**开 题 报 告 （该表格由学生独立完成)**

|  |
| --- |
| 1.课题的作用、意义，课题的介绍，国内外的研究现状  当今世界计算机技术飞速发展，各种软件让人眼花缭乱，这得意于程序设计语言的发展。常用的程序设计语言如 C 语言、C++语言、Java 语言等，用这些高级语言编写程序极大提高了软件开发人员的效率。但要使这些代码能真正被计算机识别并执行，需要将它们转化成计算机指令，完成这一工作的便是编译器。Intel公司的 David Kuck 院士曾经将编译器誉为“计算机科学与技术的皇后”，它是应用与系统之间的桥梁。编译器将由高级语言编写的程序翻译成二进制代码（或其他目标语言），并在特定平台上运行，因此程序语言的发展，又很大程度上依赖编译技术的发展。  但相比于其他自然科学，计算机科学算是较新的科学体系，还有很多技术领域有待学者进一步探索。编译技术是系统软件的设计学科，作为计算机技术发展的基石，学习编译器设计的目的不仅在于其本身的理论和技术，其解决问题的思路与方法更值得学者学习借鉴。  自上个世纪50年代以来，编译器的相关研究一直是计算机科学领域的一个活跃主题。从目标机器的角度来说，目标机器处理器从单核到多核，其体系结构在不断的发生变化；从程序语言的角度来说，高级程序语言在不断的丰富及扩展。  1954 年，John Backus在 IBM 发明了FORTRAN。它是第一个广泛使用的具有功能实现的高级通用编程语言，而不仅仅是纸上的设计。1957年，他领导的研究小组完成了 FORTRAN 编译器，该编译器被普遍认为是引入了第一个明确完整的编译器。1958年，John McCarthy 发明了Lisp，这是世界上第一门动态类型的函数式编程语言。六十年代的Simula是第一门支持面向对象编程的语言，其后继Smalltalk，一门纯面向对象的语言，在七十年代被发明。1969年至1973年，C被发明，用作Unix操作系统的系统编程语言，流行至今。Prolog，第一门逻辑编程语言，在1972年被发明。1978年，ML在Lisp之上建立了一个多态类型系统，开创了静态类型的函数式编程语言。他们都产生了大量的后代，绝大多数现代编程语言都可以追溯回到这些语言。  作为最早的编程语言之一，Lisp开创了计算机科学的许多理念，比如树形数据结构，自动内存管理，动态类型，高阶函数，递归，自举，REPL等等。但函数式编程语言作为高度抽象的高级编程语言，与现代计算机的指令之间存在巨大的差别。而传统的编译器课程只重点讲授教编译器的某几个阶段，如解析、语义分析和寄存器分配等。在国内，典型的编译器课程将重点放在前端部分，稍完整一些的则是实现命令式编程语言如C的一个子集。这种方法的问题是很难理解整个编译器是如何结合在一起的，为什么每个阶段是那样设计的，以及那些更具表达力的语言成分究竟是如何实现的。本课题希望实现一个简单的Lisp方言，将其编译到x86汇编语言，以此来学习并探索编译技术。  Scheme是一门极简主义Lisp方言，采用了词法作用域，实现了尾递归优化，支持first-class continuations等。1978年，Guy Steele[1] 实现了第一个Scheme语言编译器，该编译器也是第一个使用CPS作为IR的编译器。1984年，Kent Dybvig创造了第一个商用Scheme编译器，Chez Scheme，它的编译速度很快，生成的目标代码也非常高效。1992年，ANDREW APPEL[3] 在其著作中描述了Standard ML of New Jersey的实现。1994年，Christian Queinnec[5] 在他的书中全面地介绍了Lisp语言家族的语义和实现，给出了11个解释器和2个目标语言分别为字节码和C语言的编译器。2004年，Kent Dybvig等人[6] 提出了nanopass框架，它从印第安纳大学的编译器课程中演化而来。2006年，Dybvig[7] 做了题为The development of Chez Scheme的汇报。2012年，Andrew Keep[8] 在其论文中描述了nanopass框架对商用编译器Chez Scheme的改造，论证了该框架的效率并不显著低于传统编译器结构。2021年，Jeremy Siek[9] 介绍了编译一个简单的函数式语言到机器语言的完整过程。  2.完成任务的可能思路和方案  本课题预期实现一个简单的、静态类型的、采用词法作用域的函数编程语言，不包括macro、continuation、patter matching等更加高级的功能。  本课题将采用nanopass的方法，但不使用nanopass框架。预计的重点参考资料为2021年Jeremy Siek[9] 发表的Essentials of Compilation以及英特尔的x86-64汇编语言手册，使用的编程语言为Racket。  Nanopass即为编译过程中的每一趟（pass）仅完成非常少量的工作，例如只消除one-armed if。传统的编译器通常由少量的pass构成，在每一个pass完成尽可能多的工作。相比之下，nanopass的编译器更加容易调试，也更加灵活，更容易加入额外的优化。  课题预计将采用自顶向下的方法，先实现一个极小的语言的编译器，然后不断地增加pass，完善语言功能。  3.需要的主要仪器和设备  一台x86-64架构的PC  4.主要参考文献   1. Steele Jr GL. Rabbit: A compiler for Scheme. 1978. 2. Peyton Jones SL. The implementation of functional programming languages. Prentice-Hall, Inc., 1987 May 1. 3. Appel AW. Compiling with continuations. Cambridge university press, 1992. 4. Waddell O, Dybvig RK. Fast and effective procedure inlining. InInternational Static Analysis Symposium 1997 Sep 8 (pp. 35-52). Springer, Berlin, Heidelberg. 5. Queinnec C. Lisp in small pieces. Cambridge University Press, 2003. 6. Sarkar D, Waddell O, Dybvig RK. EDUCATIONAL PEARL: A Nanopass framework for compiler education. Journal of Functional Programming. 2005 Sep, 15(5):653-67. 7. Dybvig RK. The development of chez scheme. ACM SIGPLAN Notices. 2006 Sep 16, 41(9):1-2. 8. Keep AW, Dybvig RK. A nanopass framework for commercial compiler development. In Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming, 2013 Sep 25 (pp. 343-350). 9. Siek JG, Factora C, Kuhlenschmidt A, Newton RR, Scott R, Swords C, Vitousek MM, Vollmer M, Tolmach A. Essentials of Compilation. 2021. |
| **指导教师评语：**（建议填写内容：对学生提出的方案给出评语，明确是否同意开题，提出学生完成上述任务的建议、注意事项等）      **指导教师签名：**  **20 年 月 日** |