portland, Oregon, USA: ACM, **2011**: 805–824. http://doi.acm.org/10.1145/2048066.2048128.

[26] D. Kim *et al.* “*Automatic patch generation learned from human-written patches*”. In: *ICSE*, **2013**: 802–811.

[27] E. Kohlbecker *et al.* “*Hygienic macro expansion*”. In: *LISP and functional programming*, **1986**: 151–161.

[28] J. Korpela. *Using a C preprocessor as an HTML authoring tool*, **2000**.

[29] C. Le Goues *et al.* “*A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each*”. In: *ICSE*, **2012**: 3–13.

[30] C. Le Goues *et al.* “*GenProg: A generic method for automatic software repair*”. *Software Engineering, IEEE Transactions on*, **2012**, *38*(1): 54–72.

[31] B. Lee *et al.* “*Marco: Safe, expressive macros for any language*”. In: *ECOOP*. Springer, **2012**: 589–613.

[32] J. Li *et al.* “*SWIN: Towards Type-Safe Java Program Adaptation between APIs*”. In: *PEPM*, **2015**: 91–102.

[33] J. Liebig, C. Kästner and S. Apel. “*Analyzing the Discipline of Preprocessor Annotations in 30 Million Lines of C Code*”. In: *Proceedings of the Tenth International Conference on Aspect-oriented Software Development*. Porto de Galinhas, Brazil: ACM, **2011**: 191–202. http://doi.acm.org/10.1145/1960275.1960299.

[34] J. Liebig *et al.* “*Scalable Analysis of Variable Software*”. In: *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. Saint Petersburg, Russia: ACM, **2013**: 81–91. http://doi.acm.org/10.1145/2491411.2491437.

[35] D. Lohmann *et al.* “*A Quantitative Analysis of Aspects in the eCos Kernel*”. In: *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*. Leuven, Belgium: ACM, **2006**: 191–204. http://doi.acm.org/10.1145/1217935.1217954.

[36] K. Matsuda and M. Wang. “*“Bidirectionalization for Free” for Monomorphic Transformations*”. *Science of Computer Programming*, **2014**. http://www.sciencedirect.com/science/article/pii/S016764

[37] Ed. by R. Peña and T. Schrijvers. “*Bidirectionalization for free with runtime recording: or, a light-weight approach to the view-update problem*”. In: *PPDP*. ACM, **2013**: 297–308.

[38] Ed. by M. Felleisen and P. Gardner. “*FliPpr: A Prettier Invertible Printing System*”. In: *Programming Languages and Systems*. Springer Berlin Heidelberg, **2013**: 101–120. http://dx.doi.org/10.1007/978-3-642-37036-6_6.

[39] Ed. by A. D. Gordon. “*A Grammar-Based Approach to Invertible Programs*”. In: *ESOP*. Springer, **2010**: 448–467.

[40] Ed. by R. Hinze and N. Ramsey. “*Bidirectionalization transformation based on automatic derivation of view complement functions*”. In: *ICFP*. ACM, **2007**: 47–58.

[41] B. McCloskey and E. Brewer. “*ASTEC: A New Approach to Refactoring C*”. In: *ESEC/FSE-13*, **2005**: 21–30.

[42] N. Meng, M. Kim and K. S. McKinley. “*Systematic Editing: Generating Program Transformations from an Example*”. In: *PLDI*, **2011**: 329–342.

[43] Ed. by W.-N. Chin. “*An Algebraic Approach to Bi-directional Updating*”. In: *APLAS*. Springer, **2004**: 2–20.

[44] H. D. T. Nguyen *et al.* “*SemFix: Program repair via semantic analysis*”. In: *ICSE*, **2013**: 772–781.

[45] I. (Object Management Group. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*.

[46] J. L. Overbey, F. Behrang and M. Hafiz. “*A Foundation for Refactoring C with Macros*”. In: *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. Hong Kong, China: ACM, **2014**: 75–85. http://doi.acm.org/10.1145/2635868.2635908.

[47] Y. Padioleau. “*Parsing C/C++ Code Without Pre-processing*”. In: *Proceedings of the 18th International Conference on Compiler Construction: Held As Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009*. York, UK: Springer-Verlag, **2009**: 109–125. http://dx.doi.org/10.1007/978-3-642-00722-4_9.

[48] Y. Padioleau *et al.* “*Semantic Patches for Documenting and Automating Collateral Evolutions in Linux Device Drivers*”. In: *PLOS*, **2006**.

[49] Y. Qi *et al.* “*The strength of random search on automated program repair*”. In: *ICSE*, **2014**: 254–265.

[50] R. Rajkumar *et al. Lenses for Web Data*, **2013**.

[51] Ed. by E. Mayr, G. Schmidt and G. Tinhofer. “*Specification of graph translators with triple graph grammars*”. In: *Graph-Theoretic Concepts in Computer Science*. Springer Berlin Heidelberg, **1995**: 151–163. http://dx.doi.org/10.1007/3-540-59071-4_45.

[52] Ed. by H. Ehrig *et al.* “*15 Years of Triple Graph Grammars*”. In: *Graph Transformations*. Springer Berlin Heidelberg, **2008**: 411–425. http://dx.doi.org/10.1007/978-3-540-87405-8_28.

[53] H. Spencer and G. Collyer. *#ifdef Considered Harmful, or Portability Experience With C News*, **1992**.

[54] D. Spinellis. “*Global Analysis and Transformations in Preprocessed Languages*”. *IEEE Trans. Softw. Eng.* 2003-11: 1019–1030. http://dx.doi.org/10.1109/TSE.2003.1245303.

[55] P. Stevens. “*Bidirectional model transformations in QVT: semantic issues and open questions*”. *Software & Systems Modeling*, **2010**, *9*(1): 7–20. http://dx.doi.org/10.1007/s10270-008-0109-9.

[56] M. Vittek. "*Refactoring browser with preprocessor*". In: *Software Maintenance and Reengineering, 2003. Proceedings. Seventh European Conference on*, 2003-03: 101–110.

[57] Ed. by Z. Shao and B. C. Pierce. "*Bidirectionalization for free! (Pearl)*". In: *POPL*. ACM, **2009**: 165–176.

[58] J. Voigtländer *et al.* "*Combining syntactic and semantic bidirectionalization*". In: *ICFP*, **2010**: 181–192.

[59] J. Voigtländer *et al.* "*Enhancing semantic bidirectionalization via shape bidirectionalizer plug-ins*". *J. Funct. Program.* **2013**, *23*(5): 515–551.

[60] Ed. by M. M. T. Chakravarty, Z. Hu and O. Danvy. "*Incremental updates for efficient bidirectional transformations*". In: *ICFP*. ACM, **2011**: 392–403.

[61] Ed. by W.-N. Chin and J. Hage. "*Semantic bidirectionalization revisited*". In: *PEPM*. ACM, **2014**: 51–62.

[62] Ed. by C. Bolduc, J. Desharnais and B. Ktari. "*Gradual Refinement: Blending Pattern Matching with Data Abstraction*". In: *MPC*. Springer, **2010**: 397–425.

[63] M. Wang *et al.* "*Refactoring pattern matching*". *Sci. Comput. Program.* **2013**, *78*(11): 2216–2242.

[64] X. Wang *et al.* "*Automating presentation changes in dynamic web applications via collaborative hybrid analysis*". In: *FSE*, **2012**: 16.

[65] D. Weise and R. Crew. "*Programmable Syntax Macros*". In: *Proceedings of the ACM SIGPLAN 1993 Conference on Programming Language Design and Implementation*. Albuquerque, New Mexico, USA: ACM, **1993**: 156–165. http://doi.acm.org/10.1145/155090.155105.

附录 A　附件

pkuthss 文档模版最常见问题:

在最终打印和提交论文之前，请将 pkuthss 文档类选项中的 `colorlinks` 改为 `nocolorlinks`，因为图书馆要求电子版论文的目录必须为黑色，且某些教务要求打印版论文的文字部分为纯黑色而非灰度打印。

`\cite`、`\parencite` 和 `\supercite` 三个命令分别产生未格式化的、带方括号的和上标且带方括号的引用标记：**test-en**，[**test-zh**]、[test-en, test-zh]。

若要避免章末空白页，请在调用 pkuthss 文档类时加入 openany 选项。

如果编译时不出参考文献，请参考 `texdoc pkuthss`“问题及其解决”一章“其它可能存在的问题”一节中关于 biber 的说明。

致谢

pkuthss 文档模版最常见问题:

在最终打印和提交论文之前，请将 pkuthss 文档类选项中的 `colorlinks` 改为 `nocolorlinks`，因为图书馆要求电子版论文的目录必须为黑色，且某些教务要求打印版论文的文字部分为纯黑色而非灰度打印。

`\cite`、`\parencite` 和 `\supercite` 三个命令分别产生未格式化的、带方括号的和上标且带方括号的引用标记：**test-en**，[**test-zh**]、[**test-en**, **test-zh**]。

若要避免章末空白页，请在调用 pkuthss 文档类时加入 openany 选项。

如果编译时不出参考文献，请参考 `texdoc pkuthss`“问题及其解决”一章“其它可能存在的问题”一节中关于 biber 的说明。

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名：　　　　　　日期：　　年　　月　　日

学位论文使用授权说明

（必须装订在提交学校图书馆的印刷本）

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

- 按照学校要求提交学位论文的印刷本和电子版本；
- 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；
- 学校可以采用影印、缩印、数字化或其它复制手段保存论文；
- 因某种特殊原因需要延迟发布学位论文电子版，授权学校在 □ 一年 / □ 两年 / □ 三年以后在校园网上全文发布。

（保密论文在解密后遵守此规定）

论文作者签名：　　　　　　导师签名：　　　　　　日期：　　年　　月　　日