ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.1, January 2008, pp.17−26
DOI: 10.3724/SP.J.1001.2008.00017

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于抽象解释理论的程序验证技术*

李梦君[1+], 李舟军[2], 陈火旺[1]

[1](国防科学技术大学 计算机学院,湖南 长沙 410073)

[2](北京航空航天大学 计算机科学与工程学院,北京 100083)

Program Verification Techniques Based on the Abstract Interpretation Theory

LI Meng-Jun[1+], LI Zhou-Jun[2], CHEN Huo-Wang[1]

[1](School of Computer, National University of Defense Technology, Changsha 410073, China)

[2](School of Computer Science and Engineering, BeiHang University, Beijing 100083, China)

+ Corresponding author: Phn: +86-731-4576468, Fax: +86-731-4576468, E-mail: mengjun_li1975@yahoo.com.cn

Li MJ, Li ZJ, Chen HW. Program verification techniques based on the abstract interpretation theory. *Journal of Software*, 2008,19(1):17−26. http://www.jos.org.cn/1000-9825/19/17.htm

Abstract: The abstract interpretation theory was proposed by P. Cousot and R. Cousot in 1977, and is widely used in the program's static analysis domain to construct and approximate the program's fixpoint semantics. This paper describes the abstract interpretation framework of the program semantics based on the Galois connection, and then discusses three typical applications of the abstract interpretation theory: The program transformation, the program verification techniques about the safety property and the program verification techniques about the liveness property. Finally, it points out the main directions of the program verification techniques based on the abstract interpretation theory.

Key words: abstract interpretation theory; Galois connection; program verification

摘 要: 抽象解释(abstract interpretation)理论是 Cousot.P 和 Cousot.R 于 1977 年提出的程序静态分析时构造和逼近(approxiamation)程序不动点语义的理论.描述了程序语义基于 Galois 连接的抽象解释理论框架,讨论了基于抽象解释理论的程序变换、程序安全性验证和活性性质验证这 3 种典型的应用,并指出了基于抽象解释理论的程序验证的主要研究方向.

关键词: 抽象解释理论;Galois 连接;程序验证

中图法分类号: TP301 **文献标识码**: A

实践证明,形式化方法是分析软件和硬件系统中"bug"的一种有效手段.模型检验、定理证明是能够自动化实现的典型形式化方法.模型检验通过遍历系统所有状态空间,能够对有穷状态系统进行自动验证,并自动构造不满足验证性质的反例.对于无穷状态系统或状态数目超出现有计算机处理能力的非常复杂的有穷状态系统,模型检验方法难以解决状态空间爆炸问题.定理证明方法能够应用于解决无穷状态系统的验证问题,但是,一阶逻辑是半可判定的.理论分析结果表明,机械化的定理证明过程并不能保证停机.计算机科学理论和大规模软、

* Supported by the National Natural Science Foundation of China under Grant Nos.60473057, 90604007 (国家自然科学基金)
Received 2006-11-14; Accepted 2007-04-25

硬件系统的发展需要一种能够处理计算机科学中不可判定问题和非常复杂问题的有效方法,抽象解释理论[1]就是其中一种有效的解决方法,它为计算机科学中的不可判定问题和复杂问题的逼近求解提供了系统性的构造方法和有效算法.

抽象解释理论是 P. Cousot 和 R. Cousot 于 1977 年提出的程序静态分析时构造和逼近程序不动点语义的理论.程序指称对象域上的计算,程序语义(操作语义或者指称语义)描述了对象域上的计算过程或结果.程序的抽象解释就是指使用另一个抽象对象域上的计算抽象逼近程序指称的对象域(具体对象域)上的计算,使得程序抽象执行的结果能够反映出程序真实运行的部分信息.抽象解释本质上是在计算效率和计算精度之间取得均衡,以损失计算精度求得计算可行性,再通过迭代计算增强计算精度的一种抽象逼近方法.基于抽象解释理论的形式化方法是对大规模软、硬件系统进行自动化分析与验证的有效途径之一,已被广泛地应用于大型软、硬件系统的验证研究中[2−8].

本文第 1 节描述完备格、Galois 连接、完备格上的 Widening/Narrowing 算子等基本概念,以及程序语义基于 Galois 连接的抽象解释理论框架.第 2 节主要集中于描述基于抽象解释的典型应用研究,主要包括基于抽象解释的程序变换、基于抽象解释的程序安全性性质验证和程序活性性质验证.第 3 节对全文进行总结.

1 程序不动点语义基于 Galois 连接的抽象解释理论框架

Galois 连接、完备格上的 Widening/Narrowing 算子是抽象解释理论中的基本概念.Galois 连接中的具体算子保证抽象解释中抽象对象域和具体对象域之间逼近关系的可靠性,抽象算子建立两者之间的最优逼近关系.完备格上的 Widening 算子用于使程序不动点语义的计算过程收敛或用于加速其收敛,Narrowing 算子对程序不动点语义计算结果进行精化.程序通常描述为迁移系统,基于抽象解释理论对迁移关系进行抽象,可以构造具有不同精细程度的程序抽象语义.抽象解释理论从抽象对象域、Widening 算子、程序抽象语义 3 个方面建立起程序不动点语义抽象逼近计算的理论框架.

基于抽象解释理论的程序验证的基本思想是,在保证抽象逼近可靠性的前提下,对程序不动点语义计算过程的各个层次进行抽象逼近,使得计算过程收敛或加速其收敛,然后依据逼近计算的不动点语义验证系统需要满足的性质,若验证过程中出现虚假反例,则对抽象过程进行精化.基于抽象解释理论的形式化验证的特点是可靠但不完备的,如果验证结果表明抽象系统满足验证性质,则抽象解释理论保证原系统也满足验证性质;若验证结果表明抽象系统不满足验证性质,则原系统可能满足验证性质(表明抽象过程中出现了不满足验证性质的虚假反例),可能不满足验证性质(原系统存在不满足验证性质的真实反例),需要更精细的抽象分析.

1.1 Galois连接与完备格上的Widening/Narrowing算子

定义 1(完备格). 设$\langle L,\sqsubseteq\rangle$是一个偏序,$\langle L,\sqsubseteq\rangle$称为一个完备格当且仅当:$L$ 中任何一个子集均存在最小上界和最大下界.特别地,用$\bot=\sqcup\varnothing=\sqcap L$ 表示 L 中最小元素,用$\top=\sqcap\varnothing=\sqcup L$ 表示 L 中最大元素,其中,$\sqcup$和$\sqcap$表示最小上界算子和最大下界算子,完备格$\langle L,\sqsubseteq\rangle$一般表示为$\langle L,\sqsubseteq,\sqcup,\sqcap,\bot,\top\rangle$.

状态空间爆炸是程序验证过程中需要解决的主要问题.程序验证需要检查程序中每一条计算路径是否满足验证性质,程序指称的对象域本身的复杂性是引起程序验证过程中状态空间爆炸的原因之一,抽象解释理论使用另一个抽象对象域上的计算抽象程序指称的对象域上的计算.一般来说,抽象对象域上的计算比程序指称的对象域上的计算精确度要低,但却易于计算出与验证性质相关的不变式,计算复杂度低,易于自动化实现.区间抽象域[9]、八边形抽象域[10]、多面体抽象域[11]是常用的 3 种典型抽象对象域.

定义 2(Galois 连接). 设$\langle P,\leq\rangle$和$\langle Q,\sqsubseteq\rangle$是两个偏序集合,$\alpha:P\to Q$ 和$\gamma:Q\to P$ 是两个映射,序偶$\langle\alpha,\gamma\rangle$称为一个 Galois 连接当且仅当:$\forall x\in P,\forall y\in Q,\alpha(x)\sqsubseteq y$ 当且仅当 $x\leq\gamma(y)$,其中,映射α称为抽象算子,γ称为具体算子.

由 Galois 连接的定义,容易验证抽象算子α和具体算子γ具有如下性质:

(1) $\forall x\in P,x\leq\gamma(\alpha(x))$;

(2) $\forall y\in Q,\alpha(\gamma(y))\sqsubseteq y$;

(3) α和γ单调递增.

抽象算子α和具体算子γ具有的上述性质与 Galois 连接的定义等价.可以从上界和下界两个方向抽象逼近对象集合.抽象偏序集$\langle Q,\sqsubseteq\rangle$中的对象集合 p 可以上界抽象逼近具体偏序集$\langle P,\leq\rangle$中的对象集合 P'当且仅当$P'\leq\gamma(p),\gamma(p)$是 P'的上界保证抽象逼近的可靠性;理论上,P'存在最优上界抽象逼近$\cap\{p|P'\leq\gamma(p)\}$.抽象算子$\alpha$刻画了具体对象域和抽象对象域之间的最优上界抽象逼近映射关系,即抽象分析时$\alpha(P')$作为 P'的抽象逼近.首先,$\alpha(P')$是 P'的一个上界抽象逼近($\forall x\in P,x\leq\gamma(\alpha(x))$);其次,若 p 是 P'的任意一个上界抽象逼近,即 $P'\leq\gamma(p)$.由 Galois 连接的定义,则有$\alpha(P')\sqsubseteq p(\forall x\in P,\forall y\in Q,x\leq\gamma(y)$当且仅当$\alpha(x)\sqsubseteq y)$,表明$\alpha(P')$是 P'的最优上界抽象逼近.例如,集合构成的无穷序列{1},{1,3},{1,3,5},…,{1,3,5,…,2n+1},…与区间抽象域之间的最优抽象逼近关系由 Galois 连接$\langle\alpha,\gamma\rangle$确定,其中,$\alpha(\{1,3,5,\ldots,2n+1\})=[1,2n+1],\gamma([1,2m+1])=\{n|1\leq n\leq 2m+1\}$.

容易验证,Galois 连接的运算具有如下性质:

(1) 两个 Galois 连接的合成是 Galois 连接;

(2) Galois 连接的广义笛卡尔乘积是一个 Galois 连接;

(3) 设 P 表示一个程序,用 $S[[P]]$表示 P 的语义,如果 $S[[P]]$可以通过计算语义泛函 $F[[P]]$的最小不动点得到,即 $S[[P]]=lfpF[[P]]$,若$\langle\alpha,\gamma\rangle$是一个 Galois 连接,定义语义泛函 $F'[[P]]=\alpha\circ F[[P]]\circ\gamma$,则 $S[[P]]$的抽象$\alpha(S[[P]])=lfpF'[[P]]$.

Galois 连接的上述运算性质表明对程序语义可以连续地进行抽象,直到抽象对象域上的抽象计算满足计算复杂度的要求为止.

程序不动点语义的计算过程不收敛(不终止)是引起程序验证过程中状态空间爆炸的另一个原因.设$\langle L,\sqsubseteq,\sqcup,\sqcap,\bot,\top\rangle$是一个完备格,程序 P 的指称语义(操作语义与指称语义具有语义等价性)可以由 L 上的一个单调语义泛函 $F[[P]]:L\to L$ 来刻画,在最理想的情况下,即 $F[[P]]$是完备格 L 上的连续函数,并且$\sqcup\{F[[P]]^n(\bot)|n\in N\}$在有限步迭代计算后收敛,则容易得到 $F[[P]]$的最小不动点 $lfp(F[[P]])=\sqcup\{F[[P]]^n(\bot)|n\in N\}$;否则,$(\sqcup\{F[[P]]^n(\bot)|n\in N\})$不一定收敛,即使$(\sqcup\{F[[P]]^n(\bot)|n\in N\})$收敛,也不一定收敛于 $lfp(F[[P]])$(因为 $F[[P]]$不一定是连续函数).在抽象解释理论框架中,Widening 算子可以给出 $lfp(F[[P]])$的一个上界逼近,Narrowing 算子给出 $lfp(F[[P]])$更精细的一个上界逼近.

定义 3(Widening 算子). 设$\langle L,\sqsubseteq,\sqcup,\sqcap,\bot,\top\rangle$是一个完备格,算子$\nabla:L\times L\to L$ 称为 Widening 算子当且仅当:

(1) ∇是一个上界算子(即$\forall l_1,l_2\in L,l_1,l_2\sqsubseteq l_1\nabla l_2$);

(2) 对于任意的递增链$(l_n)_n$,递增链$(l_n^\nabla)_n$都收敛,其中,(l_n^∇)定义为:如果 n=0,则$(l_n^\nabla)=l_n$;否则,$l_n^\nabla=(l_{n-1}^\nabla)\nabla l_n$.

偏序集合可以扩张成为完备格,然后可以为完备格定义 Widening 算子.例如,区间抽象域可以扩张成为完备格,对于扩张得到的完备格$\langle L,\sqsubseteq,\sqcup,\sqcap,\bot,\top\rangle$,定义如下的 Widening 算子$\nabla$:

如果 $m_2<m_1$,并且 $n_2\leq n_1$,则$[m_1,n_1]\nabla[m_2,n_2]=[\bot,n_1]$;如果 $m_1\leq m_2$,并且 $n_1<n_2$,则$[m_1,n_1]\nabla[m_2,n_2]=[m_1,\top]$;

如果 $m_2<m_1$,并且 $n_1<n_2$,则$[m_1,n_1]\nabla[m_2,n_2]=[\bot,\top]$;如果 $m_1\leq m_2$,并且 $n_2\leq n_1$,则$[m_1,n_1]\nabla[m_2,n_2]=[m_1,n_1]$;对于区间构成的无穷递增链[0,1],[0,2],…,[0,n],…,Widening 算子∇使该递增链迅速收敛到$[0,\top]$.

设程序 P 的单调语义泛函 $F[[P]]:L\to L$ 定义在完备格$\langle L,\sqsubseteq,\sqcup,\sqcap,\bot,\top\rangle$上,并且$\nabla$是完备格$\langle L,\sqsubseteq,\sqcup,\sqcap,\bot,\top\rangle$上的一个 Widening 算子,定义序列 $(F[[P]]_\nabla^n)_n$ 为:

若 n=0,则 $F[[P]]_\nabla^n=\bot$;

若 n>0,并且 $F[[P]](F[[P]]_\nabla^{n-1})\sqsubseteq F[[P]]_\nabla^{n-1}$, 则 $F[[P]]_\nabla^n=F[[P]]_\nabla^{n-1}$;

否则, $F[[P]]_\nabla^n=F[[P]](F[[P]]_\nabla^{n-1})\nabla(F[[P]]_\nabla^{n-1})$.

可以证明序列$(F[[P]]_\nabla^n)_n$收敛,并且存在 m>0,使得 $F[[P]](F[[P]]_\nabla^m)\sqsubseteq F[[P]]_\nabla^m$,$F[[P]]$在 $F[[P]]_\nabla^m$ 处是收缩的(reductive),由 Tarski 不动点定理,$lfp(F[[P]])\sqsubseteq F[[P]]_\nabla^m$,即 $F[[P]]_\nabla^m$ 是 $lfp(F[[P]])$的一个上界逼近, $F[[P]]_\nabla^m$ 对 $lfp(F[[P]])$逼近的精度取决于 Widening 算子∇.

定义 4(Narrowing 算子). 设$\langle L,\sqsubseteq,\sqcup,\sqcap,\bot,\top\rangle$是一个完备格,算子$\Delta:L\times L\to L$ 称为 Narrowing 算子当且仅当:

(1) $\forall l_1,l_2\in L$,若 $l_2\sqsubseteq l_1$,则 $l_2\sqsubseteq(l_2\Delta l_1)\sqsubseteq l_1$;

(2) 对于所有的递降链$(l_n)_n$,递降链$(l_n^\Delta)_n$收敛;其中,(l_n^Δ)定义为:如果 $n=0$,则$(l_n^\Delta)=l_n$;否则,$(l_n^\Delta)=(l_{n-1}^\Delta)\Delta l_n$.

设Δ是完备格$\langle L,\sqsubseteq,\sqcup,\sqcap,\bot,\top\rangle$上的一个 Narrowing 算子,定义序列$([F[[P]]]_n^\Delta)$为:

若 $n=0$,则$([F[[P]]]_n^\Delta)=(F[[P]]_\nabla^m)$;

否则,$([F[[P]]]_n^\Delta)=([F[[P]]]_{n-1}^\Delta)\Delta F[[P]]([F[[P]]]_{n-1}^\Delta)$.

可以证明 $lfp(F[[P]])\sqsubseteq(F[[P]]^n(F[[P]]_\nabla^m))\sqsubseteq([F[[P]]]_n^\Delta)\sqsubseteq F[[P]]_\nabla^m$,并且$([F[[P]]]_n^\Delta)_n$收敛,即存在 $m'>0$,使得$([F[[P]]]_{m'}^\Delta)=([F[[P]]]_{m'+1}^\Delta)$.取$([F[[P]]]_{m'}^\Delta)$作为 $lfp(F[[P]])$的上界逼近,显然,它是比 $F[[P]]_\nabla^m$ 更精细的一个$lfp(F[[P]])$的上界逼近.

1.2 程序语义基于Galois连接的抽象解释理论框架

程序通常表示为迁移系统,迁移系统$\tau=\langle\Sigma,\Sigma_i,t\rangle$是一个三元组,其中,$\Sigma$为状态集合,$\Sigma_i\subseteq\Sigma$为初始状态集合,$t\subseteq\Sigma\times\Sigma$为状态之间的迁移关系.程序迁移关系的复杂性也是引起程序验证过程中状态空间爆炸的原因之一,对迁移关系进行抽象解释得到迁移系统的 4 种语义:部分踪迹语义、自反传递闭包语义、可达性语义、区间语义,这 4 种语义具有不同的抽象逼近层次.

(1) 迁移系统的部分踪迹语义

迁移系统τ的部分执行踪迹是状态的有穷序列 $s_0s_1\ldots s_n$,其中,$s_0\in\Sigma,\langle s_i,s_{i+1}\rangle\in t,i<n$.所有部分执行踪迹构成的集合称为$\tau$的部分踪迹语义,记作 Σ_τ^*,$\Sigma_\tau^*=\cup_{n\geq0}\Sigma_\tau^n$,其中,$\Sigma_\tau^0$表示长度为 0 的部分执行踪迹集合,即空集,$\Sigma_\tau^n$表示长度为 n 的部分执行踪迹的集合.部分踪迹语义精确地描述了迁移系统的行为.

(2) 自反传递闭包语义作为部分踪迹语义的抽象

在验证与中间计算步骤无关的程序性质时,部分执行踪迹语义过于精确,对迁移系统τ的部分执行踪迹集合 X 定义如下的抽象:$a(X)=\{\tilde{a}(\langle s_0s_1\ldots s_n\rangle)|s_0s_1\ldots s_n\in X\}$,其中,$\tilde{a}(\langle s_0s_1\ldots s_n\rangle)=\langle s_0,s_n\rangle$,即对 X 中的任意一条踪迹$\sigma=s_0s_1\ldots s_n$,$\tilde{a}$ 将迁移关系$\langle s_0s_1\ldots s_n\rangle$抽象为$\sigma$的初始状态 s_0 与最终状态 s_n 之间的迁移关系.由 $a(X)$的定义,对于迁移系统τ的部分踪迹语义 Σ_τ^*,$a(\Sigma_\tau^*)$是迁移关系 t 的自反传递闭包 t^*.设 Y 是初始状态和最终状态之间迁移关系构成的集合,定义$\gamma(Y)=\{s_0s_1\ldots s_n|\langle s_0,s_n\rangle\in Y\}$.容易证明,$a(X)\subseteq Y$ 当且仅当 $X\subseteq\gamma(Y)$,$\langle\alpha,\gamma\rangle$是 Galois 连接,表明自反传递闭包语义是部分踪迹语义的最优上界抽象逼近.

(3) 可达性语义作为自反传递闭包语义的抽象

迁移系统$\tau=\langle\Sigma,\Sigma_i,t\rangle$的可达性语义表示为$\{s'|\exists s\in\Sigma_i:\langle s,s'\rangle\in t^*\}$,是初始状态可达的所有状态组成的集合.定义$post[r]Z=\{s'|\exists s\in Z:\langle s,s'\rangle\in r\}$,$a^\bullet(Y)=post[Y]\Sigma_i=\{s'|\exists s\in\Sigma_i:\langle s,s'\rangle\in Y\}$,则迁移系统的可达性语义$\{s'|\exists s\in\Sigma_i:\langle s,s'\rangle\in t^*\}=post[t^*]\Sigma_i=a^\bullet(t^*)$,定义$\gamma(Z)=\{\langle s,s'\rangle|s\in\Sigma_i$ 且 $s'\in Z\}$,则 $a(Y)\subseteq Z$ 当且仅当 $Y\subseteq\gamma(Z)$,因而,$\langle\alpha,\gamma\rangle$是一个 Galois 连接,表明可达性语义是传递闭包语义的最优上界抽象逼近.

(4) 区间语义作为可达性语义的抽象

迁移系统$\tau=\langle\Sigma,\Sigma_i,t\rangle$的状态如果能用全序$<$排序,其中,$<$以$-\infty$和$+\infty$作为极值,或者更一般地,迁移系统$\tau=\langle\Sigma,\Sigma_i,t\rangle$的状态集合构成一个完备格,则$\tau$的区间语义为位于某一区间的可达状态集合.定义 $a^H(Z)=[\min Z,\max Z]$,$\min Z$ 和 $\max Z$ 分别是集合 Z 的最小值和最大值,定义$\gamma^H([l,h])=\{s\in\Sigma|l\leq s\leq h\}$,并定义抽象蕴涵$[l,h]\sqsubseteq[l',h']$为$\gamma^H([l,h])\subseteq\gamma^H([l',h'])$,则有 $a^H(Z)\sqsubseteq[l,h]$当且仅当 $Z\sqsubseteq\gamma^H([l,h])$,因而,$\langle a^H,\gamma^H\rangle$是一个 Galois 连接,表明区间语义是可达性语义的最优上界抽象逼近.

在抽象解释理论框架中,程序语义划分为标准语义、集合语义、抽象语义 3 个抽象层次,程序的标准语义是指程序的操作语义或指称语义,集合语义是对标准语义进行抽象得到的与程序验证性质相关的语义部分,对集合语义进一步抽象构造得到抽象语义,程序验证时使用抽象语义.

2 基于抽象解释的程序验证技术

2.1 基于抽象解释理论的程序变换

程序变换是程序的语法变换,它将程序变换为与其语义等价的另外一个程序,本质上是保持程序语义的程

序设计语言上的映射,语法变换导致程序语义的变换,因而可以将程序变换视为等价变换后的程序语义的反编译过程.程序语义比程序包含了更多的信息.例如,程序中包含了变量及它们的类型信息,但是不包含程序语义中具有的程序运行时对变量连续赋值的信息,程序描述了构成程序的语句的序列,但是没有程序语义所描述的程序语句的执行序列.不同于程序,程序语义中包含了程序的运行时信息.例如,程序语义能够反映程序的执行时间,但是程序运行性能在程序中却无法体现.依据 P. Cousot 的观点,可以将程序视为程序语义的抽象.

抽象解释理论为程序变换提供了一个简单而自然的实现框架:首先,基于抽象解释理论对程序的语义进行抽象(抽象解释保证语义等价性),得到程序的抽象语义;然后,对程序的抽象语义再进行抽象(程序可以视为程序语义的抽象,即对程序语义的反编译过程),构造得到的程序就是程序变换的结果.特别地,如果程序 P 的程序语义可以由计算单调的程序语义泛函 $F[[P]]$的最小不动点得到,并且$\langle\alpha,\gamma\rangle$是一个 Galois 连接,定义语义泛函 $F'[[P]]=\alpha\circ F[[P]]\circ\gamma$,则程序 P 的变换程序 $t[[P]]=lfpF'[[P]]$.基于抽象解释理论的程序变换提供了由一种语义模型构造另外一种语义模型的构造方法[12].

Bruno Blanchet 基于 Horn 逻辑的安全协议模型及其验证方法[13,14]是基于抽象解释理论的程序变换在安全协议验证领域的一个典型应用.安全协议是建立在密码体制基础上的一种通信协议,它运行在计算机网络或分布式系统中,借助于密码算法为在网络环境中实现密钥分配、身份认证、电子商务交易等任务的各方约定任务的执行步骤和执行规则.安全协议是解决互联网络安全问题最有效的手段之一,它满足的基本安全性质有保密性和认证性.Cervesato,Durgin 等人提出了基于线性逻辑的安全协议模型[15],它是安全协议基于状态迁移的一种精确模型,Blanchet 证明了基于线性逻辑的安全协议模型经过抽象解释、Skolem 化带存在量词的公式、程序变换,可以得到基于 Horn 逻辑的安全协议模型[13],给出了保密性的验证方法[13],同时,通过一组变换规则将基于进程代数的安全协议模型转换为基于 Horn 逻辑的安全协议模型,并给出了认证性的验证方法[16].具体地说,基于线性逻辑的多集重写语义,Blanchet 对基于线性逻辑的安全协议模型的踪迹(接收消息动作和发送消息动作构成的动作序列)进行抽象解释,用踪迹中出现的消息多集抽象该踪迹,并用经典逻辑中的逻辑规则替换线性逻辑中的逻辑规则,Skolem 化逻辑规则中带存在量词的公式,最后通过程序变换删除逻辑规则中与保密性验证无关的原子公式,得到了基于 Horn 逻辑的安全协议模型.

与基于线性逻辑的安全协议模型比较,基于 Horn 逻辑的安全协议模型可以采用推迟实例化、删除逻辑规则前件中相同的原子公式、被逻辑蕴涵的逻辑规则不参与消解计算等已有的计算优化方法,验证保密性和认证性时具有更好的计算效率,正因为如此,Blanchet 基于 Horn 逻辑模型设计和实现的安全协议验证工具 Proverif 在安全协议源代码级的程序验证中得到了广泛的应用[16,17].

2.2 基于抽象解释理论的程序安全性验证

2.2.1 基于抽象解释理论框架的程序安全性性质验证

基于 Galois 连接的抽象解释理论框架的程序验证工作主要是 Cousot 等人以程序静态分析工具 ASTREE 为主要成果的一系列研究[18−23],研究以 C 程序编写的运算密集型的嵌入式实时安全攸关软件系统作为研究对象,以检查软件系统中的运行时错误作为主要研究内容.

ASTREE 首先对软件系统进行编译链接,得到对应的抽象语法树,针对抽象语法树使用结构归纳法构造软件系统的不动点语义,不动点语义计算过程中使用基于阈值(threshold)的 Widening 算子,使计算过程收敛或加速其收敛.ASTREE 以存储抽象域和算术抽象域上的计算抽象程序运行过程中的计算.算术抽象域包括区间抽象域、八边形抽象域等多种抽象域,在计算精度要求较高时,八边形抽象域代替区间抽象域进行抽象计算,八边形抽象域通过迭代计算可以得到程序变量之间形如$\mp u\mp v\leq c$ 的线性不变式.

ASTREE 的抽象分析过程使用了参数化方法,用于调整抽象计算的计算精度和计算效率之间的平衡关系.例如,在选择区间抽象域、八角型抽象域方面,ASTREE 采用了对变量进行分组的方法,同一组中的变量采用同一抽象域进行抽象计算.ASTREE 基于软件系统的抽象语法树以及存储抽象域和算术抽象域的迭代计算得到抽象的不变式形式的后向不动点(post-fixpoint),以前向传播(forward propagation)的方式根据后向不动点检查软件系统是否存在运行时错误.对于虚假反例,ASTREE 基于数据依赖和控制依赖的后向程序切片方法定位引

起虚假反例的变量集合,对变量集合中与验证性质相关并且缺少精确抽象计算结果的变量集合采用计算精度较高的八角形抽象域的计算进行精化.2006 年,Mine 进一步提出了 C 程序中 Union 类型数据结构和指针算术的抽象分析方法[24],扩大了 ASTREE 分析的语法对象范围,使得它能够对许多嵌入式 C 程序检查运行时错误.

ASTREE 在应用领域专家进行参数设置后自动运行,它的抽象分析过程遍历软件的每一个抽象状态,不会遗漏每一个运行时错误,并且抽象分析过程总是终止的;ASTREE 分析几十万行的软件代码的分析过程只需要花费几个小时.例如,有 132 000 行代码的空客 A340 有线飞行系统,2003 年 11 月,ASTREE 完全自动地验证了该软件中不存在运行时错误,在 2.8GHz,300Mb 内存的 PC 机上,验证时间仅为 1 小时 20 分钟.

2.2.2 基于谓词抽象的程序安全性性质验证

谓词抽象[25−27]是抽象解释的一种特殊形式,以给定的一组谓词上的计算抽象程序指称的对象域上的计算.谓词抽象是为大规模软、硬件系统建立有穷抽象模型的一种有效方法,并且基于定理证明器和模型检验工具辅助的谓词抽象、模型检验和虚假反例制导的谓词精化的迭代验证已经成为大规模软、硬件系统自动分析与验证的一种重要方法.

谓词抽象首先由 Graf 和 Saidi 提出,对于给定的 N 个谓词 $\Phi_1,\ldots,\Phi_N$,设 $B_1,\ldots,B_N$ 为 N 个布尔变量,则谓词抽象中的 Galois 连接 $\langle\alpha,\gamma\rangle$ 定义为

$$\gamma(\exp^A(B_1,\ldots,B_N))=\exp^A[\bar{\phi}/\bar{B}];$$

$$\alpha(\phi)=\wedge\{\exp^A(B_1,\ldots,B_N)\mid\phi\Rightarrow\exp^A[\bar{\phi}/\bar{B}]\},$$

其中,$\bar{\phi}$ 表示 $\Phi_1,\ldots,\Phi_N$ 构成的谓词向量,$\bar{B}$ 表示 $B_1,\ldots,B_N$ 构成的布尔变量向量,$[\bar{\phi}/\bar{B}]$ 表示置换,它将向量 $\bar{B}$ 中的每一个分量 B_i 同时用谓词 Φ_i 替换.对于任意一个公式 ϕ,$\alpha(\phi)$ 是满足 $\phi\Rightarrow\exp^A[\bar{\phi}/\bar{B}]$ 的所有公式 $\exp^A(B_1,\ldots,B_N)$ 的合取式,$\gamma(\exp^A(B_1,\ldots,B_N))$ 将 $\exp^A(B_1,\ldots,B_N)$ 中的布尔变量 B_i 同时用谓词 Φ_i 替换,得到公式 $\exp^A[\bar{\phi}/\bar{B}]$.由于难以得到满足 $\phi\Leftrightarrow\exp^A[\bar{\phi}/\bar{B}]$ 的所有公式 $\exp^A[\bar{\phi}/\bar{B}]$,因而,谓词抽象通常将 α 用 α' 代替:

$$\alpha'(\phi)=\wedge\{B_i\mid\phi\Rightarrow\Phi_i\}.$$

Graf 和 Saidi 给出了将无穷状态系统抽象为有穷抽象状态系统的方法(对应于 Widening 算子使程序不动点语义计算过程收敛或用于加速其收敛),抽象状态之间的抽象迁移关系由系统状态之间的迁移关系在 $\Phi_1,\ldots,\Phi_N$ 上进行抽象计算得到,设抽象状态用公式 $\exp^A(B_1,\ldots,B_N)$ 表示,则抽象状态的抽象迁移关系 τ_i^A 是如下定义的函数:

$$\tau_i^A(\exp^A(B_1,\ldots,B_N))=\begin{cases}\text{false}, & \exp^A[\bar{\phi}/\bar{B}]\Rightarrow\neg g_i\\ \wedge_{j=1}^{l}\begin{cases}B_j, & post[\tau_i](\exp^A[\bar{\phi}/\bar{B}])\Rightarrow\phi_j\\ \neg B_j, & post[\tau_i](\exp^A[\bar{\phi}/\bar{B}])\Rightarrow\neg\phi_j\\ \text{true}, & \text{otherwise}\end{cases}\end{cases},$$

其中,g_i 是系统当前状态迁移的卫式条件.$\exp^A[\bar{\phi}/\bar{B}]\Rightarrow\neg g_i$ 等逻辑蕴涵公式的有效性借助于已有的定理证明器自动判定.得到有穷抽象状态系统后,借助于已有的模型检验工具对抽象系统的性质进行验证.基于谓词抽象的程序验证的典型研究工作有:Ball 等人以程序自动验证工具包 SLAM 为突出成果的一系列研究工作[28,29],Henzinger 等人以程序自动验证工具 BLAST 为突出成果的一系列研究工作[30,31],Clarke 等人以程序自动验证工具 Magic 为突出成果的一系列研究工作[32,33].

SLAM 由 C2BP,BEBOP,NEWTON 等 3 种工具组成,C2BP 是一个自动的谓词抽象工具,在给定一组谓词的条件下,它将一个 C 程序抽象为一个布尔程序,BEBOP 是布尔程序的一个模型检验工具,NEWTON 通过分析 C 程序中的可行路径发现用于精化布尔程序的其他谓词,SLAM 被应用于验证微软公司驱动开发包中的驱动程序的安全性.

Henzinger 等人提出了“惰性抽象”方法,包括 On-the-Fly 的抽象方法和 On-Demand 的精化方法,对谓词抽象、模型检验和虚假反例制导的抽象精化的迭代验证过程中的计算进行优化,以提高验证效率,并将一阶逻辑子句的 Craig 插值方法应用于虚假反例制导的抽象精化,提出了“基于证明的抽象”方法,BLAST 对具有 13 万行

源代码的 C 程序成功地进行了验证.

Clarke 等人基于“弱模拟(weak simulation)”提出了对程序与其规范的一致性进行验证的方法,程序及其规范都用标号迁移系统刻画,一致性验证过程归结为程序标号迁移系统与程序规范标号迁移系统是否弱模拟的判定过程.Magic 支持程序模块的组合验证,有效地缓解了验证过程中状态空间爆炸的问题.

基于谓词抽象、模型检验和反例制导的抽象精化的迭代验证过程也被应用于并发程序的安全性验证研究,典型的研究工作有:Chaki 等人在 Magic 的基础上对并发 C 程序的验证研究[34],Henzinger 等人在 BLAST 的基础上对多线程 C 程序中资源竞争的检测研究[35],Qadeer 等人在 SLAM 的基础上以 KISS 为主要成果的研究[36].

Chaki 等人以基于消息传递的并发 C 程序作为主要研究对象,在反例制导的抽象精化的迭代验证框架下,基于谓词抽象、动作制导抽象分别对并发 C 程序中的数据和事件进行抽象,逐步增加程序抽象的精度,直到证明程序满足安全性规范,或者构造出了不满足安全性规范的反例.

Henzinger 等人提出了对并发程序进行模型检验的 TAR 算法,TAR 算法采用基于线程模块的假设——保证推理克服多线程程序控制状态指数复杂性问题,每一次遍历一个线程的状态空间,并对环境作用在该线程上的效果作出假设,计算多线程 C 程序的状态迁移时使用上界逼近,而环境假设使用下界逼近.

Qadeer 等人采用了顺序程序模拟并发程序运行的思想,并发程序中每一个线程都有自己的运行栈,顺序程序是只有一个线程的并发程序.Qadeer 等人利用单一运行栈的行为模拟多个运行栈的形为,将并发程序顺序化,基于 SLAM 开发了工具 KISS.

2.3 基于抽象解释理论的程序活性性质验证

2.3.1 基于迁移谓词抽象的程序活性性质验证

迁移谓词抽象[37]也是程序抽象解释的特殊形式,迁移谓词抽象以给定的一组迁移谓词上的计算抽象程序迁移关系域上的计算,基于迁移谓词抽象的抽象-精化迭代验证也被应用于对大规模软件活性性质的自动验证中.典型的研究工作是 Podelski,Rybalchenko 和 Cook 等人对大规模软件活性性质验证的一系列研究[37−43].他们首先提出了迁移不变式的概念,迁移不变式是程序迁移关系的超集,基于良基(well-founded)迁移关系和程序停机性之间的关联,对于程序停机性性质验证提出了基于析取良基迁移不变式的刻画定理,基于 Vardi 的自动机理论框架[43],对于程序活性性质验证给出了基于析取良基迁移不变式的刻画定理.在此基础上,提出了基于迁移谓词抽象的抽象-精化迭代验证框架,对大规模软件的活性性质进行验证,并应用提出的验证方法对 Windows 操作系统的驱动程序进行了验证,发现了之前未曾发现的不满足活性性质的软件 bug.

2.3.2 规约为参数化数学最优化问题的活性性质验证

在 Floyd/Naur/Hoare 经典程序正确性验证方法的基础上,Cousot 针对半代数程序(semialgebraic program)提出了程序不变式和用于停机性验证的秩函数(rank function)的自动构造方法[44].基于半代数程序多项式形式的计算语义,根据程序不变式和秩函数设定的参数化抽象的表达式形式和程序的迁移关系以及验证条件,给出程序不变式和秩函数参数化表达式中参数之间的数值约束关系,参数之间的数值约束关系可以进一步使用拉格朗日松弛法进行松弛变换,消除参数之间数值约束关系中的逻辑蕴涵;然后基于求解半正定方程约束最优化问题的求解算法,给出参数化表达式中参数的一组可行解,半正定方程约束求解能够消除参数之间数值约束关系中的全称量词约束.基于程序不变式和秩函数的自动构造方法,Cousot 给出了自动验证安全性以及活性性质的新思路.

综上所述,Cousot 等人以 ASTREE 为主要成果的一系列研究建立在区间抽象域、八边形抽象域等具有不同抽象层次的抽象域的基础上,用更精细的抽象域进行抽象计算是 ASTREE 采用的抽象精化方法,谓词抽象以一组谓词作为抽象域,在原有谓词基础上增加新的谓词进行抽象计算,是谓词抽象采用的抽象精化方法,两者都应用于对程序安全性性质进行验证,抽象分析方法本质上都是对程序状态进行抽象;在对程序活性性质进行验证的研究中,抽象分析方法本质上都是对程序迁移关系进行抽象,Cousot 的研究更注重于应用参数化数学最优化问题的求解算法及其计算工具软件包.

3 结 语

抽象解释理论从对象域抽象、Widening 算子、迁移关系抽象 3 个方面建立对程序不动点语义上界抽象逼近的理论框架.本文讨论了基于抽象解释理论的程序变换、基于抽象解释理论的程序安全性和活性性质验证 3 种典型的应用.抽象解释理论在程序分析与验证研究中得到了越来越多的应用,针对大规模软件和硬件系统的基于抽象解释理论的自动分析与验证工具也在不断出现并得到了实际应用.基于抽象解释理论的程序验证技术在取得巨大进展的同时,该领域内尚有一些问题值得进一步研究,主要集中在以下几个方面:

(1) 关系型抽象域的研究

计算不变式是程序验证的一种重要方法,区间抽象域的计算能够得到程序变量取值范围的不变式,八边形抽象域以及凸多面体抽象域的迭代计算能够得到程序变量之间的线性关系不变式.能够计算变量之间非线性关系不变式的关系型抽象域需要进一步深入的研究,在满足计算精度要求的同时,还要满足规模化的要求,能够对大规模软件和硬件系统计算不变式.

(2) 虚假反例制导的抽象精化研究

基于抽象解释理论对大规模软、硬件系统进行验证,根据验证中出现的虚假反例快速、准确地定位引起虚假反例的抽象计算,使用更精细的抽象域上的计算替换引起虚假反例的抽象计算,以“惰性计算”的方式重新计算程序不动点语义和验证性质,是获得有效验证结果,提高大规模软、硬件系统验证效率需要研究的重要内容.

(3) 程序设计语言中各种语法对象抽象计算的研究

程序设计语言中动态内存分配、函数调用、指针、嵌套循环、复杂数据结构等语法对象的抽象计算仍然是程序验证研究的重要内容,特别地,动态内存分配基于堆(heap)语义的抽象计算[45]以及指针的 point-to 分析、shape 分析、alias(别名)分析[46]仍然是其中的研究热点.

(4) 基于抽象解释理论面向应用领域的程序验证技术研究

不同应用领域程序指称的对象域以及对象域上的计算模式与要满足的性质也不一样.例如,在基于计算语义的安全协议代码级源程序验证中,以概率多项式时间图灵机作为计算模型,以保密性和认证性作为验证性质.基于抽象解释理论结合应用领域特征提出有效的程序验证技术值得进一步深入研究.

References:

[1] Cousot P, Cousot R. Abstract interpretation: A unified Lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. of the 4th POPL. Los Angeles: ACM Press, 1977. 238−252. http://www.di.ens.fr/~cousot/COUSOTpapers/POPL77.shtml

[2] Cousot P, Cousot R. Systematic design of program analysis frameworks. In: Proc. of the 6th POPL. San Antonio: ACM Press, 1979. 69−282. http://www.di.ens.fr/~cousot/COUSOTpapers/POPL79.shtml

[3] Cousot P, Cousot R. Abstract interpretation frameworks. Journal of Logic and Computer, 1992,2(4):511−547.

[4] Cousot P, Cousot R. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In: Bruynooghe M, Wirsing M, eds. Proc. of the PLILP'92. LNCS 631, Springer-Verlag, 1992. 269−295.

[5] Cousot P. Abstract interpretation based formal methods and future chanllenges. In: Wilhelm R, ed. Informatics——10 Years Back, 10 Years Ahead. Berlin, Heidelberg: Springer-Verlag, 2000. 138−156.

[6] Cousot P, Cousot R. Abstract interpretation based program testing. In: Proc. of the SSGRR 2000 Computer & eBusiness Int'l Conf. 2000. Compact disk paper 248. http://www.di.ens.fr/~cousot/COUSOTpapers/SSGRRP-00-PC-RC.shtml

[7] Cousot P, Cousot R. Basic concepts of abstract interpretation. In: René J, ed. Building the Information Society. Toulouse: Kluwer Academic Publishers, 2004. 359−366.

[8] Cousot P, Cousot R. Refining model checking by abstract interpretation. Automated Software Engineering Journal (Special Issue on Automated Software Analysis), 1999,6(1):69−95.

[9] Cousot P, Cousot R. Static determination of dynamic properties of programs. In: Robinet B, ed. Proc. of the 2nd Int'l Symp. on Programming. Paris, 1976. 106−130. http://www.di.ens.fr/~cousot/COUSOTpapers/ISOP76.shtml

[10] Cousot P, Halbwachs N. Automatic discovery of linear restraints among variables of a program. In: Proc. of the 5th POPL. Arizona: ACM Press, 1978. 84−97.

[11] Miné A. The octagon abstract domain. Higher-Order and Symbolic Computation, 2006,19(1):31−100.

[12] Cousot P, Cousot R. Systematic design of program transformation frameworks. In: Launchbury J, Mitchell JC, eds. Proc. of the Symp. on Principles of Programming Languages. Portland: ACM Press, 2002. 178−190.

[13] Cervesato I, Durgin N, Lincoln P, Mitchell J, Scedrov A. A metanotation for protocol analysis. In: Pandya P, Radhakrishnan J, eds. Proc. of the 12th IEEE Computer Security Foundation Workshop (CSFW-12). Mordano: IEEE, 1999. 55−69.

[14] Blanchet B. An efficient cryptographic protocol verifier based on prolog rules. In: Pandya P, Radhakrishnan J, eds. Proc. of the 14th IEEE Computer Security Foundations Workshop. Cape Breton: IEEE Computer Society Press, 2001. 82−96.

[15] Blanchet B. From secrecy to authenticity in security protocols. In: Cousot P, ed. Proc. of the 9th Int'l Static Analysis Symp. (SAS 2002). LNCS 2477, Madrid: Springer-Verlag, 2002. 342−359.

[16] Goubaut-Larrecq J, Parrennes F. Cryptographic protocol analysis on real C code. In: Cousot R, ed. Proc. of the 6th Int'l Conf. on Verification, Model Checking and Abstract Interpretatio. LNCS 3385, Paris: Springer-Verlag, 2005. 363−379.

[17] Bhargavan K, Fournet C, Gordon AD, Tse S. Verified interoperable implementations of security protocols. In: Proc. of the 19th IEEE Computer Security Foundations Workshop. Venice: IEEE Computer Society, 2006. 139−152. http://research.microsoft.com /~adg/Publications/tfng-csfw-2006.pdf

[18] Blanchet B, Cousot P, Cousot R, Feret J, Mauborgne L, Miné A, Monniaux D, Rival X. A static analyzer for large safety-critical software. In: Proc. of the ACMSIGPLAN 2003 Conf. PLDI. San Diego: ACM Press, 2003. 196−207.

[19] Blanchet B, Cousot P, Cousot R, Feret J, Mauborgne L, Mine A, Monniaux D, Rival X. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software. In: Mogensen T, Schmidt D, Sudborough, eds. Proc. of the Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Jones ND. LNCS 2566, Springer-Verlag, 2002. 85−108.

[20] Cousot P, Cousot R, Feret J, Mauborgne L, Miné A, Monniaux D, Rival X. The ASTRÉE analyser. In: Sagiv M, ed. Proc. of the European Symp. on Programming. LNCS 3444, Edinburgh: Springer-Verlag, 2005. 21−30.

[21] Mauborgne L. ASTRÉE: Verification of absence of run-time error. In: Jacquard R, ed. Building the Information Society. Toulouse: Kluwer Academic Publishers, 2004. 385−392.

[22] Mauborgne L, Rival X. Trace partitioning in abstract interpretation based static analyzer.In: Sagiv M, ed. Proc. of the European Symp. on Programming. LNCS 3444, Edinburgh: Springer-Verlag, 2005. 5−20.

[23] Rival X. Understanding the origin of alarms in ASTRÉE. In: Hankin C, Siveroni I, eds. Proc. of the 12th Int'l Static Analysis Symp. LNCS 3672, London: Springer-Verlag, 2005. 303−319.

[24] Miné A. Field-Sensitive value analysis of embedded c programs with union types and pointer arithmetics. In: Irwin MJ, de Bosschere K, eds. Proc. of the Languages, Compilers, and Tools for Embedded Systems 2006 (LCTES). Ottawa: ACM Press, 2006. 54−63.

[25] Graf S, Saidi H. Construction of abstract state graphs with PVS. In: Grumberg O, ed. Proc. of the CAV'97. LNCS 1254, Haifa: ACM Press, 1997. 72−83.

[26] Colon M, Uribe TE. Generating finite state abstractions of reactive systems using decision procedures. In: Hu AJ, Vardi MY, eds. Proc. of the CAV'98. LNCS 1427, Vancouver: ACM Press, 1998. 293−304.

[27] Das S, Dill D, Park S. Experience with predicate abstraction. In: Halbwachs N, Peled D, eds. Proc. of the 11th Int'l Conf. CAV'99. LNCS 1633, Trento: Springer-Verlag, 1999. 160−171.

[28] Ball T, Rajamani SK. The SLAM project: Debugging system software via static analysis. In: Launchbury J, Mitchell JC, eds. Proc. of the Symp. on Principles of Programming Languages. Portland: ACM Press, 2002. 1−3.

[29] Ball T, Naik M, Rajamani S. From symptom to cause: Localizing errors in counterexample traces. In: Bert D, Bowen JP, King S, Walden M. Proc. of the Symp. on Principles of Programming Languages. New Orleans: ACM Press, 2003. 97−105.

[30] Henzinger TA, Jhala R, Majumdar R, Sutre G. Lazy abstraction. In: Launchbury J, Mitchell JC, eds. Proc. of the Symp. on Principles of Programming Languages. Portland: ACM Press, 2002. 58−70.

[31] Henzinger TA, Jhala R, Majumdar R, McMillan KL. Abstractions from proofs. In: Jones ND, Leroy X, eds. Proc. of the 31st ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. Venice: ACM Press, 2004. 232−244.

[32] Chaki S, Clarke EM, Groce A, Jha S, Veith H. Modular verification of software components in C. IEEE Trans. on Software Engineering, 2004,30(6):388−402.

[33] Clarke EM, Grumberg O, Jha S, Lu Y, Veith H. Counterexample-Guided abstraction refinement for symbolic model checking. Journal of the ACM, 2003,50(5):752−794.

[34] Chaki S, Clarke E, Groce A, Ouaknine J, Yorav K. Efficient verification of sequential and concurrent C programs. Formal Methods in System Design, 2004,25(2-3):129−166.

[35] Henzinger TA, Jhala R, Majumdar R, Qadeer S. Thread-Modular abstraction refinement. In: Jr Hunt WA, Somenzi F, eds. Proc. of the CAV 2003. Boulder: ACM Press, 2003. 262−274.

[36] Qadeer S, Wu D. KISS: Keep it simple and seqeuential. In: Pugh W, Chambers C, eds. Proc. of the ACM SIGPLAN 2004 Conf. on Programming Language Design and Implementation. Washington: ACM Press, 2004. 14−24.

[37] Podelski A, Rybalchenko A. Transition predicate abstraction and fair termination. In: Palsberg J, Abadi M, eds. Proc. of the Symp. on Principles of Programming Languages. ACM Press, 2005. 132−144.

[38] Podelski A, Rybalchenko A. Transition invariants. In: Ganzinger H, ed. Proc. of the 9th Annual IEEE Symp. on. Logic in Computer Science. Turku: IEEE Computer Society Press, 2004. 32−41.

[39] Cook B, Podelski A, Rybalchenko A. Abstraction refinement for termination. In: Hankin C, Siveroni I, eds. Proc. of the SAS 2005. LNCS 3672, London: Springer-Verlag, 2005. 87−101.

[40] Cook B, Podelski A, Rybalchenko A. Terminator: Beyond safety. In: Ball T, Jones RB, eds. Proc. of the CAV 2006. LNCS 4144, Seattle: Springer-Verlag, 2006. 415−418.

[41] Cook B, Podelski A, Rybalchenko A. Termination proofs for systems code. In: Schwartzbach MI, Ball T, eds. Proc. of the ACM SIGPLAN 2006 Conf. on Programming Language Design and Implementation. Ottawa: ACM Press, 2006. 415−426.

[42] Cook B, Gotsman A, Podelski A, Rybalchenko A, Vardi MY. Proving that programs eventually do something good. In: Hofmann M, Felleisen M, eds. Proc. of the Symp. on Principles of Programming Languages. Nice: ACM Press, 2007. 265−276.

[43] Vardi MY. Verification of concurrent programs: The automata-theoretic framework. Annals of Pure and Applied Logic, 1991,51: 79−98.

[44] Cousot P. Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. In: Cousot R, ed. Proc. of the VMCAI 2005. LNCS 3385, Paris: Springer-Verlag, 2005. 1−24.

[45] Rinetzky N, Bauer J, Reps T, Sagiv M, Wilhelm R. A semantics for procedure local heaps and its abstractions. In: Palsberg J, Abadi M, eds. Proc. of the Symp. on Principles of Programming Languages. ACM Press, 2005. 296−309.

[46] Corbera F, Asenjo R, Zapata EL. A framework to capture dynamic data structures in pointer-based codes. IEEE Trans. on Parallel and Distributed Systems, 2004,15(2):151−166.

李梦君(1975－),男,湖北云梦人,博士,讲师,主要研究领域为形式化方法与技术,信息安全技术.

李舟军(1963－),男,博士,教授,CCF 高级会员,主要研究领域为安全协议的形式化分析,进程代数理论,数据挖掘.

陈火旺(1936－),男,教授,博士生导师,中国工程院院士,CCF 高级会员,主要研究领域为软件工程,软件理论.