软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(4):819−826 [doi: 10.13328/j.cnki.jos.005196]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于 Z3 的 Coq 自动证明策略的设计和实现*

张恒若[1], 付 明[2,3]

[1](中国科学技术大学 信息科学技术学院,安徽 合肥 230026)

[2](中国科学技术大学 计算机科学与技术学院,安徽 合肥 230026)

[3](中国科学技术大学 苏州研究院软件安全实验室,江苏 苏州 215123)

通讯作者: 付明, E-mail: fuming@ustc.edu.cn

摘 要: 形式化验证方法被认为是一种构建高可信软件系统的有效手段.在定理证明工具通过手动写证明脚本来验证系统软件的功能正确性,这种验证方式表达力强,可以证明复杂系统,但是自动化程度低、验证代价比较高;而使用程序验证器接受经过规范标注的源代码生成验证条件,并将验证条件交给约束求解器自动求解,这种方式自动化程度高,缺点在于它很难验证复杂系统软件的全部功能的正确性.结合上述两种方式的优点,在定理证明工具 Coq 中实现了一个自动证明策略 smt4coq,它通过在 Coq 中调用约束求解器 Z3 自动证明 32 位机器整数相关的数学命题,提高了自动化验证的程度,减少了用户手动验证程序的开销.

关键词: 形式化验证;定理证明工具;约束求解器;Coq;Z3

中图法分类号: TP311

中文引用格式: 张恒若,付明.基于 Z3 的 Coq 自动证明策略的设计和实现.软件学报,2017,28(4):819–826. http://www.jos.org.cn/1000-9825/5196.htm

英文引用格式: Zhang HR, Fu M. Design and implementation of Coq tactics based on Z3. Ruan Jian Xue Bao/Journal of Software, 2017,28(4):819–826 (in Chinese). http://www.jos.org.cn/1000-9825/5196.htm

Design and Implementation of Coq Tactics Based on Z3

ZHANG Heng-Ruo[1], FU Ming[2,3]

[1](School of Information Science and Technology, University of Science and Technology of China, Hefei 230026, China)

[2](School of Computer Science, University of Science and Technology of China, Hefei 230026, China)

[3](Software Security Laboratory, Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou 215123, China)

Abstract: Formal verification is an effective approach to construct high confidence software. Verifying the functional correctness of complex system software by manually writing proof scripts in proof assistant tools is feasible with the low degree of automation, and the verification cost is relatively high. The automatic program verifiers verify programs by taking the annotated source code as their input to generate verification conditions automatically solved by SMT solvers. This approach has a high degree of automation, but it is impossible to verify the functional correctness of the entire system software. By combining the advantage of the above two methods, this paper implements a novel Coq tactic plug-in named “smt4coq”, which allows calling the Z3 SMT solver in Coq to automatically prove mathematical propositions involved with 32-bit machine integers. The new tactic improves the degree of automation and reduces the cost of manual verification.

Key words: formal verification; proof assistant tool; SMT solver; Coq; Z3

---

* 基金项目: 国家自然科学基金(61103023, 61229201, 61379039, 91318301, 61632005)

Foundation item: National Natural Science Foundation of China (61103023, 61229201, 61379039, 91318301, 61632005)

收稿时间: 2016-06-20; 修改时间: 2016-09-08; 采用时间: 2016-11-26; jos 在线出版时间: 2017-01-24

CNKI 网络优先出版: 2017-02-20 13:43:28, http://www.cnki.net/kcms/detail/11.2560.TP.20170220.1343.003.html

作为信息产业的核心技术,软件系统被广泛地应用于航空航天、核电、军工、高铁、医疗、金融、自动驾驶等安全攸关的国家战略基础设施和人们的日常生活中.高可信软件系统成为保障国家安全、保持经济可持续发展、维护社会稳定和保护人民生命财产安全的必要条件.

近几年来,形式化验证方法被认为是一种构建高可信软件系统的有效手段,相关的验证理论和支撑工具都有了较大的发展.在验证理论方面,程序精化验证理论[1−5]能够支持复杂并发系统的验证;在支撑工具方面,形式化证明工具 Coq[6]、Isabelle[7]、程序验证器 VCC[8]、Dafny[9]和约束求解器 Z3[10]也逐步成熟.这些理论技术的发展给安全攸关系统软件的形式化验证提供了理论基础和工具保障,同时也出现了大量的系统软件验证项目,譬如,在操作系统验证方面,有 seL4[11],VeriSoft[12],CertiKOS[13],CertiuCOS[14];在编译器验证方面有 CompCert[15].

上述系统软件验证项目采用的验证方法主要分为两类,其中一种方式是利用形式化证明工具手动编写证明脚本来完成系统软件的验证,这种验证方式的优势在于形式化证明工具能够为系统软件的验证过程产生证据,同时,所依赖的信任基(TCB)较小,手动干预使得完成复杂系统软件的验证成为可能;缺点在于自动化程度低、验证代价比较高.如 seL4 操作系统,被证明的源代码(C 语言)有约 8 000 行,而验证代码(Isabelle)有约 200 000 行,是源码的 25 倍多[11].另外一种方式是使用程序验证器接受经过规范标注的源代码生成验证条件,并将验证条件交给约束求解器自动求解,这种方式的优点在于强大的约束求解器极大地提高了验证过程的自动化程度,缺点在于它很难完成复杂系统软件功能正确性的全部验证.因为操作系统内核数据结构之间的关系复杂,对应的约束条件也会十分复杂,使得约束求解器无法在有效的时间范围内对其进行判定.因此,复杂系统软件的功能正确性验证目前几乎不可能全部自动化,人为干预不可避免.如何减少手动验证工作的代价,是构建高可信软件亟待解决的问题.

在定理证明工具中,验证操作系统内核的功能正确性的方法已被广泛采用[11−14].为了提高验证效率,都需要开发一些辅助的自动验证工具.其中,CertiuCOS[14]项目通过在定理证明工具 Coq 中使用其提供的 Ltac 策略编程语言开发了大量的自动证明策略来提高验证的效率,将验证脚本行数与被验证代码行数的比例从几百比一减少到了 26:1.这些证明策略主要用来自动生成验证条件,并自动证明验证条件中分离逻辑断言[16]间的蕴含关系,产生的一些描述操作系统内核功能正确性的纯数学命题的绝大部分则交给用户自己手动证明.由于 Coq 提供的一些自动证明策略(如 omega,ring)功能很有限,导致这些与程序上下文无关的纯数学命题的手动证明代价较高,整个项目证明脚本代码量约 21 万行,其中 30%的证明脚本是用来手动证明这些纯数学命题的.而这些纯数学命题很多是可以被 Z3 直接判定的,因此,如果能够在 Coq 中调用 Z3 来辅助判定这些纯数学的命题,则可以大大降低验证开销.

本文在 Coq 中实现了一个证明策略 smt4coq,能够在 Coq 中自动证明 32 位机器整数相关的纯数学命题.图 1 给出了 smt4coq 证明策略的使用效果示例,目标是证明定理$\forall x\in$Int32,$\forall x\in$Int32,$x\oplus y=0\rightarrow x=y$.其中,图 1(a)是用 Coq 提供的基础证明策略和相关的其他定理(*same_bits_eq*,*bits_xor*,*biths_zero*)的证明示例,而图 1(b)所示为直接使用本文开发的 smt4coq 策略进行自动证明.

```
Theorem xor_zero_equal:
forall x,y,
xor x,y=zero→x=y.
Proof.
intros.apply same_bits_eq.
intros.assert(xorb(testbit x,i)(testbit y,i)=false).
rewrite←bits_xor; auto.
rewrite H.apply biths_zero.destruct(testbit x,i).destruct(testbit y,i).
reflexivity||discriminate.
Qed.
```

(a) 原始证明代码

```
Theorem xor_zero_equal: forall x,y,
xor x,y=zero→x=y.
Proof.
smt4coq.
Qed.
```

(b) 使用 smt4coq 的证明代码

Fig.1 An example using smt4coq

图 1 smt4coq 策略使用效果示例

© 中国科学院软件研究所 http://www.jos.org.cn

本文做出如下几点贡献:

1) 设计了一个编译框架,支持将 Coq 中命题的语法形式翻译成可被 Z3 判定的命题,为利用 Z3 自动证明 Coq 中的定理奠定基础;
2) 实现了一个 Coq 的自动证明策略 smt4coq,它能够在 Coq 中调用 Z3 证明 32 位机器整数相关的数学命题,包括 32 位整数上的基本运算、加减、移位等.有机地将定理证明工具与约束求解器结合在一起,取长补短.相关源码可以从地址 http://staff.ustc.edu.cn/~fuming/smt4coq.zip 下载获得;
3) 实验结果表明,使用证明策略 smt4coq 能够大幅度减少证明代码行数.

本文第 1 节介绍自动证明策略 smt4coq 的设计思想.第 2 节介绍策略 smt4coq.第 3 节介绍对 32 位机器整数相关命题的翻译过程的实验结果.最后比较相关工作并总结全文和将来的工作.

1 smt4coq 自动证明策略的设计

1.1 smt4coq的设计思想

smt4coq 是在 Coq 中开发的一个证明策略插件,它通过调用功能强大的 Z3 约束求解器来判定一些相对复杂的数学命题,而这些数学命题在 Coq 中可能需要大量的手动证明来完成.图 2 给出了 smt4coq 自动证明策略的算法设计思想.

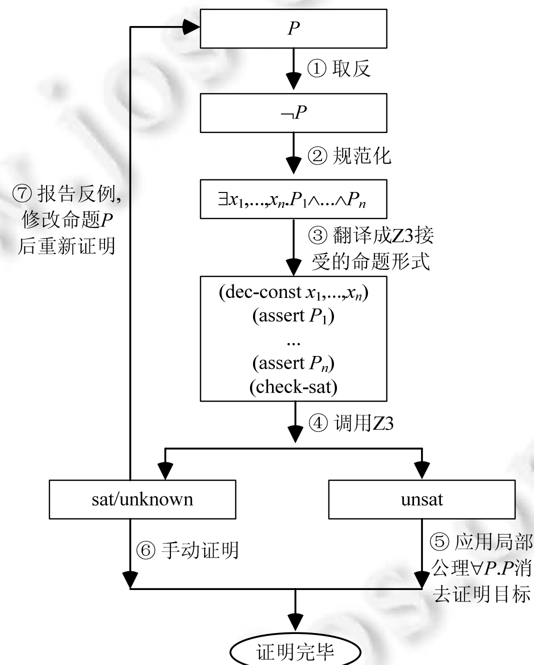

Fig.2 smt4coq's algorithm

图 2 smt4coq 算法流程

对于给定的一个命题 P,自动证明策略 smt4coq 将经过如下步骤完成对该命题的证明.

- 第 1 步:首先对命题 P 取反,通过调用 Z3 判定 $\neg P$ 是不可满足的(unsat),从而得到命题 P 是成立的.这里,对命题 P 取反为接下来的命题规范化做准备;
- 第 2 步:命题规范化,它的目标是将取反后的命题转化为形如 $\exists x_1,\ldots,x_n.P_1\wedge\ldots\wedge P_n$ 的命题,为翻译成为 Z3 所接受的命题语法形式做准备.例如:一个形如 $\forall x.P(x)\rightarrow Q(x)$ 的命题,经过取反 $\neg(\forall x.P(x)\rightarrow Q(x))$.这里,$P$ 和 Q 是原子命题,而该命题等价于 $\exists x.\neg(\neg P(x)\vee Q(x))$.然后,根据德摩根定律可以得到规范化的命

© 中国科学院软件研究所 http://www.jos.org.cn

题形式$\exists x.P(x)\wedge\neg Q(x)$:如果该命题不可满足,则说明原命题成立;否则,存在一个反例让 P 不满足;

- 第 3 步:规范化 Coq 命题到 Z3 命题的翻译,经过规范化后的命题和 Z3 接受的命题形式有着直接的对应关系,上述例子中,规范化的命题$\exists x.P(x)\wedge\neg Q(x)$所对应的 Z3 的命题为

  (*declare-const x*)(*assert P*(*x*))(*assert* ¬*Q*(*x*))
  (*check-sat*);

- 第 4 步:将翻译后的命题作为 Z3 的输入,调用 Z3 进行判定:如果返回结果为不可满足(unsat),则进入第 5 步;否则,用户可以手动证明(第 6 步)或者根据报告的反例修改输入命题重新证明(第 7 步);
- 第 5 步:应用局部公理消去 Coq 上下文中当前的证明目标,这里,虽然命题 P 经过 Z3 的判定是正确的,但是并没有在 Coq 中产生相应的证明项,因此,我们通过引入一个局部公理$\forall P.P$(任何命题都是正确的)在 Coq 中消去当前证明目标,这个公理只允许在调用 Z3 返回 unsat 后被使用.

这里,我们通过对 Coq 命题取反和规范化预处理后再翻译成 Z3 的所接受的命题,而没有选择直接将其翻译成一条 Z3 所能接受的命题,主要有以下两个原因:第一,当 Z3 接受的命题过于复杂时,证明时间会很长甚至根本无法证明(比如在 Z3 中无法量化函数,只能通过 *declare-fun* 语句引入);第二,直接翻译得到的命题会很长且只有一句,当翻译的目标语句有问题时,Z3 的出错信息只会显示在那唯一一行有错误,不利于开发过程中的定位错误.

1.2 Coq表达式到Z3语句的翻译

规范化后的命题被分为前提条件和目标.在 Coq 中,前提中的变量和命题都是同等的表达式,但在 Z3 中,这是两种截然不同的表达式.smt4coq 根据表达式的类型来区分,类型为 Set 的表达式是变量,类型是 Prop 的表达式是命题.这样,表达式已被划分为变量、命题和目标(目标是特殊的一个命题).接下来要把这些表达式转化成接近 Z3 语法的中间表达式.定义中间表达式语法如下:

⟨*stmt*⟩ ::=⟨*define-data*⟩|⟨*define-fun*⟩|⟨*declare*⟩|⟨*assert*⟩
⟨*define-data*⟩ ::=⟨*data-args*⟩⟨*sub-data*⟩
⟨*sub-data*⟩ ::=⟨*name*⟩⟨*constructors*⟩⟨*sub-data*⟩|*null*
⟨*define-fun*⟩ ::=⟨name⟩⟨*input-args*⟩⟨*output-arg*⟩⟨*expr*⟩
⟨*declare*⟩ ::=⟨*name*⟩⟨*declare*⟩|⟨*name*⟩
⟨*assert*⟩ ::=⟨*forall-expr*⟩|⟨*expr*⟩
⟨*expr*⟩ ::=⟨*app-expr*⟩|⟨*atom*⟩

stmt 表示 Z3 的一条语句,根据其作用,分为数据定义语句(define-data)、函数定义语句(define-fun)、变量声明语句(declare)和命题语句(assert).

目前,smt4coq 支持翻译归纳类型表达式(非递归类型)、箭头类型表达式、全称命题表达式、函数调用表达式、常数符号表达式和构造子表达式.其中,遇到新的归纳类型、构造子或常数符号时,首先查找符号映射表:如果不在表中,则根据其类型声明新的类型定义或函数定义;如果出现了目前未定义的表达式,则报错.

中间代码本身已经很接近 Z3 的代码了,但翻译中需注意一些细节.比如,Z3 只支持特定的泛型函数(比如“等于”),遇到其他的泛型函数要报错.另外,Z3 不支持递归函数的定义,需要把递归函数转化为等价的命题形式.

通过使用配置文件增加可扩展性,smt4coq 将部分数据结构与函数采用直接映射的方式翻译,这个映射储存在配置文件里,可由用户自定义.考虑到在执行策略时配置映射表会大大拖慢证明速度,所以把映射表的配置放到编译时,每次用户更新配置文件后都要重新编译一遍插件.选用直接映射而不是翻译,主要是出于以下几点考虑.

1) Coq 与 Z3 类型系统差异巨大,很多表达式无法精确翻译.Coq 支持归纳证明、高阶函数、全递归函数,然而这些在 Z3 中都无法直接定义;
2) 很多数据类型在 Coq 中是递归定义的,但在 Z3 中有对应的基本类型.比如 Z(整数),在 Coq 中是用二进制的方式递归定义的,但在 Z3 中有对应的 Int 类型,如果在 Z3 中也递归定义整数,那么证明的效率会大

© 中国科学院软件研究所 http://www.jos.org.cn

为降低;

3) Z3 对函数的限制非常多,比如高阶函数与递归函数都无法直接定义.为了方便、快速,有些 Z3 中存在的函数,直接映射相较于重新构造是一个更实际的选择.

1.3 调用Z3判定命题是否正确

调用 Z3 判定翻译后的命题,并接收输出结果.如果结果是 sat 或 unknown,说明证明错误或出现异常,插件会报错,转为用户手动处理;如果结果是 unsat,说明证明成功.由于该策略调用外部程序 Z3,虽然 Z3 判定命题为真,但是 Coq 内部并没有生成对应的证明项并储存证明.为了解决这个问题,创建一个 Coq 源文件如下:

Local Axiom by_smt: *forall P*:*Prop*,*P*.

Ltac smt4coq:=*intros*;*smt*;*apply by_smt*.

之后,编译该文件成独立模块.这样对用户就封装了一个黑盒,用户只需使用 smt4coq 策略就可自动证明相关定理.局部公理 *by_smt* 对其他文件不可见,可以防止用户滥用这个公理而产生错误的证明.

2 smt4coq 自动证明策略的实现

smt4coq 证明策略目前实现了对整数、布尔值以及机器整数的有关命题的翻译和自动化证明.下面介绍插件源码结构以及具体的实现.

2.1 源码结构

smt4coq 证明策略通过扩展 Coq 定理证明工具的源代码,使用 OCaml 编写新的证明策略插件模块来实现.表 1 给出了 smt4coq 证明策略插件的源码详细描述.其中,配置文件 config.ml 允许增加 Coq 和 Z3 之间新的符号映射关系,从而可以支持更多的 Coq 命题到 Z3 命题之间的翻译变换,目前只建立了 Coq 的 32 位机器整数所涉及到的符号到相应的 Z3 符号之间的映射关系(因为 CertiuCOS 项目只用到了 32 位整数,但是可以扩展到任意长度的整数).

Table 1 List of smt4coq source files

表 1 smt4coq 源码文件列表

源码文件名	功能描述	代码行数
config.ml	配置文件,用户手动定义 Coq 和 Z3 之间的符号映射	54
solver.ml	smt4coq 证明策略对应的入口.内部有 *print_coq*,*print_Z3* 和 *solve* 这 3 个函数,对应输出 Coq 内部表达式、输出翻译后的 Z3 表达式和自动证明这 3 个功能	75
trans.ml	翻译功能的主体	172
ztermops.ml	Coq 源码文件 termops.ml 的改动版,存放表达式相关的操作	1 029
ztypeops.ml	Coq 源码文件 typeops.ml 的改动版,存放类型相关的操作	567
smt4coq.ml4	策略定义文件.内含 3 个策略——printCoq,pringZ3 和 smt,对应 solver.ml 中的 *print_coq*,*print_Z3* 和 *solve* 这 3 个函数	22
smt4coq.mllib	源码目录文件.因为在 Coq 中同时只能引用一个 ML 模块,所以需要源码目录文件记录其他模块的名字.OCaml 编译器会把目录文件打包成一个静态库	7
SMT.v	引入 OCaml 模块并定义相关公理和策略,被编译成一个 Coq 模块作为封装好的黑盒	8
Demo.v	示例项目	38
zmake	项目构建脚本	3
总计		1 975

2.2 配置文件的描述

配置文件实际上也是一个 OCaml 源码文件.由于 Coq 没有提供让插件直接修改证明环境外内存的接口,所以如果在运行时读取配置文件,那么读取的配置信息退出证明就被销毁,无法储存在内存中.这样,每次运行插件都要重新读取配置文件,极大地影响了运行速度.因此,本插件使用修改配置文件再重新编译的方法来配置插件.配置文件主要的部分是一个 Hash 表,键是 Coq 的表达式名称,值是对应的 Z3 函数,每次翻译表达式时先查表,如果有对应的值就直接输出.该配置文件只提供一个简单映射功能,实现对 Coq 内置的整数运算(add,sub,mul

© 中国科学院软件研究所 http://www.jos.org.cn

等)和布尔值运算(andb,orb 等)的映射,以及对用户定义的机器整数的映射.其他复杂的函数翻译、命题翻译是由插件自动完成,无需配置.采用配置文件的原因是 Z3 中内置了整数、布尔值和机器整数类型及其运算,如果按照翻译普通类型定义和函数定义的方法翻译过去,会运行得非常缓慢,不如直接采用映射的方式.但对于其他的表达式,Z3 并没有内置,就只能采用第 1.2 节所述的过程来翻译.

2.3 确保用户配置文件的正确性

从概率上说,用户错误地配置映射关系造成错误证明几乎不可能发生.原因如下.

1) smt4coq 基于 Z3 实现自动证明,所以需要映射的表达式只有 Z3 内置的类型与函数,即只映射整数、布尔值以及机器整数这 3 种类型和相关操作.而这 3 种数据类型及其操作都非常基础,而且特征性强,用户很难配置错误;
2) 在 Z3 中,相应函数都有特定的类型限制,用户如果映射了错误的函数,极有可能会报错.比如,如果用户将 Z3 中的加法 add 映射到 Coq 中的 not 函数,那么就会出现参数数量错误以及类型不匹配的问题而报错;
3) 基本运算都是成套出现的,用户需要同时映射加减乘除这些函数,如果有一个映射错误,那会产生结果的不一致,很容易发现错误.比如,如果把加法映射成乘法,那么证明加法是减法逆运算时就会报错,而这显然是不正确的.

2.4 机器整数类型及运算的翻译

机器整数在 Coq 中被定义为一个结构体,成员包括一个 Z 类型整数和一个整数大小范围的限制.而 Z3 中内置了位矢量类型和对应的位矢量操作,所以可以直接把 Coq 中的 *x* 位机器整数映射为 *x* 位矢量(_*BitVec x*)类型,原来定义的整数运算,比如 and,or,add,mul 等,可以一并替换为 Z3 内置的位矢量运算 bvand,bvor,bvadd,bvmul 等(这些操作对任意长度的机器整数都适用).表 2 列出了部分位矢量函数.

Table 2 List of common functions for the bit vector

表 2 常用位矢量函数表

bvadd	加法	bvsub	减法	bvmul	乘法
bvsdiv	有符号除法	bvsrem	有符号求余数	bvsmod	有符号取模
bvsgt	有符号大于	bvsge	有符号大于等于	bvslt	有符号小于
bvsle	有符号小于等于	bvor	按位或	bvand	按位与
bvnot	按位取反	bvnand	按位与非	bvxor	按位异或
bvshl	左移	bvlshr	逻辑右移	bvashr	算术右移
bvneg	取相反数	bv2int	表示成整数	(_ int2bv n)	表示成位矢量

表 3 中,以命题$\forall x\in$Int32,$\forall x\in$Int32,$x\oplus y=0\rightarrow x=y$ 为例,给出了 Coq 中 32 位机器整数的语句与 Z3 语句之间的对应关系.

Table 3 Example of the comprison of Coq and Z3's expressions

表 3 Coq 语句与 Z3 语句的对应示例

Coq 语句	Z3 语句
x: Int32	(*declare-const x* (_*BitVec* 32))
y: Int32	(*declare-const y* (_*BitVec* 32))
H: *xor x,y=zero*	(*assert* (=*bvxor x,y*) (#*x*00000000))
Goal: *x*=*y*	(*assert* (not (=*x,y*)))

3 实验结果

CertiuCOS 操作系统内核验证项目[14]和 CompCert 验证编译器[15]底层都是使用相同的 32 位机器整数的 Coq 实现(Integer.v 文件),在未使用自动化证明策略之前,每个定理都要手工一点点构造证明,证明动辄上百行,而且还要证明很多引理,原来共计 4 477 行代码.使用 smt4coq 自动化证明策略之后,绝大多数的定理都只需一条

© 中国科学院软件研究所 http://www.jos.org.cn

策略即可证明,代码量下降到 1 832 行.表 4 给出了 Integer.v 中的部分定理在使用 smt4coq 前后证明策略数的对比情况,结果表明:smt4coq 可以自动证明这类 32 位机器整数相关的数学命题,大大提高了证明效率.

Table 4 The comparison of numbers of tactics after and before using smt4coq tactic

表 4 使用 smt4coq 策略前后证明策略数对比

定理名	使用前策略数	使用后策略数
bits_size	8	1
size_range	8	1
no_overlap_sound	16	1
lt_sub_overflow	20	1
notbool_istrue_isfalse	6	1
translate_cmp	4	1
rolm_rolm	7	1
ror_rol	15	1
bits_ror	38	1

我们还使用 smt4coq 测试了 Software Foundations(一个交互式学习 Coq 的电子书)中的 Basic.v,Logic.v,Ind.v 等文件中的部分命题,将很多冗长的证明都缩减为 1 行,使总共 2 681 行的证明缩减为 962 行(还有很多命题关于复杂的递归数据类型及高阶函数,无法证明).

4 相关工作比较和结论

目前有两个类似的工具:SmtCoq[17]和 Coqsmtcheck[18].SmtCoq 是在 Coq 中实现了一个约束求解器,它本身的功能没有 Z3 强大,导致不能自动证明本文中与 32 位机器整数相关的命题.coqsmtcheck 只能证明关于实数相关的简单数学命题,不支持 32 位机器整数相关命题的证明,且难以扩展.本文在定理证明工具 Coq 中开发了一个自动证明策略插件 smt4coq,它通过将经过翻译变换的 Coq 命题输入给约束求解器 Z3,然后使用 Z3 进行自动判定来提高证明效率.目前,smt4coq 只支持对整数、布尔值以及机器整数相关命题的翻译和证明,将来会对它进行扩展,以支持更多数据类型的翻译和相关命题的证明.smt4coq 发挥了 Z3 的自动证明的优势,但其也继承了 Z3 的不足,无法像 Coq 那样产生命题对应的证明项.为了弥补这一不足,将来可以考虑在 Coq 中实现 Z3 的某些判定算法,同时产生证明项.

References:

[1] De Roever WP, Engelhardt K, Buth KH. Data Refinement: Model-Oriented Proof Methods and Their Comparison, Number 47. Cambridge University Press, 1998.

[2] Liang HJ, Feng XY, Fu M. Rrely-Guarantee-Based simulation for verifying concurrent program transformations. In: Proc. of the POPL. 2012. 455−468. [doi: 10.1145/2103656.2103711]

[3] Liang HJ, Feng XY. Modular verification of linearizability with non-fixed linearization points. In: Proc. of the PLDI. 2013. 459−470. [doi: 10.1145/2491956.2462189]

[4] Liang HJ, Hoffmann J, Feng XY, Shao Z. Characterizing progress properties of concurrent objects via contextual refinements. In: Proc. of the CONCUR. 2013. 227−241. [doi: 10.1007/978-3-642-40184-8_17]

[5] Liang HJ, Feng XY, Shao Z. Compositional verification of termination-preserving refinement of concurrent programs. In: Proc. of the CSL-LICS. 2014. 65:1−65:10. [doi: 10.1145/2603088.2603123]

[6] The Coq Development Team. The Coq Proof Assistant. http://coq.inria.fr

[7] Nipkow T, Wenzel M, Paulson L. Isabelle/HOL: A Proof Assistant for Higher-Order Logic. Berlin, Heidelberg: Springer-Verlag, 2002.

[8] Ernie C, Markus D, Mark H, Dirk L, Michał M, Thomas S, Wolfram S, Stephan T. VCC: A practical system for verifying concurrent C. In: Proc. of the TPHOLs. 2009. 23−42.

[9] Leino KRM. Dafny: An automatic program verifier for functional correctness. In: Proc. of the Logic for Programming, Artificial Intelligence, and Reasoning. Springer-Verlag, 2010. 348−370. [doi: 10.1007/978-3-642-17511-4_20]

© 中国科学院软件研究所 http://www.jos.org.cn

[10] De Moura L, Bjørner N. Z3: An efficient SMT solver. In: Proc. of the Tools and Algorithms for the Construction and Analysis of Systems. Springer-Verlag, 2008. 337−340. [doi: 10.1007/978-3-540-78800-3_24]

[11] Klein G, Elphinstone K, Heiser G, Andronick J, Cock D, Derrin P, Elkaduwe D, Engelhardt K, Kolanski R, Norrish M, Sewell T, Tuch H, Winwood S. seL4: Formal verification of an OS kernel. In: Proc. of the 22nd ACM SIG OPS Symp. on Operating Systems Principles. ACM Press, 2009. 207−220. [doi: 10.1145/1629575.1629596]

[12] Godefroid P. Software model checking: The VeriSoft approach. Formal Methods in System Design, 2005,26(2):77−101. [doi: 10.1007/s10703-005-1489-x]

[13] Gu L, Vaynberg A, Ford B, Zhong S, Costanzo D. CertiKOS: A certified kernel for secure cloud computing. In: Proc. of the 2nd Asia-Pacific Workshop on Systems. ACM Press, 2011. 3. [doi: 10.1145/2103799.2103803]

[14] Xu FW, Fu M, Feng XY, Zhang XR, Zhang H, Li ZH. A practical verification framework for preemptive OS kernels. Computer Aided Verification, 2016. [doi: 10.1007/978-3-319-41540-6_4]

[15] Leroy X. Formal verification of a realistic compiler. Communications of the ACM, 2009,52(7):107−115. [doi: 10.1145/1538788.1538814]

[16] Reynolds JC. Separation logic: A logic for shared mutable data structures. In: Proc. of the Logic in Computer Science. 2002. [doi: 10.1109/LICS.2002.1029817]

[17] SMTCoq. https://smtcoq.github.io/

[18] coqsmtcheck. https://github.com/gmalecha/coq-smt-check

张恒若(1993－),男,山东烟台人,助理研究员,主要研究领域为形式化方法.

付明(1982－),男,博士,副教授,CCF 专业会员,主要研究领域为程序验证,操作系统内核验证,并发理论,程序设计语言理论.

www.jos.org.cn
www.jos.org.cn