安全C编译器的构建和形式验证方法的研究与实现

北京航空航天大学计算机新技术研究所
The Institute of Advanced Computing Technology

导师：马殿富 教授
学生：陈志伟
学号：SY1406108

北京航空航天大學
BEIHANG UNIVERSITY

报告内容

- 课题背景和意义
- 国内外研究现状
- 研究目标和内容
- 关键技术
- 系统设计与实现
- 总结与展望

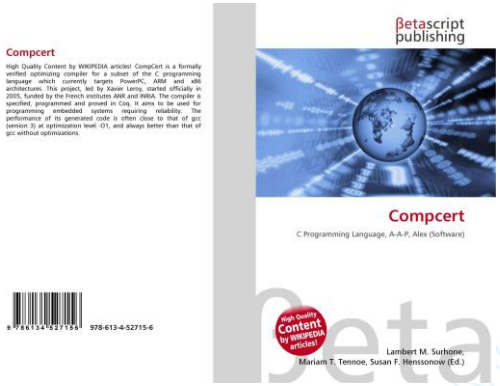
选题背景和意义

- 机载软件
  - 规模百万行级别
    - 波音787的飞控代码达到650万行
  - 软件结构复杂
  - 安全性和可靠性

- 编译器
  - 机载软件开发过程中的重要工具，负责把源代码翻译成目标代码
  - 编译器规模成倍增长
    - 2001年GCC2的代码量只有200万行左右，到了2014年GCC5的代码量增长了7倍多，达到1450万行左右[1]
  - 各大商用编译器一直以来都存在许多错误
    - https://gcc.gnu.org/bugs/
    - http://bugs.java.com/
  - 测试只能发现错误不能保证编译器是正确的
    - GCC4.7通过的torture测试集包含2853个C源程序测试用例[2]
    - GCC4.7中有超过118 bugs在下一个版本中得到修复

编译验证背景

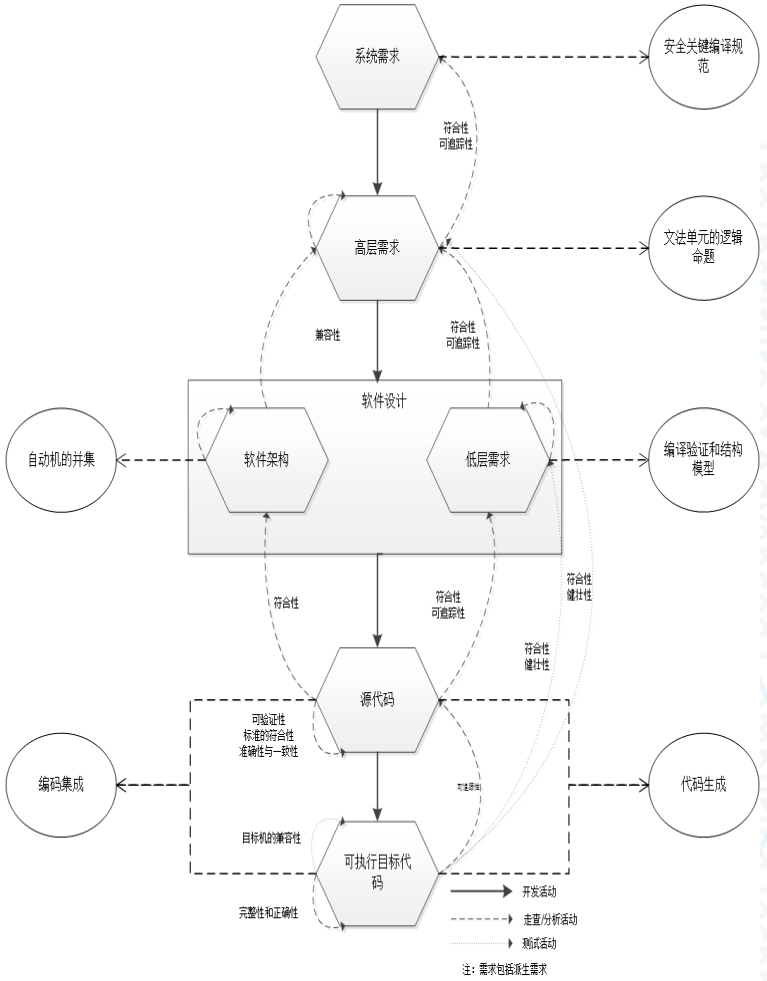
- 形式化验证方法
  - 基于严格的数学理论将软件系统和性质用**逻辑方法来规约**，通过一个逻辑公理系统来**证明软件的正确性**
  - 定理证明、模型检测和程序检验
  - 可以保证编译器的**完全正确性**
    - 最有代表性的工作是CompCert编译器项目
  - 验证过程复杂繁琐、耗费时间长
    - CompCert项目2005年~2009年[3]



- 编译验证方法
  - 验证编译器自身的正确性
    - 验证编译程序本身满足给定规范的前后置条件
  - **验证编译过程的正确性**
    - 源代码和编译后的目标代码语义一致性
  - 编译器自身的正确性间接保证编译过程的正确性
  - 难度更小

关键技术背景

- 课题来源
  - 民机专项“符合DO-178B/C的A级软件开发与认证技术研究”
- DO-178C
  - 国际公认的航空适航认证标准（2012）
  - 引入**基于逻辑证明的形式化方法**
  - 开发和验证过程的一致性和可追溯性
- MISRA-C
  - 原是汽车制造业嵌入式C编码标准
  - MISRA-C:2004应用范围扩大到其它高安全性系统
  - 与航天型号软件的特点相结合形成安全C子集
    - 实时性、高可靠性和重用性的特点
  - 安全C子集对编译器支持的C语言规范作出了一定的限制

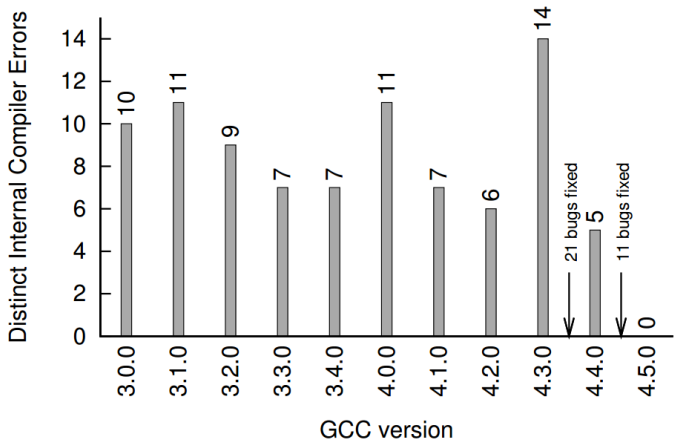


报告内容

- 课题背景和意义
- 国内外研究现状
- 研究目标和内容
- 关键技术
- 系统设计与实现
- 总结与展望

国内外研究现状

- CompCert编译器[4]
  - 一个经过形式化验证且严格保持C代码语义的编译器
  - Xavier Leroy于2009年公布了CompCert编译器的开发和验证工作
  - 基于Coq定理证明工具[5]首次完成了对一个完整且实际的编译过程正确性的形式验证
  - 整个证明过程完全形式化的且是机器自动生成的
  - CompCert支持ISO C99的一个子集尚不能完全覆盖所有的C语言元素
- Csmith[6]
  - 一个随机测试用例生成工具
  - Xuejun Yang团队于2011年使用Csmith工具对主流的11款C编译器进行测试
    - Intel CC、GCC和LLVM编译器等
  - 结果报告了325个未知的bugs
  - CompCert在其已支持的C语言子集中没有找到任何错误



中科大-耶鲁高可信软件联合研究中心

- 2008年10月中科大校长和耶鲁校长签署备忘录建立联合中心
- 联合中心旨在研究高可信软件和形式化方法的各个领域
- 主要项目：一个针对C语言子集Clike的验证编译器——CComp[7]（2010年）
  - 负责人：陈意云教授
  - 原理：携带证明代码方法[8]
  - 高层验证
    - 验证源代码
    - VCGen生成验证条件
    - 内置自动定理证明器
    - Coq自动检查证明
  - 低层验证
    - 现有的低层验证框架—SCAP[9]的变体
    - 验证汇编程序
    - 遵循Hoare式验证方法—前/后置条件
- 项目依旧在进行和完善中

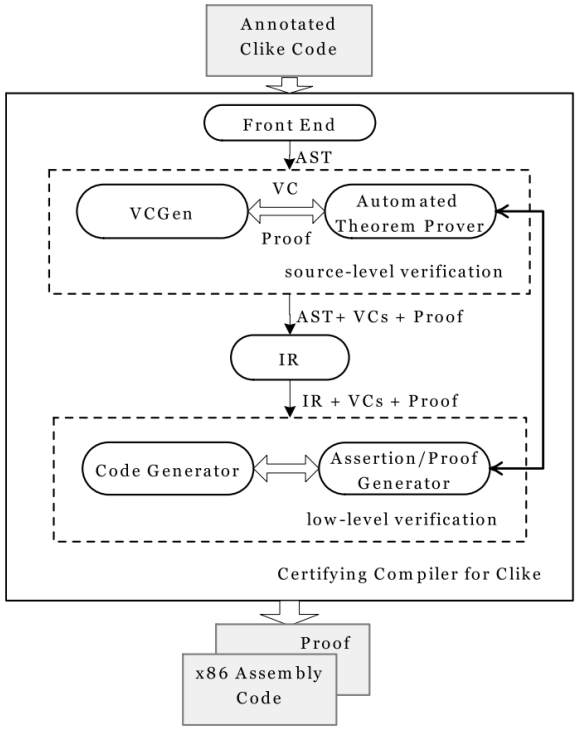


Figure 1. Structure of *CComp* Certifying Compiler

报告内容

- 课题背景和意义
- 国内外研究现状
- 研究目标和内容
- 关键技术
- 系统设计与实现
- 总结与展望

研究目标

- 提出一种基于编译语义的形式化验证方法进行编译正确性验证，并实现一个符合DO-178C规范的集编译、验证和追踪于一体的安全C编译验证工具原型。

研究内容

- 研究**一种基于编译语义的形式化验证方法**，实现从源代码到目标代码的编译过程的验证。基于形式文法和自动机的相关理论，把编译过程的正确性验证转化为文法单元和对应目标码模式的语义一致性验证。
- 研究如何在编译阶段加入对**安全C规则的检验过程**，使不符合安全C规则的源代码在初始阶段就能被识别出，并**扩展现有的编译器构建方法**使其能支持基于编译语义的形式验证方法。
- 设计**一种源代码和目标代码之间的追踪方法**，为符合DO-178C标准的A级软件的研发奠定基础。
- 设计与实现**编译验证工具原型**。

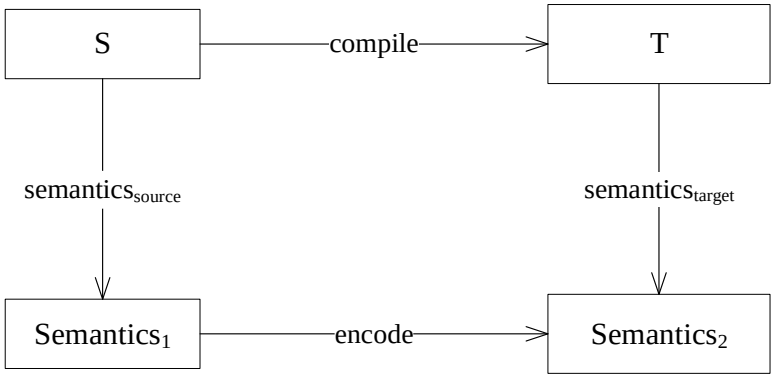
报告内容

- 课题背景和意义
- 国内外研究现状
- 研究目标和内容
- 关键技术
- 系统设计与实现
- 总结与展望

编译验证核心方法

- 编译过程正确性

定义：编译过程正确性是要保证输入的源代码和编译后的目标代码语义一致，即源代码与编译后的代码行为上要等价，编译器不能在目标代码中篡改源代码中的操作。



编译过程正确性的形式化定义可用如上图的**转换示意图**[10]表示，对编译过程正确性的验证就是证明对应的转换示意图的成立。设源代码为S，箭头表示函数映射过程，则编译过程正确性可以用如下等式来表示：

$$encode\left(semantic_{source}(S)\right) = semantics_{target}\left(compile(S)\right) \quad (1)$$

编译验证核心方法

- 编译验证方法原理

设目标代码为T，目标代码是由源代码编译得到的，故有：

$$T = compile(S) \quad (2)$$

由(1)、(2)式可以得到：

$$encode\left(semantic_{source}(S)\right) = semantics_{target}(T) \quad (3)$$

不妨设C文法单元为S_i，对应的目标码模式为T_i。根据文法单元的独立性，则有：

$$\begin{cases} S = \bigcup_{i=1}^{n} S_i \\ T = \bigcup_{i=1}^{n} T_i \end{cases} \quad (4)$$

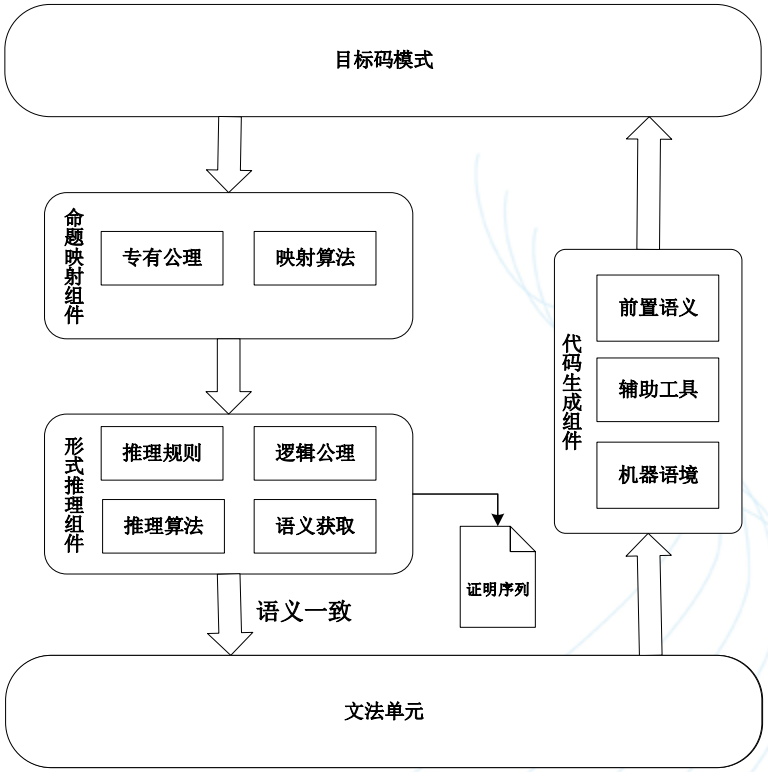
把(4)式代入(3)式，简化后有：

$$encode\left(semantic_{source}(S_i)\right) = semantics_{target}(T_i) \quad (i = 1,\ldots,n) \quad (5)$$

上述(5)式说明了编译过程正确性验证可以等价为对源代码中每个C文法单元的验证，可以通过**验证编译前后每个文法单元和对应的目标代码模式的语义的一致性**来实现。

编译验证架构

- 文法单元验证架构
  - 代码生成组件
    - 根据识别出的文法单元生成对应的目标码模式
    - 涉及目标机器的语境等
    - 文法单元的语义
  - 命题映射组件
    - 把目标码模式映射为对应的目标码模式命题
    - 专用公理
  - 形式推理组件
    - 以目标码模式命题为前提进行形式推理的过程
    - 逻辑公理系统
    - 语义一致性确认



编译验证公理系统

- 公理系统的符号定义
  - 下表定义了编译验证公理系统所涉及到的所有符号。

类型	符号	意义
逻辑联结词	¬	一元联结词，非
	->	二元联结词，蕴含
	∨	二元联结词，或
	∧	二元联结词，且
寄存器元件	GPR	表示Power PC的通用寄存器，主要用作堆栈指针、函数的第一个参数和返回值等
	CR	表示条件寄存器，用来反映某些操作的结果（比如比较指令），协助测试和分支转移指令的执行
	PC	表示程序计数器，用于存放下一条指令所在单元的地址的地方
内存	MEM	表示内存地址，与内存寻址相关
…	…	…

定义文法单元和语义

- 文法单元
  -

语句	C文法单元	C文法单元语义（前置条件）

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

- 文
  -
  -
  -
  -

定义目标码模式和命题

- 目标码模式
  -
  -
- 目
  -
  -
  -

语句（部分）	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)

编译验证公理系统

- 公理系统的推理规则
  - 推理规则是正确推理的依据，任何一条永真蕴含式都可以作为一条推理规则。下表给出本公理系统的推理规则，其中，HYP为前提或者断言， P、Q为命题。

名称	前件	后件
Rule1 CI规则（合取引入）	$\begin{cases} HYP \vdash P \\ HYP \vdash Q \end{cases}$	$HYP \vdash P \wedge Q$
Rule2 合取删除规则	$HYP \vdash P \wedge Q$	$\begin{cases} HYP \vdash P \\ HYP \vdash Q \end{cases}$
Rule3 演绎规则	$HYP, P \vdash Q$	$HYP \vdash P \to Q$
Rule4 MP规则（假言推理）	$\begin{cases} HYP \vdash P \\ HYP \vdash P \to Q \end{cases}$	$HYP \vdash Q$
Rule5 P规则（前提引入）	在推导的任何步骤上，都可以引入前提。	
Rule6 T规则（结论引用）	在推导任何步骤上所得结论都可以作为后继证明的定理。	

编译验证证明方法

- 证明方法
  - 基于命题逻辑构成的编译验证公理系统，从公理系统中事先给定的前提集（目标码模式命题）出发，根据公理系统的推理规则推导出一系列新命题，并作为定理加入到之后的证明过程中。
  - 证明过程中每一项都是前提、公理或由推理规则推导出的定理。
  - 命题逻辑公理系统是可靠且完全的，所以最终推导出来的证明结果一定是正确的。
- 证明序列
  - 证明序列是一系列证明步骤的集合，每个步骤包括公式和证据两项。
  - 公式是形式推理的命题变换过程和结果，证据表示构造这一步骤的原因。
  - 通过检查证明序列中的证据是否是公理系统的前提、定理或推理规则来判断证明过程是否正确。

编译验证证明方法

if文法单元的证明：

目标码模式命题
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:

前提集

形式推理过程

证明序列

公式	证据
S1= GPR[0] = <LOG-EXP>	P1
S2= (GPR[0] < 0 -> CR[7] = b100) || (GPR[0] > 0 -> CR[7] = b010) || (GPR[0] == 0 -> CR[7] = b001)	P2
S3= (<LOG-EXP> < 0 -> CR[7] = b100) || (<LOG-EXP> > 0 -> CR[7] = b010) || (<LOG-EXP> == 0 -> CR[7] = b001)	S1, S2, MP
S4= (CR[7] == b100 -> PC = PC + 4) || (CR[7] == b010 -> PC = PC + 4) || (CR[7] == b001 -> PC = PC + @.L1)	P3
S5= (<LOG-EXP> < 0 -> PC = PC + 4) || (<LOG-EXP> > 0 -> PC = PC + 4) || (<LOG-EXP> == 0 -> PC = PC + @.L1)	S3, S4, MP
S6= <STA-LIST_1>	P4
S7= PC = PC + @.L2	P5
S8= .L1:	P6
S9= <STA-LIST_2>	P7
S10= .L2:	P8

S11= {
(<LOG-EXP> < 0 -> PC = PC + 4) || (<LOG-EXP> > 0 -> PC = PC + 4) || (<LOG-EXP> == 0 -> PC = PC + @.L1) ∧
<STA-LIST_1> ∧
PC = PC + @.L2 ∧
.L1: ∧
<STA-LIST_2> ∧
.L2:
}
	S5, S6, S7, S8, S9, S10, CI

S12= {
(<LOG-EXP> < 0 -> <STA-LIST_1>) || (<LOG-EXP> > 0 -> <STA-LIST_1>) || (<LOG-EXP> == 0 -> <STA-LIST_2>)
}
	S11, REDUCE

S13= (<LOG-EXP> != 0 -> σ(<STA-LIST_1>) || <LOG-EXP> == 0 -> σ(<STA-LIST_2>))	S12, σ

编译验证关键算法

- 命题映射算法
  - 作用是**把目标码模式转换为命题的形式**，构造后续推理的前提集合。
  - 专用公理：PowerPC指令的指称语义
  - 算法的时间复杂复为O(n)
- 自动推理算法
  - 提出的**形式验证方法的核心**。
  - 基于构建的公理系统，使用前提集和推理规则推导出一系列新命题，把这些新的命题作为定理加入当前集合中进行后续的证明。
  - 算法的时间复杂度为$O(n^2)$
- 循环交互证明算法
  - 主要用于**含有循环结构**的文法单元的推理证明。
  - 理论基础是**限定数学归纳法**
  - 需要两次调用自动推理算法
  - 算法的时间复杂度为$O(n^2)$

编译验证关键算法

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 is not 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
```

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
```

命题映射算法

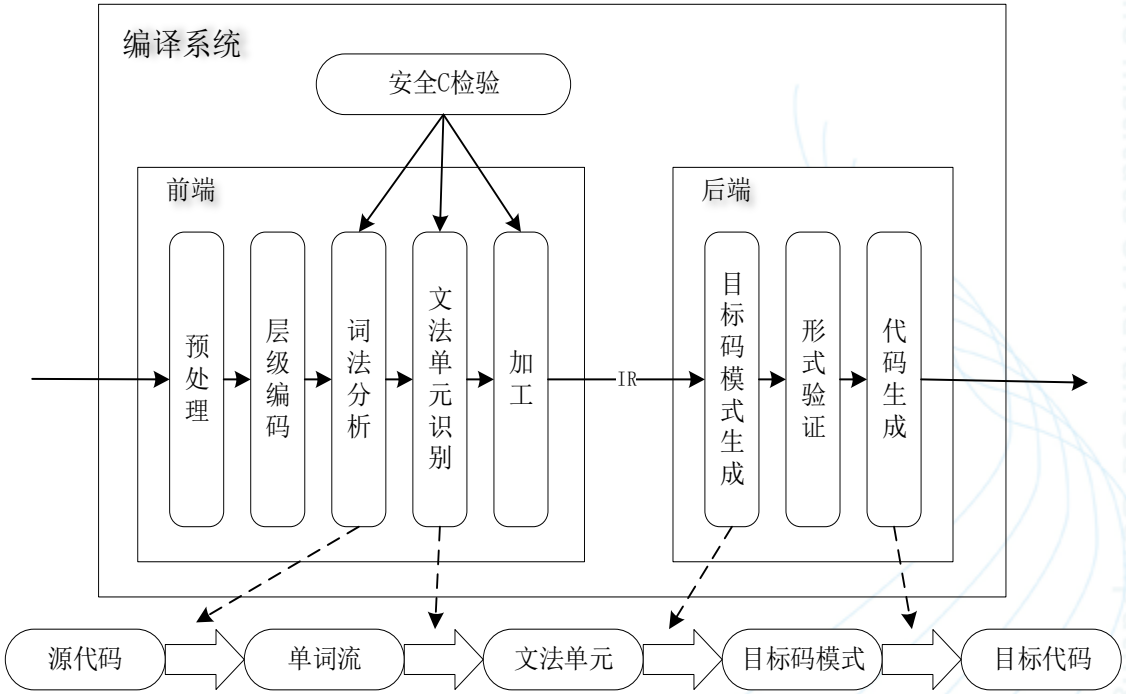
Algorithm 1 Proposition Mapping

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
```

安全C编译构建方法

- 构建方法架构
  - 上部分是编译系统的构成部分
  - 下部分为输入的数据在整个处理流程中的转化过程
  - 形式验证
    - 前文所述编译验证过程作为一个处理阶段加入到编译系统的后端实现中
  - 文法单元识别
  - 层级编码
  - 安全C检验



文法单元识别方法

- 文法单元识别主要使用的技术是下推自动机。
- 下推自动机
  - 定义：下推自动机是一个六元组 $(Q,\Sigma,\Delta,s_0,z_0,F)$，其中：
    （1）Q是状态的一个非空有穷集合。
    （2）Σ是输入字母表。
    （3）Δ是栈字母表。
    （4）s_0是起始状态。
    （5）z_0是栈起始符号。
    （6）F是终止状态集，$F \subseteq Q$。
    （7）$\delta(s,c,z)=([Js,Rs],z')$是状态映射函数。它表示从当前状态$s$出发，且栈顶符号是z，若输入的字符为c，则映射到三元偶 $([Js,Rs],z')$。
- 下推自动机算法
  - 输入：单词流、状态集和终止状态集
  - 输出：文法单元集合
  - 定义一个分析栈、一个回滚栈以及一个状态栈。
  - 分析栈中存储当前自动机已经匹配的单词；
  - 回滚栈中记录了每个阶段自动机都可能到达的状态集合，当自动机匹配失败时可以回溯到上一个状态，从而可以继续当前的匹配过程；
  - 状态栈中存储当前阶段自动机可能达到的状态。

下推自动机算法

Algorithm 5 Pushdown Automata Algorithm
Input: TokenStream, Q, F
Output: Collections<SyntaxUnit>

```
1: build AnalysStack<Token> and RollbackStack<Stack>
2: while TokenStream not eof do
3:     token ← NextToken ()
4:     state ← S0
5:     statesStack ← NextState (state, token, ε)
6:     push token to AnalysStack
7:     push statesStack to RollbackStack
8:     while RollbackStack not empty do
9:         pop the top of RollbackStack to statesStack
10:        while statesStack not empty do
11:            pop the top of statesStatck to state
12:            if state ∈ F then
13:                construct SyntaxUnit from AnalysStack
14:                add SyntaxUnit to Collections
15:                clear AnalysStack, statesStatck and RollbackStack
16:            end if
17:            if state is compound_state then
18:                (Rs, z') ← SubAutomata (state, TokenStream)
19:                if Rs ≠ Se then
20:                    push z' to AnalysStack
21:                else
22:                    ResetTokenStream ()
23:                    continue
24:                end if
25:            end if
26:            token ← NextToken ()
27:            tmpStatesStack ← NextState (state, token, top of AnalysStack)
28:            push token to AnalysStack
29:            push statesStack to RollbackStack
30:            push tmpStatesStack to RollbackStack
31:            break
32:        end while
33:        if statesStack is empty then
34:            pop the top of AnalysStack
35:        end if
36:    end while
37: end while
```

层级编码方法

- 层级编码
  - 一种把源代码按照代码的嵌套层次依次赋予对应的数字编号的方法。
  - 编译的前端实现的，只用到了代码的少量结构信息。
  - 实现从源代码到目标代码及证明序列之间的相互追踪，从而完成DO-178C标准中规定的A级软件可追踪性需求。
- 层级编码算法
  - 使用栈的方式存储当前的层级编号。
  - 用入栈和出栈来对应复合语句的进入和退出过程。
  - 基于词法分析过程实现的。
  - 算法的时间复杂复为O(n)

层级编码算法

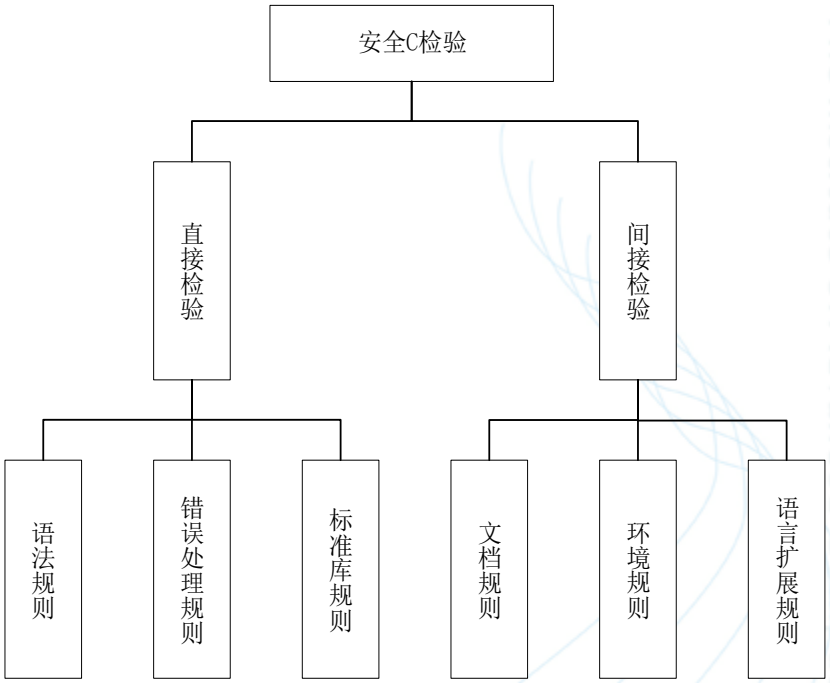
Algorithm 6 Hierarchical Coding Algorithm

Input: SourceCode
Output: TokenStream

```
1: build stack, libsList
2: push 1 to statck
3: for each line in SourceCode do
4:       if line contains '}' then
5:             remove the top of stack
6:             pop the top of stack to tmp
7:             push (tmp + 1) to stack
8:       end if
9:       if line.length ≠ 0 then
10:            tmpLibs ← solveLine (line, stack, TokenStream)
11:            add tmpLibs to libsList
12:      end if
13:      if line contains '{' then
14:            push 0 to statck
15:      end if
16:      if line.length ≠ 0 then
17:            pop the top of stack to tmp
18:            push (tmp + 1) to stack
19:      end if
20: end for
21: for each lib in libsList do
22:      solveMultipleFile (lib, stack, TokenStream)
23: end for
```

安全C检验方法

- 安全C检验
  - 为了分析源代码是否符合安全C子集
  - 安全C子集规定了系统处理的高级语言规范
  - 实现系统安全性和可靠性的重要保证
- 检验过程
  - 直接检验
    - 在编译系统的构建中实现检验过程
    - 增加检验函数或者设计检验算法
  - 间接检验
    - 除去直接检验之外的其它过程，不能在编译构建中完成
    - 如：文档规则规定了编写与源程序相关的文档时需要遵循的标准

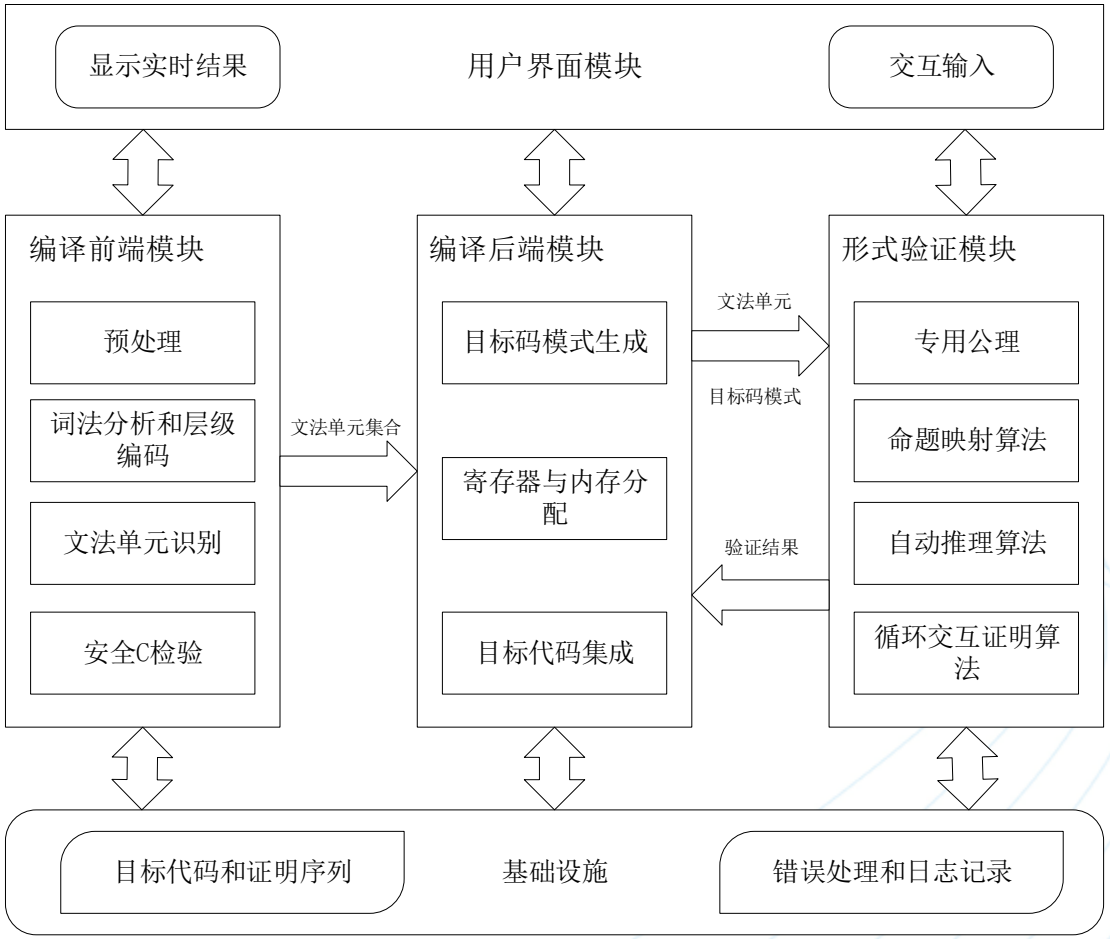


报告内容

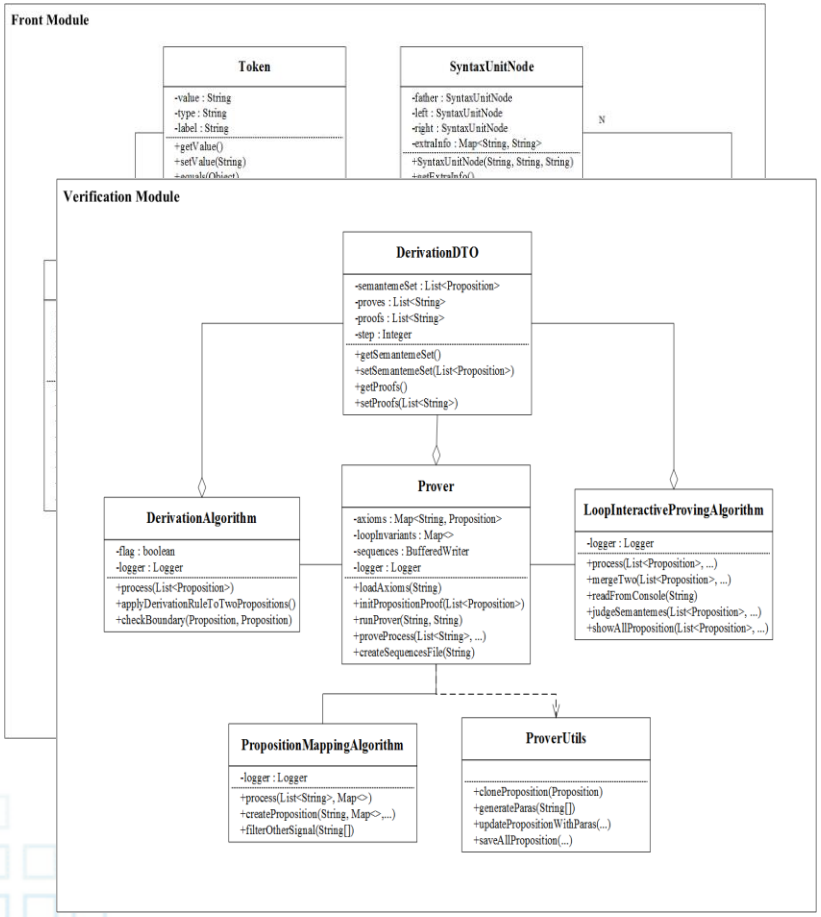
- 课题背景和意义
- 国内外研究现状
- 研究目标和内容
- 系统设计与实现
- 总结与展望
- 参考文献

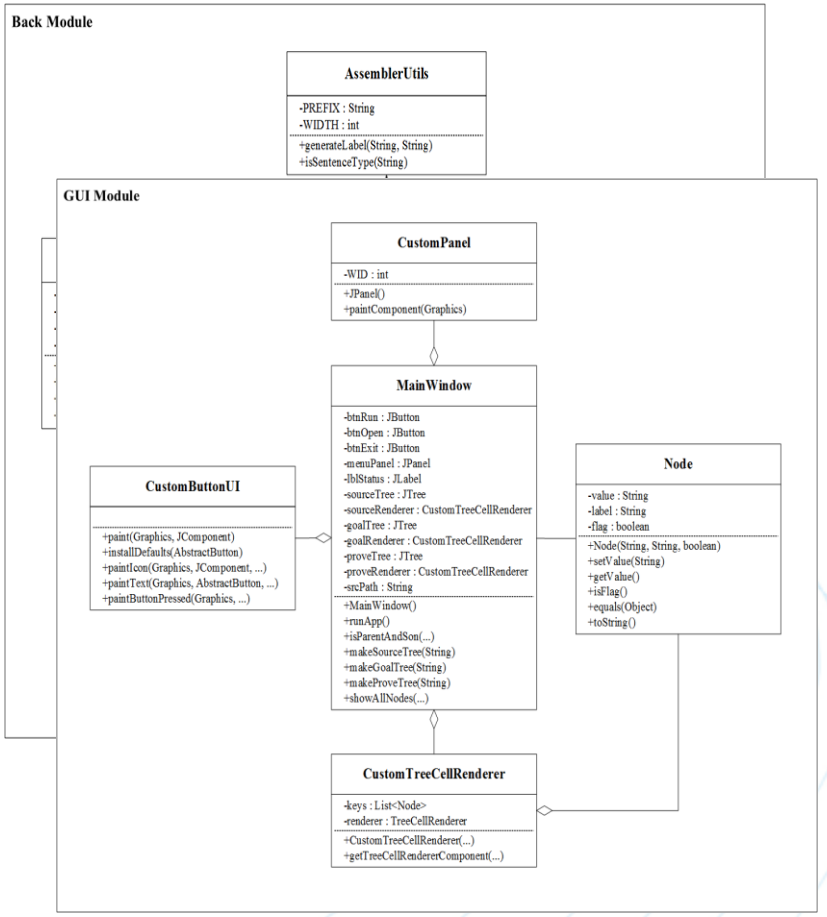
编译验证工具架构

- 编译前端模块
  - 处理输入的源代码并生成文法单元集合
- 编译后端模块
  - 目标代码生成与集成
- 形式验证模块
  - 编译语义形式验证
- 用户界面模块
  - 用户与程序交互的接口



系统设计与实现


Front Module
Token
-value : String
-type : String
-label : String
+getValue()
+setValue(String)
SyntaxUnitNode
-father : SyntaxUnitNode
-left : SyntaxUnitNode
-right : SyntaxUnitNode
-extraInfo : Map<String, String>
+SyntaxUnitNode(String, String, String)
N
Verification Module
DerivationDTO
-semantemeSet : List<Proposition>
-proves : List<String>
-proofs : List<String>
-step : Integer
+getSemantemeSet()
+setSemantemeSet(List<Proposition>)
+getProofs()
+setProofs(List<String>)
DerivationAlgorithm
-flag : boolean
-logger : Logger
+process(List<Proposition>)
+applyDerivationRuleToTwoPropositions()
+checkBoundary(Proposition, Proposition)
Prover
-axioms : Map<String, Proposition>
-loopInvariants : Map<>
-sequences : BufferedWriter
-logger : Logger
+loadAxioms(String)
+initPropositionProof(List<Proposition>)
+runProver(String, String)
+proveProcess(List<String>, ...)
+createSequencesFile(String)
LoopInteractiveProvingAlgorithm
-logger : Logger
+process(List<Proposition>, ...)
+mergeTwo(List<Proposition>, ...)
+readFromConsole(String)
+judgeSemantemes(List<Proposition>, ...)
+showAllProposition(List<Proposition>, ...)
PropositionMappingAlgorithm
-logger : Logger
+process(List<String>, Map<>)
+createProposition(String, Map<>,...)
+filterOtherSignal(String[])
ProverUtils
+cloneProposition(Proposition)
+generateParas(String[])
+updatePropositionWithParas(...)
+saveAllProposition(...)


Back Module
AssemblerUtils
-PREFIX : String
-WIDTH : int
+generateLabel(String, String)
+isSentenceType(String)
GUI Module
CustomPanel
-WID : int
+JPanel()
+paintComponent(Graphics)
MainWindow
-btnRun : JButton
-btnOpen : JButton
-btnExit : JButton
-menuPanel : JPanel
-lblStatus : JLabel
-sourceTree : JTree
-sourceRenderer : CustomTreeCellRenderer
-goalTree : JTree
-goalRenderer : CustomTreeCellRenderer
-proveTree : JTree
-proveRenderer : CustomTreeCellRenderer
-srcPath : String
+MainWindow()
+runApp()
+isParentAndSon(...)
+makeSourceTree(String)
+makeGoalTree(String)
+makeProveTree(String)
+showAllNodes(...)
CustomButtonUI
+paint(Graphics, JComponent)
+installDefaults(AbstractButton)
+paintIcon(Graphics, JComponent, ...)
+paintText(Graphics, AbstractButton, ...)
+paintButtonPressed(Graphics, ...)
Node
-value : String
-label : String
-flag : boolean
+Node(String, String, boolean)
+setValue(String)
+getValue()
+isFlag()
+equals(Object)
+toString()
CustomTreeCellRenderer
-keys : List<Node>
-renderer : TreeCellRenderer
+CustomTreeCellRenderer(...)
+getTreeCellRendererComponent(...)

系统设计与实现

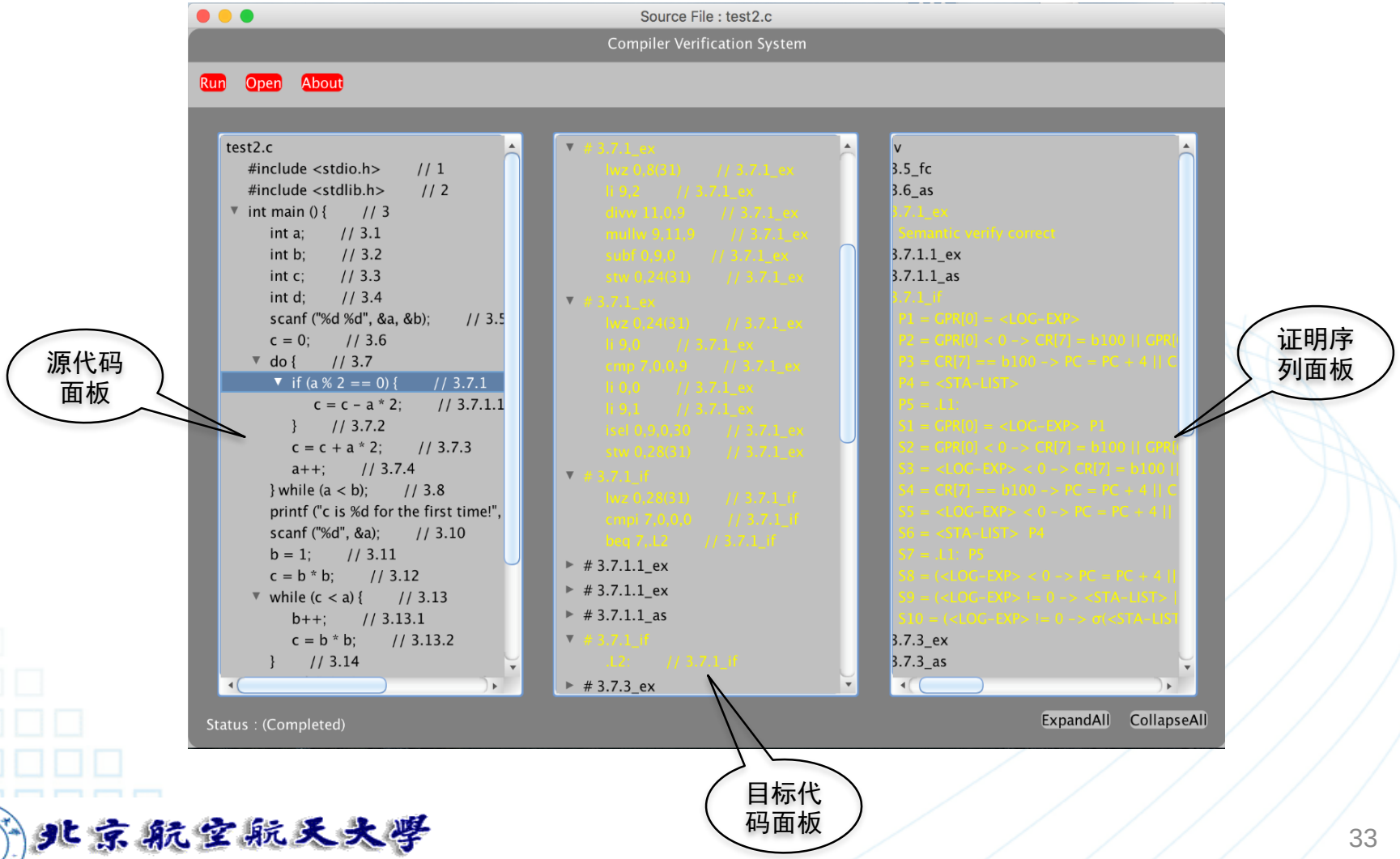
Work Content			Code lines	
CompilerVerification System	Front Module	Lexer	1169	3332
		Recognizer	1625	
		Front POJO	253	
		Utils & Others	285	
	Back Module	Assembler	1496	5837
		AssemblerExpression	4189	
		Utils & Others	152	
	Verification Module	Prover	905	2458
		Three Algorithm	1038	
		Utils & Others	515	
	GUI Module	MainWindow	815	1124
		Custom components	309	
	Total java lines		**12751**	

系统设计与实现

```
==========Labeled C Code==========
#include <stdi
#include <stdl
int main () {
    int a;
    int b;
    int c;
    int d;
    scanf ("%d
    c = 0;
    do {
        if (a
            c
        }
        c = c
        a++;
    } while (c
    printf ("c
    scanf ("%d
    b = 1;
    c = b * b;
    while (c <
        b++;
        c = b
    }
    printf ("T
    for (a = 0
        b++;
    }
    return 0;
}
```

```
for : 3.16_fo

==========目标码模式==========
        <ASS-EXP_1>
        b .L2
        .L1:
        <STA-LIST>
        <ASS-EXP_2>
        .L2:
        <LOG-EXP>
        cmpi 7,0,0,0
        bne 7,.L1

==========目标码模式命题==========
P1 = <ASS-EXP_1>
P2 = PC = PC + @.L2
P3 = .L1:
P4 = <STA-LIST>
P5 = <ASS-EXP_2>
P6 = .L2:
P7 = GPR[0] = <LOG-EXP>
P8 = GPR[0] < 0 -> CR[7] = b100 || GPR[0] > 0 -> CR[7] = b010 || GPR[0] == 0 -> CR[7] = b001
P9 = CR[7] == b100 -> PC = PC + @.L1 || CR[7] == b010 -> PC = PC + @.L1 || CR[7] == b001 -> PC = PC + 4

==========循环交互证明算法==========
用户输入的语义 :
σ(<ASS-EXP_1>)
{<LOG-EXP> != 0 -> [σ(<STA-LIST>); σ(<ASS-EXP_2>)]} ** n || <LOG-EXP> == 0 -> skip

辅助前提 :
P0 = (σ(<ASS-EXP_1>)) ∧ ({<LOG-EXP> != 0 -> [σ(<STA-LIST>); σ(<ASS-EXP_2>)]} ** n || <LOG-EXP> == 0 -> skip)

推导序列 :
S1 = <ASS-EXP_1>                P1
S2 = PC = PC + @.L2             P2
S3 = .L1:               P3
S4 = <STA-LIST>         P4
S5 = <ASS-EXP_2>                P5
S6 = .L2:               P6
S7 = GPR[0] = <LOG-EXP>         P7
S8 = GPR[0] < 0 -> CR[7] = b100 || GPR[0] > 0 -> CR[7] = b010 || GPR[0] == 0 -> CR[7] = b001           P8
S9 = <LOG-EXP> < 0 -> CR[7] = b100 || <LOG-EXP> > 0 -> CR[7] = b010 || <LOG-EXP> == 0 -> CR[7] = b001          S7,S8,MP
                                                .L1:                                                    # 3.7_dw
```

系统设计与实现


Source File : test2.c
Compiler Verification System
Run
Open
About
test2.c
#include <stdio.h> // 1
#include <stdlib.h> // 2
int main () { // 3
int a; // 3.1
int b; // 3.2
int c; // 3.3
int d; // 3.4
scanf ("%d %d", &a, &b); // 3.5
c = 0; // 3.6
do { // 3.7
if (a % 2 == 0) { // 3.7.1
c = c - a * 2; // 3.7.1.1
} // 3.7.2
c = c + a * 2; // 3.7.3
a++; // 3.7.4
} while (a < b); // 3.8
printf ("c is %d for the first time!",
scanf ("%d", &a); // 3.10
b = 1; // 3.11
c = b * b; // 3.12
while (c < a) { // 3.13
b++; // 3.13.1
c = b * b; // 3.13.2
} // 3.14
# 3.7.1_ex
lwz 0,8(31) // 3.7.1_ex
li 9,2 // 3.7.1_ex
divw 11,0,9 // 3.7.1_ex
mullw 9,11,9 // 3.7.1_ex
subf 0,9,0 // 3.7.1_ex
stw 0,24(31) // 3.7.1_ex
# 3.7.1_ex
lwz 0,24(31) // 3.7.1_ex
li 9,0 // 3.7.1_ex
cmp 7,0,0,9 // 3.7.1_ex
li 0,0 // 3.7.1_ex
li 9,1 // 3.7.1_ex
isel 0,9,0,30 // 3.7.1_ex
stw 0,28(31) // 3.7.1_ex
# 3.7.1_if
lwz 0,28(31) // 3.7.1_if
cmpi 7,0,0,0 // 3.7.1_if
beq 7,.L2 // 3.7.1_if
3.7.1.1_ex
3.7.1.1_ex
3.7.1.1_as
3.7.1_if
.L2: // 3.7.1_if
3.7.3_ex
v
3.5_fc
3.6_as
3.7.1_ex
Semantic verify correct
3.7.1.1_ex
3.7.1.1_as
3.7.1_if
P1 = GPR[0] = <LOG-EXP>
P2 = GPR[0] < 0 -> CR[7] = b100 || GPR[
P3 = CR[7] == b100 -> PC = PC + 4 || C
P4 = <STA-LIST>
P5 = .L1:
S1 = GPR[0] = <LOG-EXP> P1
S2 = GPR[0] < 0 -> CR[7] = b100 || GPR[
S3 = <LOG-EXP> < 0 -> CR[7] = b100 ||
S4 = CR[7] == b100 -> PC = PC + 4 || C
S5 = <LOG-EXP> < 0 -> PC = PC + 4 ||
S6 = <STA-LIST> P4
S7 = .L1: P5
S8 = (<LOG-EXP> < 0 -> PC = PC + 4 ||
S9 = (<LOG-EXP> != 0 -> <STA-LIST> |
S10 = (<LOG-EXP> != 0 -> σ(<STA-LIST
3.7.3_ex
3.7.3_as
Status : (Completed)
ExpandAll
CollapseAll
源代码面板
证明序列面板
目标代码面板

报告内容

- 课题背景和意义
- 国内外研究现状
- 研究目标和内容
- 系统设计与实现
- 总结与展望
- 参考文献

总结与展望

- 总结
  - 提出一种基于编译语义的形式化验证方法，实现从源代码到目标码的编译过程的验证，并给出了应用于编译形式化验证过程中的三大核心算法。
  - 提出了安全C编译器构建中的关键方法，并给出了编译构建中的系统架构。其中，下推自动机解决了文法单元的识别问题，层级编码和安全C检验方法实现了DO178C规定的编译验证系统的可追踪性和高安全性需求。
  - 实现了一套基于上述编译构建和验证方法的编译验证工具原型，为安全C编译器的构建和验证提供一种可行的解决方案。
- 展望
  - 为编译验证系统加入代码优化部分。

参考文献

[1] GCC5[EB/OL], http://www.phoronix.com/scan.php?page=news_item&px=MTg3OTQ

[2] GCC[EB/OL], http://gcc.gnu.org/

[3] Surhone L M, Tennoe M T, Henssonow S F. Compcert[M]. Betascript Publishing, 2010.

[4] Leroy, Xavier. "Formal verification of a realistic compiler." Communications of the ACM 52.7 (2009): 107-115.

[5] COQ DEVELOPMENT TEAM. The Coq proof assistant reference manual[J]. TypiCal Project, 2012.

[6] Yang, Xuejun, et al. "Finding and understanding bugs in C compilers." ACM SIGPLAN Notices. Vol. 46. No. 6. ACM, 2011.

[7] Li, Zhaopeng, et al. "A Certifying Compiler for Clike Subset of C Language." IEEE International Symposium on Theoretical Aspects of Software Engineering IEEE Computer Society, 2010:47-56.

[8] Necula, George C. "Proof-carrying code." Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. ACM, 1997.

[9] Feng X, Shao Z, Vaynberg A, et al. Modular verification of assembly code with stack-based control abstractions[C]//ACM SIGPLAN Notices. ACM, 2006, 41(6): 401-414.

[10] McCarthy J, Painter J. Correctness of a compiler for arithmetical expressions[C]//Mathematical Aspects of Computer Science 19: Proc of Symposia in Applied Mathematics, 1967: 33−41.

谢谢大家！

请老师批评指导！