

**研究生学位论文开题报告**

**Graduate Thesis/Dissertation Proposal**

|  |  |
| --- | --- |
| **学号 Student ID** | 121033910117 |
| **姓名 Name** | 刘思雨 |
| **学生类别 Degree Program** | 专业型硕士生 Professional Master Student |
| **学习形式 Study Mode** | 全日制 Full-time |
| **导师 Supervisor(s)** | 汪宇霆 |
| **论文题目 Thesis title** | 基于静态单赋值中间语言的函数式编译器验证方法 |
| **学院 School** | 电子信息与电气工程学院 |
| **专业 Major** | 电子信息（085400） |
| **开题日期 Date** |  |
| **开题地点 Venue** |  |

填 报 说 明

**Instruction**

1. 校本部研究生的开题报告应通过[数字交大](http://my.sjtu.edu.cn/)在线提交申请，填写本表并上传系统。特殊情况下经研究生院事先同意，可不上传系统，并使用《上海交通大学研究生论文开题评审表》完成评审。

The application for thesis/dissertation proposal should be submitted online through [My SJTU](http://my.sjtu.edu.cn/). The student shall filled this form and upload it in the system. Under special circumstance, this form does not need to be uploaded and the review can be proceeded with the review form with prior consent from the graduate school.

1. 开题报告为A4大小，于左侧装订成册。各栏空格不够时，请自行加页。考核前提前一周送交导师、评审专家审阅。

This form should be printed with A4 papers and bound together on the left. If the space left is not enough, please feel free to add extra pages. The print version shall be sent to the supervisor, and the review committee members for review at least one week before the oral presentation.

1. 博士生导师可以根据博士生学位论文选题情况自行确定是否进行开题查新，博士学位论文开题查新报告应由查新工作站提供。

The supervisor should decide, based on the proposed topics, whether a novelty assessment report is needed or not, which should be conducted by an authorized novelty assessment department.

1. 开题报告通过后，定稿版开题报告由研究生、导师各存档一份，无需上传系统。

Upon passing the proposal, the final version of this report shall be archived by the graduate student and his/her supervisors for future reference.

1. 医学院研究生如果以函评形式开题，开题地点请填写“函评”，专家组组长签名由导师签名。

For students in the School of Medicine, if the dissertation proposal is conducted via peer review, the “Proposal venue” shall be filled with “peer review” and the “Signature of Committee Chair” shall be signed by the supervisor.

|  |  |
| --- | --- |
| 论文题目  Proposed Title | **基于静态单赋值中间语言的函数式编译器验证方法** |
| 研究课题来源  Source of Research Project | 请在合适选项前画√ Please select proper options by “√”.  国家自然科学基金课题 NSFC Research Grants  国家社会科学基金 National Social Science Fund of China  国家重大科研专项 National Key Research Projects  其它纵向科研课题 Other Governmental Research Grants  企业横向课题 R&D Projects from Industry  自拟课题 Self-proposed Project  其它 Other |

1. **请综述课题****国内外研究进展、现状、挑战与意义，可分节描述。博士生不少于10,000汉字，硕士生不少于5,000汉字。请在文中标注参考文献。 Please review the frontier, current status, challenges and significance of the research topic. The citations should be marked in the context and listed in order at the end of this section. No less than 8,000 words for doctoral students and 4,000 words for master students if written in English.**
   1. 国内外研究进展

编译器作为重要的系统软件，关于其效率和正确性的研究一直是国内外学者不断探讨的方向。随着形式化方法理论和工具的发展，编译器验证研究也有了长足的进步。本文探讨如何将关键的函数式编译过程和编译器基础设施中常用的静态单赋值形式通过形式化方法联系起来。国内外关于函数式编译器、静态单赋值中间语言以及编译器验证的研究已经有了许多进展和成果。

对于典型的函数式编译器，源程序通常被转化为延续传递风格（Continuation Passing Style或CPS）的中间语言，然后针对CPS程序进行各类优化，最终转换为更低阶的语言。关于函数式编译器中的CPS转换的研究将在1.1.1节中进行介绍。

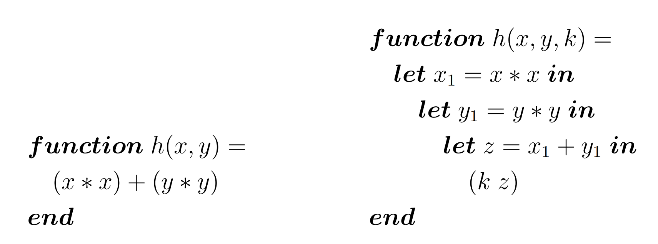
为了控制验证的工作量和难度，编译验证工作一般采用自行开发的中间表示语言(Intermediate Representation或IR)和编译器后端。而当今主流的程序语言大多使用编译器基础设施（如LLVM和GCC）作为其后端[5]（例如Rust, Swift等语言的编译器[6,27]）。这些设施提供了统一的中间表示语言（如静态单赋值（Static Single Assignment或SSA）形式），可以模块化地组合后端阶段的编译和优化过程，并被各种编译器前端复用。在1.1.2节中对有关静态单赋值中间语言性质的研究进行介绍，并在1.1.3节中介绍关于CPS与SSA之间联系与转换算法的研究成果。

现代编译器规模日趋庞大，其中的错误和漏洞也越来越复杂。传统的软件测试方法只能后验查找已经存在的编译错误。相比之下，基于形式化验证可以通过对编译器的中间语言给予严格的程序语义、并通过将编译过程在逻辑系统中形式化表述，从数学层面验证目标程序保存源程序的语义。该语义保存概念(Semantics Preservation)从逻辑上确定了编译过程的正确性，可以在设计和实现阶段排除编译器中的绝大部分漏洞。编译器形式化验证领域的代表性工作与进展将在1.1.4节中进行介绍。

* + 1. 函数式编译器与延续传递风格（CPS）

函数式编程（Functional Programming）是一种通过应用和组合函数来构建程序的编程范式，以λ演算(λ-Calculus)作为其数学基础[10,29]。表达函数式程序控制流的形式有很多种，如典型的直接风格（Direct Style）和延续传递风格（CPS）。图1展示了同样含义的程序在这两种风格下的例子。CPS中的延续（Continuation）指的是代表当前执行节点后所有剩余计算的函数，该函数需要当前计算得到的结果作为输入。CPS的关键特性在于通过延续明确的表达程序控制流。具体来说，CPS函数需要将延续作为额外的参数传入，当运行该函数得到计算结果后，程序通过调用该延续来传递这个值，也就是“返回”该结果。

以图1为例，函数h在CPS中的延续参数为k，在得到最终结果z之后，程序调用(k z)以“返回”z，并执行k所代表的剩余计算。当调用CPS函数时，调用者需要提供一个延续函数来表示剩余计算，且在CPS中所有中间结果、控制流中的控制点都需要被明确命名。这些特点导致用户直接用CPS形式编写代码较为困难，但是由于明确表示的控制流利于程序分析和优化，CPS是编译器中常见的中间表示形式。除此之外，CPS在程序语义学中也被广泛研究[16]。



(a)直接风格程序 (b) CPS风格程序

图1 函数式程序的直接风格和CPS形式

Plotkin[35]、Danvy和Nielsen[36]等人研究了将直接风格的函数式程序转换为CPS程序的方法。其中Plotkin的方法需要后续进行管理性缩减（Administrative Reduction），以去除冗余的λ结构。Felleisen等人提出了另一种方法非组合式的方法，其基于λ-演算的语义。Danvy和Nielsen将这两种方法联系起来，可以只通过一步转换完成，不需要在后续过程中进行管理性缩减。

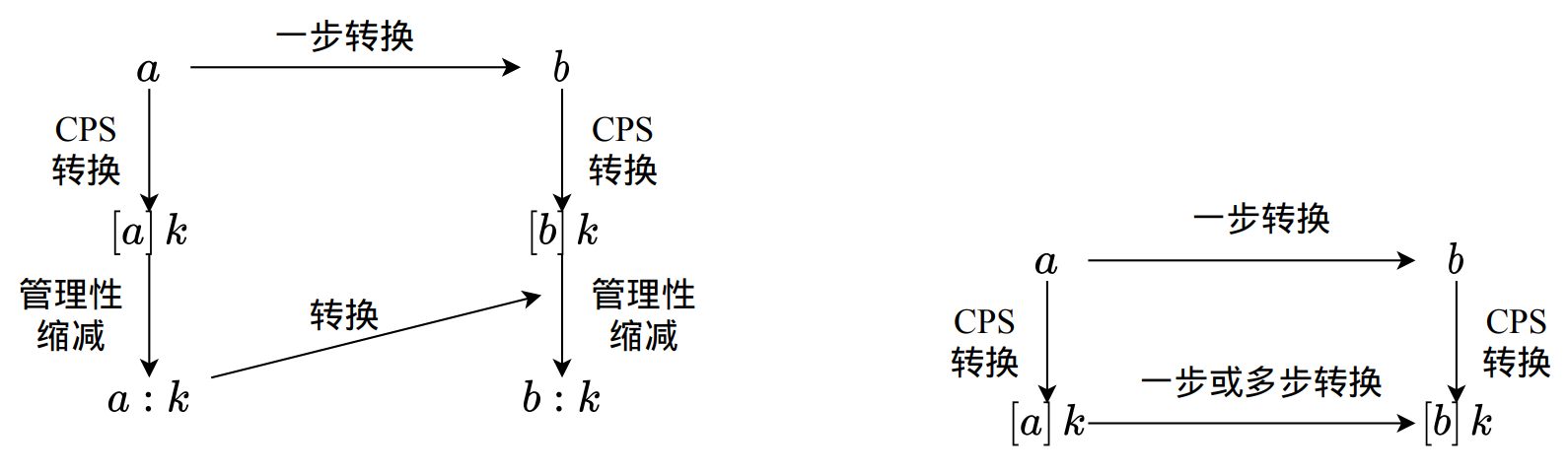


图2 Plotkin、Danvy和Nielsen的CPS转换方法

* + 1. 静态单赋值（SSA）中间语言

SSA是命令式语言编译器中的一种中间表示类型。在SSA中，每个变量只能有一处赋值，且被使用之前已经被赋值，这种性质很接近函数式语言中的名字绑定（Name Binding）[17]。变量被划分为不同的版本，新版本的变量往往使用原名加下标来表示，以使每个定义得到自己的版本。这样一来，变量的使用定义链（Use-def Chains）更加清晰，从而使许多编译器优化算法在SSA中间语言上能够更好地实现，例如常量传播、无用代码消除、寄存器分配等[17-18]。

基于SSA的命令式程序的控制流图如图2，基本块的前序基本块可能是或者自身，不同的前序块对变量有不同的赋值。所以，将和中被赋值的重命名为不同的名字，然后在初始处的Φ函数中把它们作为参数，根据实际流向得到此处选择的的版本。

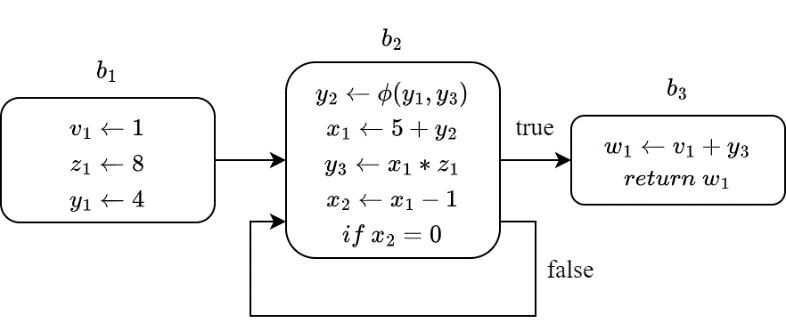


图3 SSA程序控制流图

* + 1. CPS与SSA的关系及转换

Appel等人[20] 发现SSA也是一种函数式语言：λ演算和SSA虽然有不同的形式，但它们所做的工作其实是相同的。L. Beringer[17] 给出了SSA的函数式表示，将SSA中的术语与函数式编程中的概念联系起来。比如，函数式语言中的let绑定对应着SSA中的赋值、绑定变量的词法作用域对应着SSA中的可支配区域。Let 绑定与变量定义点之间的对应关系还延续到了程序结构的其他方面。与命令式语言中的返回地址或函数指针相似，延续指定了当前代码片段求值完成后程序下一步该如何执行。SSA中的控制流图对应着函数式编程中的函数，虽然顶层延续的参数在控制流图中并不是明确可见的，但它对应一个命令式调用中用来保存返回地址的地方。

Kelsey[18]通过将CPS中的λ进行标注来区分完整过程、跳转、延续，从而进一步从语法上将CPS程序转换为SSA程序。同理，也可以将SSA程序转换为被标注的CPS程序。Kelsey的转换算法没有关注过程间的流程分析，而是选择先将CPS过程进行合并，再作为一个整体进行转换。在Kelsey的CPS语言中，二元运算表达式参数、条件语句的条件等等也是表达式，而不是变量，也就意味着它的控制流更像是带有延续的普通函数式程序。

* + 1. 编译器形式化验证

经过形式化验证的C语言编译器CompCert[3]是由INRIA的Xavier Leroy领导开发的。它支持将C语言的一个重要子集翻译为多个体系结构（包括PowerPC、ARM、X86、RISC-V）的汇编代码，并基于Coq定理证明工具完整验证了该过程的正确性。CompCert已被应用于诸如核电站控制软件和飞行控制系统的开发。CompCert提供了一套验证编译过程正确性的理论和框架，采用模拟（Simulation）技术证明了语义保存性质。

国内也有基于CompCert的L2C编译器[4]，被用于安全关键的工业领域。L2C 的源语言是被广泛用于安全关键的工业领域（高铁、核电站等）的 Lustre，这些类型的应用对开发工具本身的安全性要求很高。它的目标语言是ComperCert中使用的C子集Clight。

除命令式语言编译验证之外，针对函数式语言编译验证的代表性工作是CakeML[33]，其验证了从标准ML函数式语言(SML)到汇编和机器代码的编译过程，整个开发在HOL4定理证明工具中完成。

* 1. 国内外相关研究现状

国内外的研究者已经注意到使函数式编译器复用基于SSA中间语言的基础设施后端的重要性。例如，为了利用主流编译器基础设施后端的优势，工业级函数式编译器正在迁移到基于LLVM的后端上，在1.2.1节中对此进行详细介绍。在经验证的可靠编译器领域，将CPS函数式语言编译到Clight的工作将函数式编译器与成熟的高可信编译器CompCert连接起来。1.2.2节中介绍了这种完整的经验证的函数式编译器，但它的中间语言不是基于SSA的。1.2.3节中介绍了CompCert一个新的基于SSA中间语言的中端，它将RTL语言转换为SSA，经过全局值编号（Global Value Numbering，GVN）优化，再转换回RTL。

* + 1. SML-New Jersey基于LLVM编译器后端的版本[8,30]

SML-New Jersey（Standard ML of New Jersey，SML/NJ）编译器在新版本中将基于MLRisc框架的后端进行更改，将CPS语言编译到CFG IR，再将其编译到基于静态单赋值的LLVM IR。SML/NJ后端读入高阶的CPS中间语言，经过一系列转换，得到一阶CPS中间语言。其后端的最后一步是从MLRisc程序生成机器代码。

在新版本的后端中，不再生成MLRisc，而是将CPS转换为控制流图（CFG）中间语言，并进一步生成基于SSA的LLVM IR。这样的工作说明CPS到基于SSA的目标语言的编译是值得推动的工作。但它不是经过形式化验证的，在可靠性上有所不足。

* + 1. Gallina经验证的编译器CertiCoq[9]

CertiCoq将Gallina编译到了C语言的子集Clight，从而与CompCert连接起来，最终编译到汇编语言。L6是一种带有相互递归函数和模式匹配的CPS语言。从纯函数式的L6语言生成了命令式的C语言，L6中的每一个函数对应着Clight中的一个函数。

在Coq中对该从CPS到C的编译过程进行了实现和正确性验证。CertiCoq中定义了L6 CPS语言的大步操作语义。它的目标语言不是基于SSA的，所以无法使用基于静态单赋值语言完成的各种优化。

* + 1. CompCert经验证的基于SSA的中端[34]

原本的CompCert并不使用基于SSA的中间语言。而在这个版本的CompCert中，先将RTL中间语言转换为SSA程序，对其进行针对SSA的GVN优化，再转换回RTL，与CompCert接下来的编译过程连接起来。GVN优化是指对相同值的变量赋予相同的编号，从而消除冗余的计算。这里的SSA程序状态与RTL程序状态类似，除了寄存器和当前函数类型要进行修改。ϕ函数没有放在基本代码块中，ϕ指令代码块与普通指令构成的基本块是平行的结构。它们的语义也是平行定义的，对基本代码块的控制流图定义了小步操作语义，对ϕ代码块定义了大步操作语义。对于指令Inop pc’，如果pc’是一个连接点，就去执行依附在这个点上的ϕ代码块，然后控制流才能回到pc’。定义该SSA语言的操作语义和结构上的约束条件，利用它的全局属性对程序的局部优化进行推理。这样，就得到了一个经验证的、基于SSA的中端。

这样的工作提供了一个经验证的、基于SSA中间语言的编译器。虽然它的源语言是Clight，但是其他语言也可以直接编译到SSA中间语言，复用之后的优化及编译过程，得到完整的经验证的编译链。

* 1. 研究面临的问题和挑战

在编译器验证领域，许多工作是围绕CompCert编译器展开的。这也就使得研究者们在确立函数式语言的编译目标时，为了使之与CompCert链接起来，选择将函数式语言编译到Clight。现有的函数式编译器验证工作主要分2种：一种使用基于CompCert的框架进行验证，但是由于CompCert原先的版本不支持SSA格式，因此无法和编译器基础设施实现连接；另外一种基于逻辑关系(Logical Relations)，面临的问题在于缺少从CPS到SSA的编译过程。在使用模拟技术进行验证的框架下，从函数式IR到命令式IR的编译器普遍选择了CompCert作为编译目标，而不是SSA IR。例如Gallina的编译器CertiCoq[9]，它的目标语言是C语言的子集Clight，将其与CompCert连接了起来。这样一来，虽然得到了经完整验证的函数式编译器，但是无法提供与基于SSA中间语言的主流基础设施连接起来的桥梁，也就无法复用针对SSA程序的各种优化来提升效率。

虽然已有工作探讨了CPS到SSA语言的转换算法，并且SML/NJ的新版本后端实现了CPS语言到基于SSA的LLVM IR的编译，但是这些都是未经验证的。在安全关键的领域中仍无法提供可靠的经验证的基于SSA中间语言的函数式编译器。

如果直接选择LLVM IR作为编译过程的目标语言，后续后端优化及生成机器代码的过程是没有经过验证的，无法得到一条经过完整验证的编译链。对LLVM后端编译过程进行验证是一项巨大的工作。Vellvm在Coq中对LLVM抽象语法树进行了定义，并提供了LLVM IR的形式化语义。但是它的小步操作语义版本已经停止维护。正在维护的新版本已经将操作语义改为基于ITree的语义[32]，难以与前端中CPS语言的操作语义联系起来。并且，从LLVM IR到机器代码的编译过程也是未经验证的。所以，目标SSA语言的选择也是研究面临的一项问题。

* 1. 研究意义

编译器作为关键系统软件之一，其正确性对于计算机系统的安全运行有重要意义。这是由于编译器可能在转换程序的过程中引入错误，导致目标程序的行为和源程序不一致，进而使得在源程序端花费大量精力的测试和验证工作在目标程序层级失效。因此，工业界长期以来对保障编译器正确性问题非常重视。例如，按照航空领域的 RTCA DO­178B/C 标准[1]，需要按照机载软件的要求一样对待编译器。关于编译器形式化验证的研究是值得关注的工作。

在学术界和工业界中，函数式编程的应用日趋广泛。传统命令式编程中，计算通过执行程序语句序更新程序状态来实现。相比之下，函数式编程中程序由包含函数定义和应用的表达式构成表示为表达式构成，而计算通过对表达式求值来实现。函数式程序的一大特点是函数可以作为参数传递，或者被其他函数返回，形成所谓的高阶函数[26]。此外，纯粹函数式程序的执行不会引起改变程序状态的副作用。这些特点使得函数式程序设计语言编写的程序更加简洁、安全和易于验证[11]，因此在并发编程、系统内存编程等方面获得了成功应用。除了Haskell[12]等纯函数式语言、OCaml、Erlang、Scala [13-14]等语言都对函数式编程有内生的支持，且诸多命令式编程语言如C++和Rust也在积极的引入函数式编程机制。建造高效、可靠的函数式程序编译器将为许多应用领域带来益处。

为了构建高效的、经验证的函数式编译器，需要将CPS函数式程序与SSA中间语言连接起来。建立了这样的连接，才能够使函数式编译器复用针对SSA中间语言的程序分析与优化过程，从而实现编译可靠性与效率的双重保障。在经验证的函数式编译器中，通常使用CompCert框架作为其后端。然而主流编译器基础设施的中间语言是基于SSA的。如何将经过形式化验证的函数式编译器与现代成熟的主流编译器组合起来，以得到更加高效和通用的可信编译器，是一个值得研究的问题。其中，将SSA中间语言作为函数式编译器前端的目标语言是一项关键的工作。

从1.1.1节关于SML/NJ新后端的介绍中可以看到，为了利用主流编译器基础设施后端的优势，工业级函数式编译器正在迁移到基于SSA的后端上。新的后端选择放弃MLRisc，而将CPS编译到LLVM IR，从而与LLVM框架连接起来。那么，对于经验证的函数式编译器，将CPS编译到SSA IR也是一个值得努力的工作。如果能够将函数式语言与SSA中间语言连接起来，就可以使高可靠的函数式编译器复用针对SSA的后端优化，在编译性能上进行提升。在高可靠函数式编译器领域中，提供将CPS与SSA语言连接起来的桥梁，就可以为连接主流编译器基础设施打下基础。

参考文献 Reference：

1. Author List, *paper title*, Journal **Volume**, pages, year.
2. Authors, *book name*, publisher, pages, year.
3. Brosgol B, Comar C. DO­178C: *A new standard for software safety certification* [R], ADA CORE TECHNOLOGIES NEW YORK NY, 2010.
4. Yang X, Chen Y, Eide E, et al. *Finding and understanding bugs in C compilers*, Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation. 2011: 283­294.
5. Leroy X. *A formally verified compiler back­end*[J]. Journal of Automated Reasoning, 2009, 43(4): 363­446.
6. SHANG S, GAN Y K, SHI G, et al. *Key Translations of the Trustworthy Compiler L2C and Its Design and Implementation*[J]. Journal of Software, 2017: 05.
7. Lattner C. *Introduction to the llvm compiler infrastructure*[C], Itanium conference and expo. 2006.
8. Balasubramanian A, Baranowski M S, Burtsev A, et al. *System programming in Rust: Beyond safety*[C], Proceedings of the 16th Workshop on Hot Topics in Operating Systems. 2017: 156­161.
9. Lattner C. *LLVM and Clang: Next generation compiler technology*[C], The BSD conference: vol. 5. 2008.
10. Farvardin K, Reppy J. *A New Backend for Standard ML of New Jersey*[C]. IFL 2020: Proceedings of the 32nd Symposium on Implementation and Application of Functional Languages. 2020: 55-66.
11. Bélanger O S, Weaver M Z, Appel A W. *Certified code generation from CPS to C*[J]. preparation. https://www. cs. princeton. edu/~ appel/papers/CPStoC.pdf, 2019
12. Church A. *The Calculi of Lambda Conversion*.(AM-6), Volume 6[M], Princeton University Press, 2016.
13. Hudak P. *Conception, evolution, and application of functional programming languages*[J]. ACM Computing Surveys (CSUR), 1989, 21(3): 359­411.
14. O’Sullivan B, Goerzen J, Stewart D B. *Real world haskell: Code you can believe in*[M]. ” O’Reilly Media, Inc.”, 2008.
15. Cesarini F, Thompson S. *Erlang programming: a concurrent approach to software development*[M]. ” O’Reilly Media, Inc.”, 2009.
16. Odersky M, Rompf T. *Unifying functional and object­oriented programming with scala*[J]. Communications of the ACM, 2014, 57(4): 76­86.
17. Skarsaune M. *The SICS Java Port Project: automatic translation of a large system from Smalltalk to Java*[D]. 2008.
18. Boudol G. The π­calculus in direct style[J]. *Higher­Order and Symbolic Computation*, 1998, 11(2): 177­208.
19. L. Beringer J S, Rastello F. *Static Single Assignment Book*[M]. Springer Science & Business Media, 2018.
20. Kelsey R A. *A correspondence between continuation passing style and static single assignment form*[J]. ACM SIGPLAN Notices, 1995, 30(3): 13­22.
21. Bolz C F, Cuni A, Fijalkowski M, et al. *Tracing the meta­level: PyPy’s tracing JIT compiler*[C], Proceedings of the 4th workshop on the Implementation, Compilation, Optimization of Object­Oriented Languages and Programming Systems. 2009: 18­25.
22. Appel A W. *SSA is functional programming*[J]. ACM SIGPLAN Notices, 1998, 33(4): 17­20.
23. Zhao J, Nagarakatte S, Martin M M K, et al. *Formalizing the LLVM intermediate representation for verified program transformations*[C], Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. 2012: 427-440.
24. Dowek G, Lévy J J. *Introduction to the theory of programming languages*[M]. Springer Science & Business Media, 2010
25. Charguéraud A. *The locally nameless representation[J]. Journal of automated reasoning*, 2012, 49(3): 363-408.
26. Kennedy A. *Compiling with continuations, continued*[C], Proceedings of the 12th ACM SIGPLAN international conference on Functional programming. 2007: 177­190.
27. Smith J B. *Ocamllex and Ocamlyacc*[J]. Practical OCaml, 2007: 193-211.
28. Sussman G J, Steele G L. *Scheme: A interpreter for extended lambda calculus*[J]. Higher-Order and Symbolic Computation, 1998, 11(4): 405-439.
29. Zhang Y, Yang M, Zhou B, et al. *Swift: a register-based JIT compiler for embedded JVMs*[C], Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments. 2012: 63-74.
30. Filaretti D, Maffeis S. *An executable formal semantics of PHP*[C], European Conference on Object-Oriented Programming. Springer, Berlin, Heidelberg, 2014: 567-592.
31. Church A. *A formulation of the simple theory of types*[J]. The journal of symbolic logic, 1940, 5(2): 56-68.
32. Appel A W. *Compiling with continuations*[M]. Cambridge university press, 2007.
33. Plotkin G D. *LCF considered as a programming language*[J]. Theoretical computer science, 1977, 5(3): 223­255.
34. Zakowski Y, Beck C, Yoon I, et al. *Modular, compositional, and executable formal semantics for LLVM IR*[J]. Proceedings of the ACM on Programming Languages, 2021, 5(ICFP): 1-30.
35. Ramana Kumar, Magnus O. Myreen, Michael Norrish, and Scott Owens. *CakeML: A verified implementation of ML*. In Principles of Programming Languages (POPL), pages 179--191. ACM Press, January 2014.
36. Barthe G, Demange D, Pichardie D. *Formal verification of an SSA-based middle-end for CompCert*[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 2014, 36(1): 1-35.
37. Plotkin G D. *Call­by­name, call­by­value and the λ­calculus*[J]. Theoretical computer science, 1975, 1(2): 125­159.
38. Danvy O, Millikin K, Nielsen L R. *On one­pass CPS transformations*[J]. BRICS Report Series, 2007, 14(6).
39. **课题研究目标、主要研究内容和拟解决的关键问题。 Research objectives, main contents and key issues to be solved.**

2.1 研究目标

探讨如何将关键的函数式编译过程和编译器基础设施中常用的静态单赋值形式通过形式化定义联系起来，并验证该编译过程的正确性。

2.2 主要研究内容

1. CPS转换、CPS到SSA转换算法设计
2. 将编译器验证方法应用在该编译过程中（模拟）。
3. 将目标SSA语言与成熟的经验证的编译器后端连接起来

2.3 拟解决的关键问题

两种程序结构不同的语言如何证明转换算法是对的（语义保存 是否等价 怎么证明等价）

实际可信编译器是否可以连接

1. CPS转换优化减少冗余延续变量的引入。
2. Kelsey的转换算法有一些不足，需要改进，并适配具体的CPS及SSA语言的特定要求。
3. 选择什么样的SSA语言？为了使函数式编译器前端和经验证的编译器后端连接起来，拟选择CompCertSSA。

CompCertSSA和普通SSA程序或者LLVM IR相比有一些限制。

如何将CPS程序的状态与CompCertSSA程序的状态对应起来。

1. **拟采取的研究方法、****研究方案及其可行性分析。Research methods and research scheme to be adopted and feasibility analysis.**

3.1 拟采取的研究方法

主流的编译器验证方法：模拟。

3.2 可行性分析

小步语义是否能描述程序语义

怎么表示函数式程序local nameless…

对于一个编译过程，形式化定义了源程序和目标程序的操作语义，就可以定义它们的匹配状态，并通过模拟的方式证明语义保存。

CompCertSSA中的小步操作语义介绍。

设计函数式语言小步操作语义的经验。

1. **课题的创新点 Novelties of the proposed topic.**

将函数式编译器与主流的基于SSA的编译器中间语言连接起来，并进行形式化验证。

之前的工作。。。

1. **计划进度、预期成果 Research schedule, and expected outcomes**
2. **与本课题有关的工作积累、****已有的研究工作成绩。Prior experience and accomplished achievements related to the proposed topic.**

做了什么 针对Vellvm

**本人承诺：开题报告中的内容真实无误，若有不实，愿承担相应的责任和后果。****I hereby declare and confirm that the details provided in this Form are valid and accurate.** **If anything untruthful found, I will bear the corresponding liabilities and consequences.**

**学生签字/Signature of Student： 日期/Date：** 2022-11-24