基于模拟关系的编译优化实现正确性验证方法

徐 超[1,2,3],何炎祥[1,3],吴 伟[1,3],陈 勇[1,3],刘健博[1,3]
(1.武汉大学计算机学院,湖北武汉 430072;2.徐州工业职业技术学院,江苏徐州 221000;
3.软件工程国家重点实验室,湖北武汉 430072)

摘 要: 编译器中通常采用各种优化方法来提高目标代码的质量,为了实现较好的效果,一些编译优化算法通常十分复杂,很容易给可靠性和安全性带来隐患.现有的编译器缺陷大部分是由优化阶段引起的.传统的编译优化正确性研究大部分只关注优化算法的正确性,但是只有该算法被正确的实现了才能确保实际运行的优化过程是正确的.本文提出一种基于模拟关系的方法来验证编译优化实现的正确性.在每次优化结束后,我们通过建立优化前代码和优化后代码之间的模拟关系生成优化正确应满足的逻辑条件,然后验证逻辑条件是否成立从而判定编译优化的实现是否正确性.以优化编译中的常量折叠优化和变量替换的验证作为示例显示了本方法的有效性和可靠性.

关键词: 编译优化; 正确性; 模拟关系; 定理证明
中图分类号: TP309.1 **文献标识码:** A **文章编号:** 0372-2112 (2012)11-2171-06
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2012.11.005

Verifying Implementation Correctness of Compiling Optimization Based on Simulation Relation

XU Chao[1,2,3], HE Yan-xiang[1,3], WU Wei[1,3], CHEN Yong[1,3], LIU Jian-bo[1,3]
(1. *Wuhan University Computer of School*, *Wuhan*, *Hubei* 430072, *China*;
2. *Xuzhou College of Industrial Technology*, *Xuzhou*, *Jiangsu* 221000, *China*;
3. *State Key Laboratory of Software Engineering*, *Wuhan*, *Hubei* 430072, *China*)

Abstract: Many optimization methods are used to raise the quality of target code in the process of compiling. Because some optimization methods for compiling are too complicated, hidden dangers won't be avoided for reliability and security of compiling. The existing defects of compiler are usually caused at the optimization phase. Mostly traditional correctness of compiling optimization only focus on correctness of optimization algorithm, but only when the algorithm is implemented correctly, the optimization process run in practice can be proved to be correct. This paper proposes a method based on simulation relation to verify correctness of compiling optimization realization. We build up simulation relation between code before optimization and code after optimization to generate logic condition that the correctness optimization should satisfy, and then we verify it if logic condition is satisfied or not to judge if compiling optimization realization is correct or not after each optimization is finished. The method is proved to be effective and reliable by verifying statement exchange and variable substitution in the process of optimization compiling.

Key words: compiling optimization; correctness; simulation relation; theorem proving

1 引言

编译器的可信性越来越受到重视.为了使生成的目标代码运行速度快、体积小,编译器通常对代码进行编译优化.编译优化就是对代码进行语义等价的变换,使代码在空间上和时间上的效率更高.随着研究的不断深入,编译优化的种类不断增多、复杂度也不断增大,导致优化变得越来越不可信.任何软件都需要经过编译器的编译,因此编译器在可信软件的构造中起着关键作用,而通常编译器的不可信大部分是由于编译优化的缺陷造成的,因此保证编译优化的正确性是提高编译器可信性的重要方面.

在编译器开发中,测试是一项重要的提高编译器可信性的方法.但是由于编程语言的复杂性,许多编译优化错误需要经过大量的测试才能发现,甚至直到出现重大事故时才会被发现.因此,对编译优化正确性进行形

收稿日期:2012-03-15;修回日期:2012-06-13
基金项目:国家自然科学基金重点项目(No.91118003);国家自然科学基金面上项目(No.61170022)

式化验证可以有效弥补测试的效率低、遗漏率大等问题.许多研究者对编译优化的正确性验证进行了研究,Lacey 等人[1,2]采用一种归纳证明方法对过程式语言程序上多种经典优化[3]的正确性进行了形式证明.在他们的方法中,编译优化被表示为含条件的重写规则:$I \Rightarrow I'$ if ϕ,其含义为:如果条件 ϕ 满足,则可实施变换 $I \Rightarrow I'$.其中,I 和 I' 表示优化前后代码.ϕ 采用一种扩展的时序逻辑 CTL-FV 表达.除了文献[1,2]之外,过程式语言上优化转换正确性证明的相关研究还包括文献[4,5].这些研究对经典优化的正确性考察得较多,然而对于各种高级优化(指令调度、向量化、Cache 优化等[6,7]的正确性还未能给出严格证明.从研究的对象可以分为两类,一类是研究编译优化算法的正确性[1,2,4,5,8~10].这类研究的内容包括给出程序语言的语义、优化算法的形式化描述、证明优化前后代码语义等价的形式化证明.另一类是研究编译优化算法实现的正确性[1,2,4,5,8~10],基本思想是在每次编译优化后验证优化后的代码与优化前的代码语义一致,这类方法通常包括 certifying compiler[11~13] 和 translation validation[14~16].在这种方法的尝试中,证明携带代码(Proof-Carrying Code)[13,17]是一种比较强大的灵活的实现系统,原理上,它允许任何逻辑系统作为其系统基础.本文主要研究第二类方法,传统的研究过程复杂,自动化程度不高,本文给出一种高效的验证方法.首先从优化正确性的语义角度定义两个程序段之间的模拟关系,将其作为编译优化正确性的判断标准.验证的过程利用每次编译优化前后的程序的控制流图生成模拟关系,通过定理证明工具判断模拟关系是否成立.

2 中间语言语法和迁移系统

2.1 中间语言语法

本文利用一种简单的过程式中间语言来说明编译优化实现正确性验证的过程,其语法如图 1 所示.整个程序是由各个子函数组成.一个函数体是由中间代码指令语句组成.中间代码指令语句包括变量赋值语句,内存读写语句,函数调用,标签语句,无条件跳转语句,条件跳转语句以及返回语句.其中函数调用语句第一个参数是所调用函数的地址,后面是参数列表.中间代码的表达式包含了取地址运算、算术运算、位运算以及关系运算.$a|b|c|\cdots$表示代码中出现的所有变量名字.

```
Statements S   ::=x:=E|load(E,x)|store(x,E)|label(L)|goto L|
                  if(E) goto L|x:=call(E,E1,E2,...)|return E
Expressions E ::=const|x|&x|E1 op E2
 Operators op  ::=+|-|*|&|=|>|...
    Variable x  ::=a|b|c|...
       Label L  ::=L0|L1|...
```

图1 中间语言语法

变量的不同取值反应了程序的运行状态,本文即利用变量取值之间的关系来验证编译优化的正确性.为了方便验证过程,需要把中间代码转换成静态单赋值(SSA)形式,使得变量不会被重复赋值.

2.2 迁移系统

编译优化正确性的证明需要验证编译前后代码语义的一致性,为了描述中间语言的形式化语义,我们采用迁移系统来描述.一个迁移系统 TS 实际上是一个状态机,可以用四元组 $< V, O, \theta, \pi >$ 表示.其中,V 表示状态变量集合,O 表示程序运行过程中可观察的变量集合,它反映了程序的动态行为,因此,在验证的过程中我们只需要关注这类变量.θ 表示系统运行时的初始状态应满足的条件.π 表示状态迁移关系,描述了状态之间的迁移和迁移条件.状态变量一般是带类型的,状态是使用类型一致的变量表示的.对于一个状态 S 和变量 $x \in V$ 来说,我们使用 $S[x]$ 表示在状态 s 下 x 的值.如果在一个状态迁移过程中,一个变量被赋值了,那么迁移关系中通常涉及到一个变量的两种版本,即赋值之前的版本和赋值之后的版本.为了避免变量名字的动态性,我们采用静态单赋值(SSA)形式的中间代码.如在状态迁移中,"$x = x + 1$"变换为"$x' = x + 1$".

用迁移系统表示代码的语义时,比较两个系统是否等价可以通过比较它们的可观察变量是否一致.如程序的输出文件、运行时打印的字符序列等.甚至可以包括调用其他程序的可观察变量.迁移系统的语义计算过程是求最长有限或者无限的状态序列 $\rho: s_0, s_1, \cdots$,这个状态序列的起始状态满足初始条件 θ,即 $s_0 \vdash \theta$.状态序列中每两个相邻的状态通过状态迁移关系关联起来,即对于任意的 $i, 0 \leqslant i+1 < |\rho|$,有 $< s_i, s_{i+1} > \vdash \pi$ 成立.如果状态序列是有限的,$|\rho|$ 表示状态序列长度,否则表示无穷大.例如图 2 中给出了一段中间代码,我们以这段代码为例说明如何将中间代码转换成一个迁移系统.使用代码行前的标号表示程序位置.程序状态变量集合 $V = \{pc, a, b, c, d\}$,其中 pc 表示程序下一步该执行的程序位置.Pc 的范围是 $\{L0, L1, L2, L3\}$.其它变量对应着程序中的整型变量.起始状态条件为 $\theta: pc = L0$,表示程序从位置 $L0$ 开始执行.可观察变量集合 $O = \{d\}$.状态迁移关系 π 可以用式子 $\pi = \pi_{0,1} \vee \pi_{0,2} \vee \pi_{1,2} \vee \pi_{2,3}$ 表示,

```
L0   a=1;b=1;if(a>b) goto L2
L1   c=a+b;
L2   d=a-c;
L3
```

图2 示例代码

其中,$\pi_{i,j}$ 表示程序从位置 Li 迁移到位置 Lj 所应满足的条件.$\pi_{i,j}$ 表示的具体内容如图 3 所示.

假设编译优化前后的程序的迁移系统用 $TS_s = <$

$V_s, O_s, \theta_s, \pi_s >$ 和 $TS_t = < V_t, O_t, \theta_t, \pi_t >$ 表示,如果两个迁移系统之间的可观察变量 O_s 和 O_t 存在一一对应关系,那么就说这两个系统是可比较的.我们用 $X \in O_s$ 表示 TS_s 中可观察变量,用 $x \in O_t$ 表示 TS_t 中对应的可观察变量.如果对于任何 TS_s 的有限语义计算序列 $\rho^s: \rho_0^s, \cdots, \rho_m^s$,存在 TS_t 的有限语义计算序列 $\rho^t: \rho_0^t, \cdots, \rho_n^t$,满足任意 $x \in O_t, \rho_n^t[x] = \rho_m^s[X]$ 成立,那么就说 TS_s 和 TS_t 是语义等价的.

$$\pi_{0,1} = (pc = L0) \wedge (a = 1) \wedge (b = 1) \wedge (a \le b)$$

$$\pi_{0,2} = (pc = L0) \wedge (a = 1) \wedge (b = 1) \wedge (a > b)$$

$$\pi_{1,2} = (pc = L1) \wedge (c = a + b)$$

$$\pi_{2,3} = (pc = L2) \wedge (d = a - c).$$

图3　迁移关系

3　模拟关系及语义一致性验证

模拟关系涉及到两个中间代码程序 S 和 T 分别表示优化前代码和优化后代码,是形式为 (PC_S, PC_T, φ) 的元素的集合.其中,PC_S 和 PC_T 分别表示 S 和 T 控制流图中的程序点.ϕ 表示程序变量间的逻辑关系表达式,程序变量包含优化前程序和优化后程序的变量,关系一般使用线性算术关系,这样易于自动化证明.从非形式角度讲,一个模拟关系实际上是指两个程序片段语义相等应满足的条件.两个程序片段语义相等可能会有多种不同程度的度量标准.度量标准应该谨慎的选择,如果标准太严格、颗粒度太细会导致复杂的编译优化无法验证通过,同样如果标准太宽泛就会导致无法精确描述某些编译优化错误,降低了验证的准确性.

使用迁移系统表示程序的语义,根据 2.2 节中的说明,两个程序的语义等价可以表示为程序执行轨迹中可观察行为的一致性.如果两个程序执行轨迹的函数调用序列等价并且返回值一致我们就说这两个执行轨迹是等价的.两个执行轨迹的函数调用等价是指在函数调用时的内存状态一致、参数一致并且调用的函数地址一致.返回值一致是指返回的值相同并且返回时的内存状态等价.因为我们是在中间代码层次上对编译优化实现的正确性进行验证,内存状态可以用符号状态表示,它们之间的等价性可以用程序变量之间的约束关系来表示.

假设 Σ 是程序 S 和 T 之间的模拟关系,对于任意的 $(PC_S, PC_T, \phi) \in \Sigma$,如果 S 在 PC_S 处的任意执行状态和 T 在 PC_T 处的任意执行状态满足条件 ϕ,那么就说这个模拟关系是一个正确的模拟关系.因此,验证编译优化的正确性可以先为优化前后代码建立一个正确的模拟关系,然后通过某种算法判断模拟关系是否成立.本文在第 4 节将给出一种有效的算法.以模拟关系作为两个程序段的等价标准需要两个程序段的执行轨迹是有限的,即可终止的.针对可观察行为,优化编译器可以产生一个模拟关系来解释优化过程是如何转换程序的.在这种情况下,我们就可以通过检测模拟关系是否正确来验证优化的正确性.基于此,本文提出一种方法可以有效的检测模拟关系的方法.

4　编译优化实现正确性验证过程

证明语义一致性的主要思路是首先利用迁移系统给出优化前后程序的形式语义,然后证明优化前后语义计算结果是等价的.示意图如图 4 所示.图中左边两个节点分别表示优化前后代码的语法表示,右边两个节点 Sem(S)与 Sem(T)分别表示相应的语义.验证 Sem(S)与 Sem(T)的一致性可以利用模拟关系进行,即在相同的初始状态下,S 和 T 的运行轨迹有一一对应关系.对于简单的优化,可以利用前后代码产生逻辑验证条件,然后验证验证条件是否成立来判定优化是否正确.对于较复杂的优化,则靠自动生成的验证条件还不够充分,还需要一些额外的信息如优化时的控制流变化信息、变量变换映射等,而这些信息可以由编译器给出或者在代码中添加辅助信息.

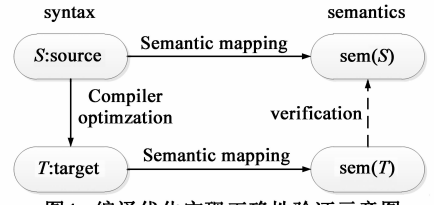


图4　编译优化实现正确性验证示意图

基于模拟关系的编译优化实现正确性验证方法首先要建立一个优化前后控制流图节点之间的映射关系.映射关系包括基本块和变量之间的映射.自动建立映射关系是本文验证方法的关键和难点.首先,我们定义对应基本块和对应变量.设 $\vec{b} = b_1 b_2 \cdots b_k$ 和 $\vec{b'} = b'_1 b'_2 \cdots b'_k$ 分别是程序 S 和 T 中的基本块序列,$\vec{x} = x_1 x_2 \cdots x_h$ 和 $\vec{x'} = x'_1 x'_2 \cdots x'_h$ 分别是程序 S 和 T 中的变量序列.用 $\rho^s: \rho_0^s, \cdots, \rho_m^s$ 和 $\rho^t: \rho_0^t, \cdots, \rho_n^t$ 分别表示程序 S 和 T 的语义计算序列,设 $\rho_b^s: \rho_{i1}^s, \rho_{i2}^s \cdots$ 和 $\rho_{b'}^t: \rho_{j1}^t, \rho_{j2}^t \cdots$ 分别表示包含和的语义计算子序列.在相同的起始状态下运行程序 S 和 T,如果以下条件满足,我们称基本块序列 $\vec{b}$ 和 $\vec{b'}$,变量序列 $\vec{x}$ 和 $\vec{x'}$ 之间存在对应关系,即本文验证方法所需要的映射关系.

(1) $\forall l, \rho_{ik}^s[PC] = b_l \Leftrightarrow \rho_{jk}^t[PC] = b'_l$

(2) $\rho_{ik}^s[x_1] = \rho_{jk}^t[x'_1], \cdots \rho_{ik}^s[x_h] = \rho_{jk}^t[x'_h]$

(3) $\rho_{ik+1}^s[x_1] = \rho_{jk+1}^t[x'_1], \cdots \rho_{ik+1}^s[x_h] = \rho_{jk+1}^t[x'_h]$

上面的条件实际上表示两个程序中访问对应块的次序要一致,同时对应变量的值在对应基本块的出口

和入口要相等.为了获得对应基本块和对于变量,我们采用一种基于随机解释的方法获取尽可能大的基本块和变量对应集合.首先随机选取输入数据,然后对程序 S 和 T 进行解释执行,执行的过程中记录基本块的执行次序和变量的内存值.这样执行多次就会得到许多执行历史记录,显然,如果是对应基本块,其位置在历史记录中次序相同,对应变量的值也相等.我们可以根据历史记录利用统计方法猜测出对应基本块和基本变量.“猜测”表示这种方法的结果有可能是错误的,尽管如此,因为供猜测的方案是有限的,如果验证不成功,可以尝试下一个猜测.如果遍历完所有的可能性验证仍然失败,则可以确定编译优化过程出现错误.

在获得映射关系后,需要利用编译器根据优化规格建立节点间的满足条件.验证编译优化的正确性就是验证编译前后模拟关系的正确性.验证模拟关系正确性的算法如图 5 所示.

Algorithm 1: optimization validation algorithm
Name: opva
Input: $\kappa: PC_S \rightarrow PC_T$ program location map
$\gamma: PC_S \times PC_T \rightarrow \phi$ Simulation relation map
CFG_S, CFG_T : control flow graph of program
PC : *CFG* node
Output: Bool

```
1   begin
2       if marked(PC)
3           Return true
4       else
5           mark(PC)
6       endif
7       for PC' in successor(PC):    /*对 PC 的后继节点进行遍历处理*/
8           φ1 = gettraconditon(CFG_S, PC, PC')   /*获得优化前转移条件*/
9           φ2 = gettraconditon(CFG_T, κ(PC), κ(PC'))   /*获得优化后转移条件*/
10          φ3 = γ(PC, κ(PC))
11          φ4 = γ(PC', κ(PC'))
12          if not solve(φ1 ∧ φ2 ∧ φ3 → φ4)   /*验证生成的条件*/
13              return false
14          endif
15          if not opva(κ, γ, CFG_S, CFG_T, PC')   /*递归验证后继节点*/
16              return false
17          endif
18      endfor
19      return true
20  end
```

图5 编译优化实现正确性验证算法

算法的输入包括优化前后流程图节点即程序位置之间的映射 κ、模拟关系映射 γ、优化前后程序的控制流图 CFG_S,CFG_T 以及优化前控制流图的节点 PC.如果验证成功,输出 true,否则输出 false.算法主要基于结构归纳法,证明模拟关系的正确性就是证明在任何节点处,如果与优化后控制流图对应节点满足模拟条件,那么这个节点的后继节点也满足模拟条件.算法的主要流程是递归遍历优化前控制流图的节点,证明每个节点满足上述性质.在算法过程中需要检测生成的验证条件是否成立,条用了 solve 函数,这个函数可以有多种实现方法,如使用可满足性求解器、定理证明器等一些自动化证明工具.在算法的过程中调用了 gettraconditon 函数来生成节点之间状态迁移条件,这个函数的输入是控制流图上两个结点,输出是这两个节点之间的状态迁移条件.算法主要通过求以两节点为端点的路径上相邻节点的状态迁移条件的合取获得输出结果.算法中 π 是控制流图中边上的状态迁移条件映射.

Algorithm 2: get transition condition between two program location
Name: gettraconditon
Input: CFG : control flow graph of program
PC , PC' : program location
Output: constraint condition

```
1  φ = Φ      /*设置初始条件为空*/
2  While successor(PC) ≠ PC'      /*判断后继节点是否为目标节点*/
3      φ = φ ∧ π_{PC,successor(PC)}      /*合并转移条件*/
4      PC = successor(PC)      /*处理下一个节点*/
5  Endwhile
6  Return φ
```

图6 获取两个程序的之间的迁移条件算法

5 应用示例

常量折叠是一种常见的优化,可以有效提高代码的执行速度.在寄存器分配阶段,经常要进行的代码变换是变量名字替换.为了说明本文方法的基本过程,利用包含常量折叠优化和变量名字替换的具体示例来进行说明.示例程序见图 7,图中(a)表示优化前代码;(b)表示优化后代码.为了便于说明,直接采用程序标号作为控制流图的节点.程序中 a, b, c, d, e, f 为全局变量,$t1, t2$ 为局部变量.节点之间的映射关系为 $\{L1 \rightarrow L1', L5 \rightarrow L4', L9 \rightarrow L6', L11 \rightarrow L7'\}$,根据变量变换规则,变量之间的映射关系为 $\{a \rightarrow a', b \rightarrow b', c \rightarrow c', d \rightarrow d', e \rightarrow e', f \rightarrow f'\}$.据此可建立优化前后代码之间的模拟关系.具体为 $\gamma(L1, L1') := i = i' \wedge b = b' \wedge c = c' \wedge d = d'$,$\gamma(L5, L4') := e = e' \wedge a' = i + 1 \wedge a = a' \wedge c = c'$,$\gamma(L9, L6') := i + 1 = a' \wedge a = a' \wedge c = c'$,$\gamma(L11, L7') := e = e'$.接下来就要证明每个对应节点处的模拟关系都成立.图 7 中所示程序的赋值语句及条件语句可以看作迁移系统的迁移条件,根据图 5 中算法描述,需要验证的命题包括:

$$\gamma(L1, L1') \wedge a = i + 1 \wedge f = b + c \wedge t1 = c + d \wedge e = t1 + b \wedge a' = i' + 1 \wedge f' = b' + c' \wedge e' = f' + d' \rightarrow \gamma(L5, L4') \quad (1)$$

$$\gamma(L5, L4') \wedge t2 = i + 1 \wedge b = t2 \wedge c = b \wedge e > 0 \wedge c' = a' \wedge e' > 0 \rightarrow \gamma(L9, L6') \quad (2)$$

$$\gamma(L5, L4') \wedge e \leqslant 0 \wedge e' \leqslant 0 \rightarrow \gamma(L9, L6') \quad (3)$$

$$\gamma(L9, L6') \wedge f = i + 1 \wedge b = f + a \wedge e = b - c \wedge b' = 2a' \wedge e' = b' - c' \rightarrow \gamma(L5, L8) \quad (4)$$

显然这些命题都是可以证明通过的,因此,示例所示的编译优化的正确性得以验证.

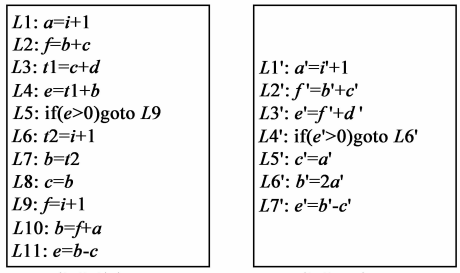


图7　示例程序

本文提出的算法是可靠的,因为如果两个程序状态的行为是一致的,则认为这两个程序状态等价.我们这里将程序的输出视为程序行为,具体表现为可观察变量.本算法运行成功,则表明优化前后程序存在可观察状态序列 $\rho: \rho_0, \cdots, \rho_m$ 和 $\rho': \rho_0', \cdots, \rho_i'$,且 ρ_i 和 $\rho_i'(0 \leqslant i \leqslant m)$的可观察变量保持一致,即表明了优化前后程序的行为从初始状态到结束状态都是等价的.所以本算法可以正确可靠的验证编译器优化的实现正确性.

6　结论

可信编译器是可信软件开发工具链中一个重要环节[18],编译优化的正确一直是编译器可信性[19]的重要条件.许多商业编译器的 bug 大部分出现在编译优化阶段[20].传统的测试方法无法完全保证编译优化的正确性[21],对编译优化算法的正确性验证也无法确保其实现的正确性.因此,本文主要研究了编译优化实现的正确性验证方法.利用 translation validation 思想,基于两个程序段之间的模拟关系提出了一种可以利用外部定理证明器的验证方法.这种方法的优点是可以高效利用已有的定理证明工具,提高验证过程的自动化程度.本方法无需对传统编译器进行大量修改,可以很好的与传统编译器结合起来,实现对代码复用的同时提高可信性.如何生成模拟关系是本方法的关键步骤,通常需要根据具体的优化算法的规格来实现,一些常见的比较规范的优化算法可以实现自动生成模拟关系,而比较复杂的就需要结合人工生成.本文接下来的工作主要是研究如何提高模拟关系生成的自动化程度以及对程序结构变化比较复杂的优化算法的验证.

参考文献

[1] Lacey D′Jones ND, Wyk EV, Frederiksen CC. Compiler optimization correctness by temporal logic[J]. Higher-Order and Symbolic Computation, 2004, 17(3): 173 – 206.

[2] Frederiksen CC. Correctness of classical compiler optimizations using CTL[J]. Electronic Notes in Theoretical Computer Science, 2002, 65(2): 37 – 51.

[3] Abo AV, Sethi R, Ullman JD. Compilers: Principles, Techniques, and Tools[M]. Boston: Addison-Wesley, 1986: 86 – 99.

[4] Kozen D, Patron M. Certification of compiler optimizatious using Kleene algebra with tests[A]. In: Lloyd nE Dahl v, Furbach U. Kerber M, Lau K, Palamidessi C. Pereim LM, Sagiv Y, Stuckey PJ, eds. Proc. of the I st Int′l Conf. on Computational Logic(CL2000)[C]. London: Springer-Verlag, 2000. 568 – 582.

[5] Benton N. Simple relational correctness proofs for static analyses and program transformations[J]. ACM SIGPLAN Notices, 2004. 39(1): 14 – 25.

[6] Muchnick SS. Advanced Compiler Design and Implementation [M]. San Francisco: Morgan Kaufrnann Publishers, 1997: 106 – 110.

[7] Alien R, Kennedy K. Optimizing Compilers for Modem Architectures: A Dependence-Based Approach[M]. San Francisco: Morgan Kaufinann Publishers, 2002: 23 – 30.

[8] Wand M. Specifying the correctness of binding time analysis [A]. In: Deusen MV, Lang B, eds. Proc. of the 20th ACM SIGPLANSIGACT Symp. on Principles of Programming Languages[C]. New York: ACM Press, 1993. 137 – 143.

[9] Wand M, Siveroni I. Constraint systems for useless variable elimination[A]. In: Appel A, Aiken A . eds. Proc. of the 26th ACM SIGPLAN-SlGACT Symp. On Principles of Programming Languages[C]. NewYork: ACMPress, 1999. 291 – 302.

[10] Kubayashi N. Type-Based useless variable elimination [J]. ACM S1GPLANNotices, 1999, 34(11): 84-93.

[11] Kozen, D Efficient code certification. Technical Report[R]. Cornell Univer-sity, Ithaca, USA, 1998: 98 – 166.

[12] Morrisett G, Walker D, Crary K, Glew N. From system F to typed assem-bly language[A]. ACM Transactions on Programming Languages and Systems[C]. Tennessee, USA, 1999, 21(3): 527 – 568.

[13] Necula, G. Proof-carrying code[A]. In: Proc of the 24th ACM SIGPLAN-SIGACT SYMP[C]. On Principles of Programming Languages, New York, USA, 1997. 106 – 119.

[14] 胡定磊,陈书明.基于互补谓词的编译优化[J].电子学报,2006,34(07):1280 – 1285.
HU Ding-lei, CHEN Shu-ming. Optimizing compiler based on complementary predicate[J]. Acta Electronica Sinica, 2006, 34(07): 1280 – 1285. (in Chinese)

[15] 胡定磊,陈书明.低功耗编译技术综述[J].电子学报,2005,33(04):676 – 681.

HU Ding-lei, CHEN Shu-ming. Low power/ energy compilation technology[J]. Acta Electronica Sinica, 2005, 33(04): 676 – 681. (in Chinese)

[16] 唐遇星,邓▮,周兴铭.基于 Trace-Cache 的多级动态优化框架设计[J].电子学报,2005,33(11):1946 – 1951.
TANG Yu-xing, DENG Kun, ZHOU Xing-ming. Trace-cache based framework for mult-i level dynamic optimization[J]. Acta Electronica Sinica, 2005, 33(11): 1946 – 1951. (in Chinese)

[17] PC Van Oorschot. Revisiting software protection[A]. In: Proc of 6th international Information Security Conference (ISC′03). Lecture Notes in Computer Science 2851[C]. Berlin: Springer-Verlag, 2006. 1 – 13.

[18] A Appel. Verified software toolchain[J]. Programming Languages and Systems, 2011, 66(02): 1 – 17.

[19] 何炎祥,吴伟,刘陶,等.可信编译理论及其核心实现技术[J].计算机科学与探索,2011,5(1):1 – 22.
HE Yanxiang, WU Wei, LIU Tao. Theory and key implementation techniques of trusted compiler: A survey[J]. Journal of Frontiers of Computer Science and Technology, 2011, 5(1): 1 – 22. (in Chinese)

[20] X Yang, Y Chen, E Eide, J Regehr. Finding and understanding bugs in C compilers[A]. In 32nd Conf. on Programming Language Design and Implementation (PLDI′11)[C]. San Jose, California, 2011. 283 – 294.

[21] S Kundu, Z Tatlock, S Lerner. Proving optimizations correct using parameterized program equivalence[J]. ACM Sigplan Notices, 2009, 44(6): 327 – 337.

[22] Lerner S, Millstein T, Chambers C. Automatically proving the correctlless of compiler optimizations[J]. ACM SIGPLAN Notices, 2003, 38(5): 220 – 231.

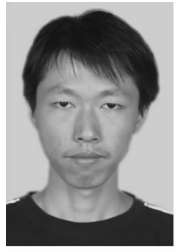
[23] Necula GC. Translation validation for optimizing compiler[J]. Acm Sigplan Notices, 2000. 35(5): 83 – 94.

[24] Zaks A, Pnueli A. Program analysis for compiler validation [A]. In: Proc. of the 8th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering [C]. New York, USA, 2008. 1 – 7.

[25] 聂久焘,程旭,王克义.一种高效的完全值编号算法[J].电子学报,2010,38(2):416 – 421.
NIE Jiu-tao, CHENG Xu, WANG Ke-yi. An efficient complete global value numbering algorithm[J]. Acta Electronica Sinica, 2010, 38(2): 416 – 421. (in Chinese)

作者简介

徐 超 男,1980 年出生,湖北红安人,博士生,讲师,CCF 会员(E200019473G),研究方向为可信软件和嵌入式系统.
E-mail: xuch@whu.edu.cn



吴 伟 男,1985 年,河南新县人,博士生,研究方向为可信软件和编译理论.
E-mail: wuwei@whu.edu.cn

何炎祥 男,1952 年出生,湖北应城人,博士,教授,博士生导师,研究方向为可信软件、并行分布处理、软件工程和嵌入式系统.
E-mail: yxhe@whu.edu.cn

陈 勇 男,1986 年出生,湖南冷水江人,博士生,研究方向为可信软件和嵌入式系统.
E-mail: cyong@whu.edu.cn