北京航空航天大学计算机学院

硕士学位论文中期检查报告

论文题目：安全 C 编译器的构建和形式验证方法的研究与实现

专　　业：软件工程

研究方向：软件形式建模与验证

研 究 生：陈志伟

学　　号：SY1406108

指导教师：马殿富 教授

北京航空航天大学计算机学院

2016 年 08 月 31 日

目　　录

安全 C 编译器的构建和形式验证方法的研究与实现

1 论文工作计划

1.1 论文选题

随着计算机应用技术的飞速发展，计算机软件已经在航空航天领域中得到了广泛的应用，现代飞机几乎所有重要功能系统都与机载软件密切相关。安全攸关软件，如航空机载软件，作为各类安全关键系统的重要构成部分，其内部结构越来越复杂、应用环境越来越开放，这些因素使得人们更加关注其安全可靠性。因此，对安全攸关软件尤其是大型客机机载软件进行安全性分析、设计以及验证变得尤为重要。

目前航空领域中主要采用的验证标准是美国航空无线电委员会（RTCA）于 1992 年 12 月发布的航空适航认证标准体系 DO-178B[1]《机载系统和设备认证中的软件要求》标准。DO-178B 标准体系规定了软件开发过程中的各阶段软件制品所要达到的安全目标，对机载软件系统的安全性提出了严格的要求，但未规定针对特定的安全认证目标软件开发方所应提供的安全证据以及在软件开发过程中安全证据的技术和方法。RTCA 于 2012 年又发布了 DO-178C[2]。DO-178C 对 DO-178B 的补充有四个方面：软件工具验证、基于模型的开发和验证、面向对象编程和形式化方法。在软件开发新技术日新月异的今天，这些补充和修订很好的适应了当前安全攸关软件的开发过程。

编译器作为安全攸关软件开发过程中的关键工具，是实现软件从设计到在硬件上运行的桥梁，如何确保编译器编译过程的正确性是进行软件开发所面临的重要难题。传统检测编译错误的方法是进行大量的测试，但是测试只能证明软件是有错的，不能证明软件是没有错误的。近年来，形式化验证方法在编译器的编译

过程正确性验证[3~4]中得到了持续的关注。形式化验证方法基于严格的数学理论，将软件系统和性质都用逻辑方法来规约，通过基于公理和推理规则组成的形式系统，以如同数学中定理证明的方法来对软件系统进行证明。

实践中在安全攸关系统中使用 C 语言时，必须对语言的使用加以限制，避免那些确实可以产生问题或编译器支持的不完善的地方，直到它是可以安全应用的。C 安全子集严格要求了编译器的成熟度及稳定性，编译器必须忠实地反映源语言的代码结构和语义，以方便编译前后的代码审查、比较和追踪，确保编译后代码的安全可靠。

1.2 论文研究目标

本课题的研究目标是构建一个带有形式验证功能的编译工具。该工具不仅能完成基本的编译功能，如词法分析、语法分析等，还可以检查源代码是否符合安全 C 标准；能够正确的生成目标代码，使用基于语义的形式验证方法来保证编译过程的正确性；能够实时反馈编译和验证过程的信息；能够从源代码追溯到目标代码，实现编译过程的完整性、一致性和准确性的需求。

1.3 论文主要研究内容

为了实现上述研究目标，本文拟进行如下几个方面的研究：

(1) 研究如何在编译阶段，即词法分析和语法分析等中加入对安全 C 约束规则的检验过程，使得不符合安全 C 标准的源代码在初始阶段就能被识别出，同时需要结合实际实现对安全 C 子集做出一定的强制规定。

(2) 研究一种基于语义的形式验证方法验证编译过程是否正确。基于形式文法和自动机的相关理论，可以把对于源程序编译过程正确性的证明，转化为对源程序中包含的文法单元语义的一致性证明。通过设计命题映射算法把文法单元对应的目标码模式转化为命题，又基于一阶逻辑的公理系统，设计命题自动推理算法，从公理系统中事先给定的公理（如，目标码模式命题）出发，根据推理规则推导出一系列新命题，并作为前提加入到之后的证明过程中。比较最终推导出来的证明序列与前置条件的语义是否一致，从而完成整个证

明过程。

(3) 针对 A 级软件开发中源代码和目标代码的可追踪性需求，设计一种方法实现源代码中的每一个语句与汇编代码相应片段的对应。

(4) 基于以上的程序编译验证工具的设计与实现。编译验证工具要能对输入的源代码自动识别出不同的文法单元，对于普通的运算、赋值等语句由于其语义较简单，只需要保证其语法正确；对于循环和选择语句不仅要保证其语法正确，还要使用证明工具保证其语义的一致性。最后生成目标代码和设计一种整个过程的记录方法。

2 已经完成的工作

2.1 安全 C 子集

安全 C 子集将 MISRA-C[5]与航天型号软件的特点相结合，重新定义了一系列 C 语言软件的编程准则，为安全相关领域的 C 语言软件提供了相应的安全语言规范和编译要求。

MISRA-C，汽车制造业嵌入式 C 编码标准，从 MISRA-C:2004 开始其应用范围扩大到其他高安全性系统。在 MISRA-C:2004 中，共有强制规则 121 条，推荐规则 20 条，并删除了 15 条旧规则。任何符合 MISRA-C:2004 编程规范的代码都应该严格的遵循 121 条强制规则的要求，并应该在条件允许的情况下尽可能符合 20 条推荐规则。MISRA-C:2004 认为 C 程序设计中存在的风险可能由 5 个方面造成：程序员的失误、程序员对语言的误解、程序员对编译器的误解、编译器的错误和运行出错(runtime errors)。MISRA-C 编程规范是一个很好的典范。它始于汽车工程师和软件工程师经验的总结，然后逐渐发展成为一种对整个高安全性领域都有指导意义的规范，对于推动整个高安全性领域的正规化发展， MISRA-C 无疑有着重要意义。

采用 MISRA-C:2004 规范也会对程序有负面影响，比如可能会影响代码量、执行效率和程序可读性等，所以实际中也需要结合不同领域的软件的特点对 MISRA-C 进行限制。在我们的实践过程中，为了设计与实现工具的方便和一些特殊的需求，需要为安全 C 子集加入一些自定义的规则，如为了实现 A 级软件的可追踪性需求，强制要求所有的循环和选择语句必须使用大括号。这些规则的加入不仅不会违反 C 语言的语法，还会让安全 C 子集更安全。

2.2 形式验证方法

2.2.1 C 文法单元和语义

在计算机科学中， 抽象语法树 （abstract syntax tree 或者缩写为 AST），或者语法树（syntax tree），是源代码的抽象语法结构的树状表现形式，这里特指编程语言的源代码。树上的每个节点都表示源代码中的一种结构。简单来说就是将一段代码抽象成一个有特定节点类型的树，以便可以用过变换树，来实现代码的转换。抽象语法树中包含了源程序的所有语义信息，因而源程序与其对应的抽象语法树本质上是等价的，于是对源程序的验证可以转换为对抽象语法树的验证。又由树的定义可知：每棵树都是由其子树递归定义的，所以可以通过分别对语法子树的验证来完成对整个语法树的验证。

每棵语法子树都是使用下推自动机从源代码中识别出的一种语法结构，安全 C 子集中有多种形式文法，而每种形式文法对应的一种下推自动机，因此在源代码中就存在着多种语法结构,这些语法结构的 C 语言表达形式就是 C 文法单元。源程序的编译过程正确性验证可以等价为对每个 C 文法单元的验证，可以通过验证编译前后每个 C 文法单元和对应的目标代码模式的语义是否保持一致来实现。

为了获得每个 C 文法单元的语义，本文引入了语境的概念，根据语境可以定义出的文法单元的语义。语境表示待证明序列中每一个证明项所在的环境和上下文，其中包括函数局部变量、 全局变量以及上下文等。下表 1 给出了部分 C 文法单元和其对应的语义。

表 1　C 文法单元和语义

语句	C 文法单元	C 文法单元语义

<if-statement>	if (<LOG-EXP>)
	{
	　<STA-LIST_1>
	}
	else
	{
	　<STA-LIST_2>
	}
		σ(<LOG-EXP>) -> σ(<STA-LIST_1>)
		~σ(<LOG-EXP>) -> σ(<STA-LIST_2>)

<while-statement>	while (<LOG-EXP>)	{σ(<LOG-EXP>) -> σ(<STA-LIST>)} ** n

	{
	<STA-LIST>
	}
		~σ(<LOG-EXP>) -> skip

<do-while-statement>	do { <STA-LIST> } while (<LOG-EXP>);	σ(<STA-LIST>) {σ(<LOG-EXP>) -> σ(<STA-LIST>)} ** n ~σ(<LOG-EXP>) -> skip

<for-statement>	for(<ASS-EXP_1>;
	<LOG-EXP>;
	<ASS-EXP_2>)
	{
	<STA-LIST>
	}
		σ(<ASS-EXP_1>)
		{σ(<LOG-EXP>) -> σ(<STA-LIST>);
		σ(<ASS-EXP_1>)} ** n
		~σ(<LOG-EXP>) -> skip

表中，σ 符号代表着取值的过程，σ(<LOG-EXP>)表示获得逻辑表达式的值，按照安全 C 子集规范，逻辑表达式的值只能为 0 和 1。<STA-LIST>表示语句块，可以包括表达式语句、条件选择语句等，一般把其交给识别语句块的下推自动机进行递归处理。<ASS-EXP>表示赋值语句，其取值后的返回值就为表达式的值。“{..} ** n” 代表着循环执行大括号内的语句，用来定义循环语句的语义。skip 表示直接跳转到下一条语句进行执行，在 32 位的 Power PC 指令集[13]下，定义 skip 等于 σ(PC = PC + 4)，PC 表示程序计数器。

2.2.2 目标码模式和命题

目标码模式是通过 GCC 编译器编译在一定语境下的 C 文法单元得到目标码序列，消除掉语境对目标码序列的影响而获得的目标码序列的一般化（Generalize）表示形式。对编译器的形式化验证，最终需要转化为 C 文法单元的语义和目标码模式的语义一致性验证。

程序的形式化证明需要特定的公理系统作为基础。公理系统（axiomatic system）就是把一个科学理论公理化，用公理方法研究它，每一科学理论都是由一系列的概念和命题组成的体系。由目标码模式和下文中的 2.3.1 命题映射算法，可以得到每个目标码模式的命题。表 2 中给出了条件选择语句（<if-statement>）和循环语句（<while-statement>）目标码模式和目标码模式命题。

表 2 目标码模式和目标码模式命题

语句	目标码模式	目标码模式命题
<if-statement>	<LOG-EXP> cmpi 7,0,0,0 beq 7,.L1 <STA-LIST_1> b .L2 .L1: <STA-LIST_2> .L2:	P1: GPR[0] = <LOG-EXP> P2: (GPR[0] < 0 -> CR[7] = b100) || (GPR[0] > 0 -> CR[7] = b010) || (GPR[0] == 0 -> CR[7] = b001) P3: (CR[7] == b100 -> PC = PC + 4) || (CR[7] == b010 -> PC = PC + 4) || (CR[7] == b001 -> PC = PC + @.L1) P4= <STA-LIST_1> P5: PC = PC + @.L2 P6: .L1: P7: <STA-LIST_2> P8: .L2:
<while-statement>	b .L2 .L1: <STA-LIST> .L2: <LOG-EXP> cmpi 7,0,0,0 bne 7,.L1	P1: PC = PC + @.L2 P2: .L1: P3: <STA-LIST> P4: .L2: P5: GPR[0] = <LOG-EXP> P6: (GPR[0] < 0 -> CR[7] = b100) || (GPR[0] > 0 -> CR[7] = b010) || (GPR[0] == 0 -> CR[7] = b001) P7: (CR[7] == b100 -> PC = PC + @.L1) || (CR[7] == b010 -> PC = PC + @.L1) || (CR[7] == b001 -> PC = PC + 4)

2.2.3 推理证明

本文提出的形式化验证方法是基于一阶逻辑的公理系统，从公理系统中事先给定的公理（如，目标码模式命题）出发，根据推理规则推导出一系列新命题，并作为前提加入到之后的证明过程中。由于证明序列中的每一项都是前提、公理或者定理，又因为一阶逻辑的公理系统是可靠的，所以证明序列中的每一项一定是正确的，从而最终推导出来证明序列一定是正确的。

MP(Modus Ponens)规则，被称为分离论证或分离规则（Rule of Detachment）是一阶逻辑的公理系统中的最基本的推理规则。MP 规则可表示为：

$$p \rightarrow q, p \vdash q$$

其含义是: 如果 p 成立, 那么 q 成立, 又 p 成立, 因此 q 成立。分离规则由三个陈述（或命题，或语句）组成：第一个陈述是一个条件陈述，即 p 蕴涵 q；第二个陈述是 p，即条件陈述的前提为真。从前两个陈述就能逻辑上推出 q，即条件陈述的结论也必定为真。

CI(Conjunction Introduction)规则，被称为合取引入或组合规则。CI 规则可表示为：

$$p, q \vdash p \wedge q$$

其含义是：若 p, q 成立，则 p∧q 成立。合取规则主要用来把多个为真的命题转化为单一的命题。

在实际的验证过程中，对于表达式文法单元、条件选择文法单元等的目标码模式命题，由于它们不含有循环结构，运用 MP 规则和一阶逻辑的公理系统中的公理集、定理集等，可以很方便的完成命题的推理证明。但是，对于循环结构，如<while-statement>的目标码模式命题进行直接推理后，得到的目标码模式的语义与 C 文法单元的语义差异较大，无法直接证明两者的语义是一致的，所以本文引入了限定数学归纳法对循环结构目标码模式命题进行证明。

限定数学归纳法的逻辑基础是自然数公理，也称皮亚诺公理。一般数学归纳法的逻辑表达式为 P(0)∧(∀n)(P(n)→P(s(n))→(∀n)P(n)，限定数学归纳法是在一般数学归纳法的基础上，限定 n 是有穷的，即对于循环结构程序，循环是可终止的，终止条件由人给出。

2.3 形式验证算法

2.3.1 命题映射算法

命题映射算法的作用是把目标码模式转换为命题的形式，以方便进行后续的推理证明。算法中需要把 Power PC 指令集中每条指令对应的指称语义作为专用公理输入，逐条遍历输入的目标码模式，把每条目标码转化为对应的指称语义的形式，最终把目标码模式的指称语义表示成命题集的形式输出。命题映射算法的伪代码如表 3 所示。

表 3 命题映射算法

Algorithm 1 Proposition Mapping

```
Input:    ObjectCodePatternSet
Output: PropositionSet
1: axiomSet = loadAxiom(denotationalSemanticsFileName)
2: for each line in ObjectCodePatternSet do
3:       lines = line.split(regex)
4:       lines = filterOtherCharacter(lines)
5:       if lines.length == 0 then
6:             continue
7:       else if lines.length == 1 then
8:             add new Proposition(lines) to PropositionSet
9:       else
10:            paras = generateParas(lines)
11:            seman = generateSemantic (lines, paras, axiomSet)
12:            add new Proposition (lines, paras, seman) to PropositionSet
13:      end if
14: end for
```

2.3.2 自动推理算法

自动推理算法是本文提出的形式验证方法的核心。算法以 2.3.1 中命题映射算法输出的命题集和一阶逻辑的公理集为前提，根据推理规则推导出一系列新命题，把这些新的命题加入前提中进行后续的证明，最终可以构造出证明序列并得到结论。对于不包含循环的目标码模式命题集可以直接对推导出的证明序列进行取值（σ）操作，从而得到目标码模式命题的语义，与 C 文法单元的语义比较，可以直接判断出二者的语义是否保持一致。对于包含循环的目标码模式命题集，直接对推导出的证明序列进行取值操作却无法完成语义一致性检验过程，需要引入 3.3 中的循环交互证明算法才能完成整个过程。命题自动推理算法的伪代码如表 4 所示。

表 4 自动推理算法

```
Algorithm 2 Automatic Derivation
Input:    PropositionSet
Output: SemantemeSet
1: for each p in PropositionSet do
2:       if newPropositionSet.size() == 0 then
3:             add p to newPropositionSet
4:       else
5:             for each q in newPropositionSet do
6:                   newProposition = applyDerivationRuleToTwoPropositions (p, q)
7:                   if newProposition != null then
8:                         add newProposition to newPropositionSet
9:                   end if
10:                  if q's content is empty then
11:                        remove q from newPropositionSet
12:                  end if
13:            end for
14:      end if
15: end for
16: for each p in newPropositionSet do
```

```
17:       s = obtainSemantemeFromProposition (p)
18:       add s to SemantemeSet
19: end for
```

2.3.3 循环交互证明算法

循环交互证明算法的理论基础是限定数学归纳法。算法首先引导用户输入 n 为 1 时 C 文法单元的语义，然后按照循环条件分别为真或假时，分别构造新的命题加入到 2.3.1 命题映射算法输出的命题集的一个 copy 中，调用 2.3.2 中的自动推理算法对 copy 命题集进行推理，从而得到此时目标码模式命题的语义。

把目标码模式命题的语义和用户输入的语义对比。若二者语义不一致，则直接给出形式验证过程出现错误的提醒并退出；若一致，提醒用户输入当 n 为 N 时 C 文法单元的语义，在此基础上再次对源语义集进行一次推理。通过 CI 规则，把推理出的语义结果加入到 n 为 N 时 C 文法单元的语义中，得到 n 为 N + 1 时目标代码模式的语义。把用户输入的 n 为 N 时 C 文法单元的语义中的 N 使用 N + 1 替换，比较程序推理出的目标码模式语义和用户输入 C 文法单元的语义在 n 为 N + 1 是否一致，返回判断结果。循环交互证明算法的伪代码如表 5 所示。

表 5 循环交互证明算法

```
Algorithm 3    Loop Interactive Proving
Input: PropositionSet
Output: Flag
1: Flag = true
2: for stage ← FIRST, LAST do
3:       userSemantemeSet = ReadUserInputSemanteme ()
4:       copy PropositionSet to cpyPropositionSet
5:       add true loop condition Propositionto cpyPropositionSet
6:       trueSemantemeSet = AutomaticDerivationAlgorithm(cpyPropositionSet)
7:       copy PropositionSet to cpyPropositionSet
8:       add false loop condition Propositionto to cpyPropositionSet
9:       falseSemantemeSet = AutomaticDerivationAlgorithm(cpyPropositionSet)
10:      semantemeSet = trueSemantemeSet || falseSemantemeSet
11:      if stage == LAST then
12:            semantemeSet = CI (semantemeSet, userSemantemeSet)
13:            update n from N to (N + 1) in userSemantemeSet
14:      end if
15:      if semantemeSet != userSemantemeSet then
16:            Flag = false
17:            return Flag
18:      end if
19: end for
```

2.4 编译验证工具的设计与实现

本课题要实现一个编译验证工具，其系统框架架构如图 1 所示：

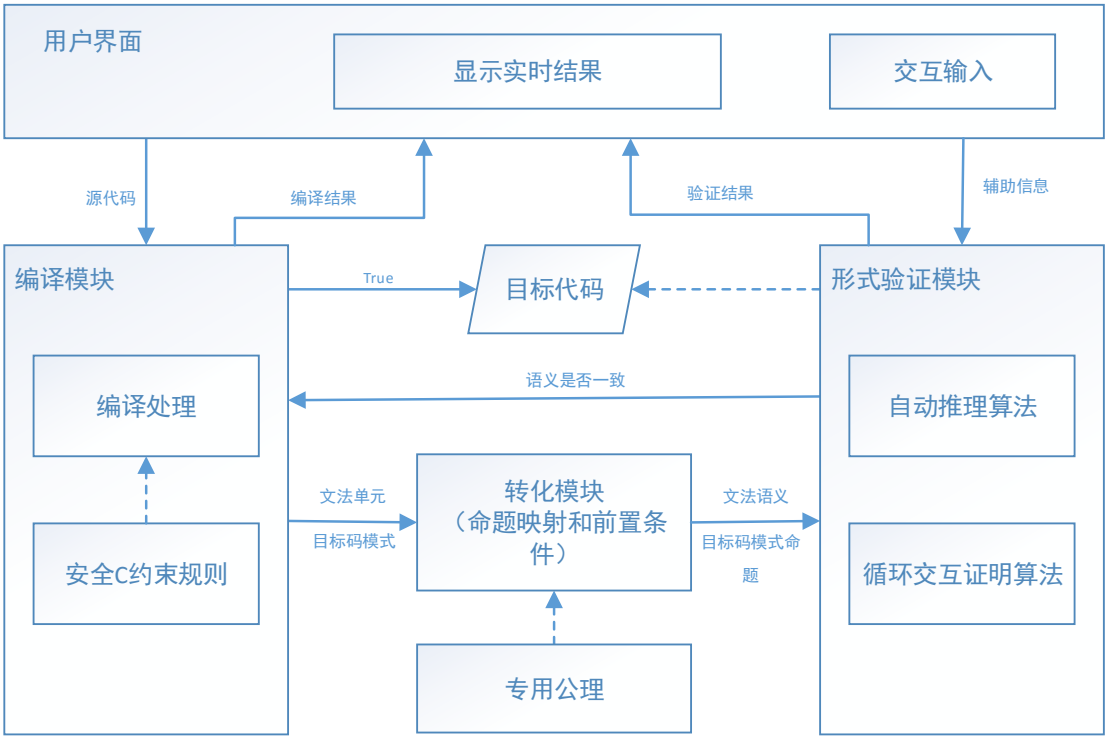

图 1 系统框架

- 编译模块完成基本的编译处理过程和安全 C 约束规则的检验，同时根据形式验证模块给出的验证结果决定是否生成目标代码；
- 转化模块根据识别出的文法单元分别求出每个文法单元和和对应的目标码模式的语义，并且把文法单元的语义作为后续证明的前置条件；
- 专用公理集提供不同指令集（如 MIPS、PowerPC）的指称语义；
- 形式验证模块实现对文法单元语义和目标代码段语义一致性的确认，对于普通的选择语句可以直接使用自动推理算法来进行验证，但对于循环语句的验证需要额外使用循环交互证明算法；
- 用户界面辅助用户进行形式化验证。

3 关键技术或难点

3.1 指称语义

指称语义是采用形式系统方法，用相应的数学对象（如 set， function 等）对一个即定形式语言的语义进行注释的学问。指称语义还可以解释为：存在着两个域，一个是语法域，在语法域中定义了一个形式语言系统；另外一个是数学域（或称之为已知语义的形式系统）。为了满足形式验证的需求，需要用指称语义的形式来分别表示 C 文法单元和目标代码模式的语义。

文中以一种形式规约的方式给出了 C 文法单元的语义，并以之作为后续推导的前置条件。然后，根据所使用指令集的每条指令的操作语义，为指令建模并得到对应的指称语义，把指令集对应的指称语义集作为专用公理输入到验证部分，通过验证部分的推导即可得到目标码模式的语义。

3.2 语义一致

编译器的任务是将源代码正确的编译为目标代码，而验证源代码与编译生成的目标代码之间是否具有一致的语义是判断编译过程是否正确的有效方法。因此，课题中需要构造一个证明器（Prover）来证明源代码的文法单元和目标码模式的语义一致。

证明器的原理是基于一阶逻辑的公理系统，推理规则使用的是分离规则和组合规则，主要使用了自动推理算法模拟人对于命题的推理过程，由此得到目标码模式的语义并输出。最后，把推导出的结果作为后置条件和文法单元的语义这个前置条件做比较，看二者是否一致，从而完成语义一致性的验证。

3.3 限定数学归纳法

限定数学归纳法的逻辑基础是自然数公理，也称皮亚诺公理。限定数学归纳法是在一般数学归纳法的基础上，限定 n 是有穷的，即对于循环结构程序，循环是可终止的。循环的可终止性由人来证明，终止条件由人给出。

传统的程序验证方法对于循环语句的证明依赖于循环不变式，但是循环不变式的构造没有一般方法，且往往构造难度比较大，这便使得验证难度增大。本论文将限定数学归纳法引入到循环的证明中，避免了循环不变式的构造，降低了验证难度。

4 下一阶段工作计划

4.1 存在的问题和解决思路

目前已经完成了整个编译验证工具的设计和大体实现，但主要的问题如何显示整个编译和验证过程。特别是源代码中识别出的需要进行形式验证的语句,若输出整个证明过程的话，最后的输出结果会很混乱。初步想到的解决方案是引入 log4j 日志系统来详细记录，前台界面简洁输出，二者结合起来可能会比较好。

4.2 尚未完成的工作

工作内容		已完成	进行中	未展开
理论分析	安全 C 子集完善	√		
	C 文法单元的定义和目标码模式命题的理论推导	√		
算法实现	命题映射算法的实现	√		
	自动推理算法的实现	√		
	基于限定数学归纳法的循环交互证明算法的实现	√		
工具实现	系统设计	√		
	编译过程的安全 C 子集检查	√		
	非循环结构程序的自动验证	√		
	循环结构程序的交互证明	√		
	目标码生成和证明过程记录的生成		√（80%）	
	用户界面完善		√	
编译验证工具应用测试			√	
论文发表		√		

4.3 下一阶段计划

时间	计划
2016 年 9 月~10 月	进一部完善编译验证工具，完成界面设计及工具应用的测试

2016 年 10 月~11 月	毕业论文撰写
2016 年 12 月	毕业答辩事宜

5 主要参考文献

[1] RTCA Inc., "RTCA/DO-178B: Software Considerations in Airborne Systems and Equipment Certification", Washington D.C.: RTCA Inc., 1992.

[2] RTCA Inc., "RTCA/DO-178C: Software Considerations in Airborne Systems and Equipment Certification", Washington D.C.: RTCA Inc., 2011.

[3] Leroy X. Formal verification of a realistic compiler[J]. Communications of the ACM, 2009, 52(7): 107-115.

[4] Blum M, Kannan S. Designing programs that check their work[J]. Journal of the ACM (JACM), 1995, 42(1): 269-291.

[5] Motor Industry Software Reliability Association. MISRA-C: 2004: Guidelines for the Use of the C Language in Critical Systems[M]. MIRA, 2008.

[6] McCarthy J, Painter J. Correctness of a compiler for arithmetic expressions[J]. Mathematical aspects of computer science, 1967, 1.

[7] Stepney S, Whitley D, Cooper D, et al. A demonstrably correct compiler[J]. Formal Aspects of Computing, 1991, 3(1): 58-101.

[8] Gaul T, Goos G, Heberle A, et al. An Architecture for Verified Compiler Construction[C]//Joint Modular Languages Conference. 1997, 1996.

[9] Leroy X. A formally verified compiler back-end[J]. Journal of Automated Reasoning, 2009, 43(4): 363-446.

[10] Leroy X. Mechanized semantics for compiler verification[M]//Certified Programs and Proofs. Springer Berlin Heidelberg, 2012: 4-6.

[11] Yang X, Chen Y, Eide E, et al. Finding and understanding bugs in C compilers[C]//ACM SIGPLAN Notices. ACM, 2011, 46(6): 283-294.

[12] Barnett M, Leino K R M, Schulte W. The Spec# programming system: An overview[M]//Construction and analysis of safe, secure, and interoperable smart devices. Springer Berlin Heidelberg, 2004: 49-69.

[13] EREF：A Programmer’s Reference Manual for Freescale Power Architecture Processors[M].Rev.1, 2014，Freescale.