支持抽象解释的静态分析方法的形式化体系研究

张 弛 黄志球 丁泽文
(南京航空航天大学计算机科学与技术学院 南京 211106)

摘 要 在安全关键领域中,如何保证软件的安全性已经成为了一个广受关注的重要课题。静态程序分析是一类十分有效的程序自动化验证方法。基于抽象解释的静态分析技术在验证软件的非功能性安全属性上表现十分突出。可配置程序分析(Configurable Program Analysis,CPA)是一种通用静态分析方法形式化体系,旨在用一种形式化体系对静态分析的分析阶段进行建模。使用CPA对基于抽象解释的静态分析进行建模,给出如何使用CPA形式化体系描述基于抽象解释的静态分析,给出了从待分析程序到CPA形式化体系的转换规则;提供了一种在安全关键性领域中的软件正确性自动验证方法,为基于抽象解释的静态分析工具的实现提供了一种可行方案。

关键词 形式化方法,静态分析,抽象解释,可配置程序分析
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.12.025

Research on Static Analysis Formalism Supporting Abstract Interpretation

ZHANG Chi HUANG Zhi-qiu DING Ze-wen
(School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China)

Abstract In safety-critical domain, software safety assurance becomes a widely concerned issue. Static program analysis is an effective method for automating program validation. Static analysis is an efficient formal method for the use of verification of non-functional safety-critical properties. CPA(Configurable Program Analysis) is a formalism for static analysis. It intends to describe the analysis phrase in static analysis by a general formal framework. This paper aimed at formally modeling the analysis phrase in abstract interpretation with the formalism CPA. We illuminated the rules of transformation from source code to the formalism of configurable program analysis. This paper provided a way to automatically verify the software correctness in safety-critical domain and provided a feasible solution for static analysis tool based on abstract interpretation.

Keywords Formal methods, Static analysis, Abstract interpretation, Configurable program analysis

1 引言

静态分析方法是一种保障软件正确性和可靠性的重要手段。静态分析不需要真实执行程序,只需对程序的语法语义进行静态分析[1]。与动态测试相比,首先,静态分析能够模拟程序运行时的所有状态,能对程序中所有可能的路径进行分析;其次,静态分析的本质是分析程序所转换成的状态模型内的状态转换关系。但是,分析整个程序的全部状态空间是不现实的。因此,静态分析方法往往需要对程序的状态空间进行抽象。静态分析往往要通过损失精度来达到提高分析效率和保证分析结果可靠性的目的。在保证分析结果正确性的前提下,静态分析权衡了分析过程的精度和效率。

目前,静态分析的主要形式化方法有模型检测、定理证明和抽象解释。模型检测的基本思想是通过穷举系统模型的状态空间来验证该模型是否满足时序逻辑描述的系统属性[2]。模型检测的工具主要有BLAST,Spin,UPPAAL等。定理证明利用逻辑系统对软件进行描述,通过公理或者推理规则对系统具有的安全性质进行规约验证[3],其主要工具有PVS原型验证系统、定理证明器HOL等。抽象解释理论对程序语义进行抽象,将程序的具体语义转化到抽象域中,在抽象域中的抽象语义下对程序性质进行分析[4]。主要工具有ASTRÉE[5],Polyspace,Fluctuat等。

可配置程序分析是由Beyer等人提出的用于支持模型检测和抽象解释的静态分析形式化体系,旨在用一种形式化体系同时描述模型检测和抽象解释[6]。其可以被用来对抽象解释的分析阶段进行建模。

目前,国内抽象解释的相关工作主要集中在非凸抽象域的研究[7]和抽象解释在安全关键性(Safety Criticla Software)软件中的应用[8]。本文工作主要关注于提出一种可以扩展和参数化的抽象解释分析框架,以适应各种分析场景,为抽象解

到稿日期:2016-09-17 返修日期:2017-01-09 本文受国家高技术研究发展计划(863)(2015AA105303),国家自然科学基金资助项目(61272083),软件新技术与产业化协同创新中心资助。

张 弛(1991—),男,硕士生,主要研究方向为模型检测、抽象解释,E-mail:ZC_ruler@nuaa.edu.cn;黄志球(1965—),男,教授,博士生导师,CCF高级会员,主要研究方向为软件工程、软件度量;丁泽文(1991—),男,硕士生,主要研究方向为模型检测、抽象解释。

释工具的设计与实现提供一个可行方案。同时，用形式化体系描述抽象解释便于对其进行扩展，如多抽象域的结合或者通过参数调节分析过程的精度和效率。

本文第 2 节介绍基于抽象解释的静态分析的一般过程，给出一个通用的分析框架；第 3 节具体介绍可配置程序分析的形式化体系表示方法；第 4 节阐述如何将程序控制流图转化到 CPA 形式化体系中；第 5 节通过实例来说明如何使用 CPA 对程序进行分析验证；最后总结全文。

2　基于抽象解释的静态分析框架

抽象解释理论是 Cousot 和 Cousot[9] 于 1977 年提出的在保证分析程序语义可靠性的基础上对程序语言进行抽象的理论。抽象解释理论的基本思想是用抽象语义代替具体语义来描述源程序语言，以确定程序抽象语义和具体语义之间的转化关系，之后求解程序抽象语义，利用得到的抽象语义来实现具体语义的计算，程序抽象执行的结果能反映程序真实执行的部分信息。其以损失精度为代价来确保计算的可行性和高效性。该理论本质上是一种在计算精度和计算效率上获取平衡的静态分析方法。

目前，基于抽象解释的静态分析方法主要用于分析和验证系统的非功能性属性[10]，其中主要包括运行时错误的验证（程序中是否含有除零错、是否发生数组越界、是否发生算术溢出等）、最坏情况下执行时间（Worst-Case Execution Time，WCET）的计算等。

图 1 给出了一个基于抽象解释的静态分析框架，该过程主要分为 3 个阶段：预处理阶段、分析阶段和验证阶段。

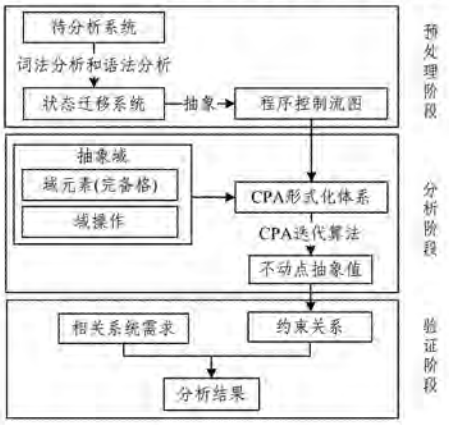

图 1　基于抽象解释的静态分析框架

在预处理阶段，由于抽象解释的静态分析工具是一个基于程序的分析工具，其直接分析对象是源程序，因此需要在预处理阶段将源代码转化成与之等价的抽象形式，即通过词法分析和语法分析器得到其抽象语法树，然后转化成便于分析的状态迁移系统。一般地，我们选择程序控制流图（Control-Flow Graph，CFG）来表示源程序的状态迁移关系。一些成熟的抽象解释工具，如 ASTRéE，在这个阶段还会进行一些参数化的处理，以便在实际验证分析时通过配置不同的参数来调节分析器的效率和精度[11]。

分析阶段是抽象解释理论表现的主要阶段。在该阶段，构建特定的抽象域，将程序的具体语义转化到抽象语义上，然后结合抽象域和控制流图的信息，运用抽象解释中的迭代策略计算程序控制流图中所有结点的不动点抽象值，从而完成分析。抽象解释理论中的所有计算都是在抽象域中展开的，抽象域是抽象解释中的核心元素。在程序数值性质的验证方面，目前使用的经典的抽象域主要有区间抽象域[12]、八边形抽象域[13]、多面体抽象域[13]、椭圆抽象域[11]等。抽象域的构成主要包括以下两个方面。

域元素：具体程序在抽象域中的表示方法。具体域和抽象域之间的关系可以通过一个 Galois 连接[14]来表示：如变量在某一结点处的取值表现在区间抽象域中即为变量在该结点处的取值区间。实际验证时需根据不同的需求选择域元素的表示方法。

域操作：抽象值在抽象域中的操作方法。一般将域操作分为以下 4 类：交（meet）和接合（join）、区间算术运算、迁移函数、加宽（widening）和变窄（narrowing）算子[15]。

在验证阶段，将得到的程序结点上的不动点抽象值转化为程序具体的变量约束关系，并根据系统的需求文档和设计说明文档对程序的变量数值性质进行分析，判断变量的数值性质是否满足规约，得到分析结果，从而完成整个抽象解释的静态分析过程。

工业界常用的仿真、模拟和测试等方法可以找到程序中的错误，却无法保证程序中没有错误。而基于抽象解释的静态分析过程需要对程序语义进行抽象，其分析过程会覆盖所有的执行路径，可以对程序的正确性进行验证，但是程序执行路径的过近似可能导致产生虚假反例。也就是说，抽象解释方法在验证结果为正确时，可以保证程序中没有错误，但在验证结果为错误时，不能保证程序中一定有错误。自动化分析验证方法中误报的产生会带来额外的工作量。在分析过程中，常需要根据实际情况选择不同精度和效率的分析方法，分析精度越高，其分析效率就越低。因此需要一种参数化的分析框架，根据实际情况配置不同的参数，以满足分析过程对不同精度的需求。

本文使用可配置程序分析这种静态分析框架，对抽象解释的分析阶段进行建模。其以程序控制流图为分析对象，配置分析过程中的抽象域、抽象状态之间的合并方式以及分析的终止判断方法，根据配置好的分析方法计算各个程序结点的不动点抽象值，完成分析过程。

3　可配置程序分析 CPA

可配置程序分析是一个形式化地进行软件验证和程序分析的通用框架，旨在用一种通用框架来描述不同的静态分析方法。不同的静态分析方法主要在抽象层次和迭代算法两个方面有所区别。例如在抽象解释方法中，对变量的取值范围和相互关系进行的区间抽象、八边形抽象等，在模型检测方法中，用于减少抽象状态数目的谓词抽象（predicate abstraction）[16]、惰性抽象（lazy abstraction）[17]等；抽象解释基于不动点理论迭代抽象状态直至找到不动点，模型检测的算法是对状态空间进行遍历，直至所有状态空间遍历完毕或者找到真实错误路径。基于这种思想，可配置（configurable）意为通过配置静态分析的抽象环境和迭代算法调整分析过程的精度

和效率，来满足不同的分析精度和分析效率需求。为配置不同的迭代算法，需要选择不同的合并(merge)操作和终止判断(stop)操作。合并操作用以合并抽象状态，终止判断操作用以判断迭代算法是否终结。下面介绍 CPA 的形式化体系描述。

可配置程序分析 CPA 被形式化定义为 $\mathbb{D}=(D,\rightsquigarrow,merge,stop)$，其中包括抽象域 D、抽象状态迁移关系 $\rightsquigarrow$、合并操作 *merge*、终止判断操作 *stop*。下面具体说明这 4 部分。

(1)抽象域 $D=(C,\varepsilon,[\cdot])$，其中 C 为程序具体状态的集合，其中一个具体状态 c 为程序运行到某一状态时变量在不同程序结点处的实际值，即 $X\cup\{Pc\}$，其中 X 为变量集合，$\{Pc\}$ 为程序结点集合。ε 是用于描述抽象环境的完备格 $\varepsilon=(E,\sqsubseteq,\sqcap,\sqcup,\bot,\top)$，其中 E 是抽象状态集合，$\sqsubseteq\subseteq E\times E$ 是抽象状态间的偏序关系，$\sqcap$ 表示最大下界，$\sqcup$ 表示最小上界，$\bot\in E$ 表示集合中的最小元素，$\top\in E$ 表示集合中的最大元素。具体化方法 $[\cdot]:E\to 2^C$ 赋予抽象状态对应的具体状态集，即一个抽象状态可能对应多个具体状态。为保证分析的可靠性，要求 $[\top]=C$，$[\bot]=\emptyset$；$\forall e,e'\in E$，$[e\sqcup e']\supseteq[e]\cup[e']$。

(2)抽象状态迁移 $\rightsquigarrow\subseteq E\times G\times E$，其中 E 是抽象状态集合，G 是控制流图的边的集合，$\rightsquigarrow$ 表示一个抽象状态 e 通过一个 CFG 的边 g 到达一个新的抽象状态 e'，记为 $e\overset{g}{\rightsquigarrow}e'$。为保证分析过程的可靠性，要求 $\forall e\in E$：$\exists e'\in E$：$e\overset{g}{\rightsquigarrow}e'$，$\forall e\in E$，$g\in G$：$\bigcup_{e\overset{g}{\rightsquigarrow}e'}[e']\supseteq\bigcup_{c\in[e]}\{c'\mid c\xrightarrow{g}c'\}$，其中 $c\xrightarrow{g}c'$ 表示程序结点 c 通过控制边 g 上的操作到达程序结点 c'。有关控制流图的具体形式化定义将在第 4 节给出。

(3)合并操作符 $merge:E\times E\to E$ 将两个抽象状态的信息进行合并。为保证分析的可靠性，要求 $e'\sqsubseteq merge(e,e')$。

(4)终止判断操作 $stop:E\times 2^E\to\mathbb{B}$ 表示某一个抽象状态是否被一个抽象状态集合所覆盖。为了保证分析的可靠性，要求 $stop(e,R)=\text{true}\to[e]\subseteq\bigcup_{e'\in R}[e']$。

根据 CPA 的形式化定义，给出如图 2 所示的基于 CPA 的抽象解释静态分析过程。在分析阶段，以预处理阶段生成的控制流图为分析对象，结合抽象域域元素的定义，得到抽象状态的迁移关系；根据抽象域的域操作定义，结合具体的分析精度要求，配置 *merge* 操作和 *stop* 操作，从而完成不动点的计算；最终当 *stop* 操作为真时，迭代结束，得到程序变量的约束关系。具体的迭代算法见文献[6]中的相关描述，在此不再赘述。

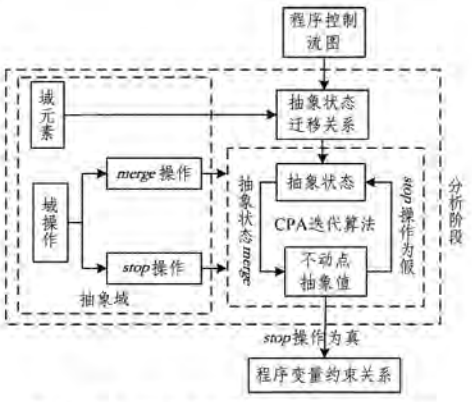

图 2 基于 CPA 的抽象解释方法的分析过程

4 程序控制流图到 CPA 的转化

CPA 的形式化体系 $\mathbb{D}=(D,\rightsquigarrow,merge,stop)$。其中 D 为抽象域的描述，在数值分析的抽象解释中，可以选择的抽象域有区间抽象、八边形抽象、多面体抽象等。*merge* 是合并操作，基于用户的配置来完成不同功能的分析过程。*stop* 是终止判断操作，基于用户配置来选择何时完成 CPA 执行算法。$\rightsquigarrow$ 表示抽象状态的迁移关系，源程序在转化成程序控制流图 CFG 后，需要转化成抽象状态的迁移关系 $\rightsquigarrow$ 才能完成 CPA 形式化体系，从而运行执行算法以完成静态分析过程。

定义程序控制流图 $C=(Pc,pc_0,E)$。$pc_0\in Pc$ 是初始结点，执行操作 $E\subseteq Pc\times Ops\times Pc$，称 (pc,op,pc') 或者 $pc\xrightarrow{op}pc'$ 为程序结点 pc 通过 op 操作到达结点 pc'。根据文献[18]中对数据流分析(data-flow analysis)的相关介绍，在程序分析过程中关注的操作主要有两类：赋值(assignment)操作和判断(assume)操作。赋值操作的基本形式是 $x:=e$，其中 x 是某一变量，e 是一个算术表达式；判断操作的基本形式是 *assume*(p)，p 是一个布尔表达式。这两类语句的结合使用可以描述大多数常见的程序结构。例如，表 1 列出了 C 语言中的一些语句结构，赋值操作的结合使用可以描述顺序执行的简单语句；图 3 分别给出了 if-else 语句和 while 语句的控制流图表示，二者可以通过赋值操作和判断操作的结合使用来表示。

表 1 C 语言语句结构的类型及特点

语句类型	语句结构	语句特点
简单语句	赋值语句、变量声明语句等	顺序执行，不影响控制流图走向
复合语句	if 语句、while 语句等	逻辑结构复杂，改变控制流图走向
转移语句	break 语句、continue 语句等	与复合语句结合，改变控制流图走向

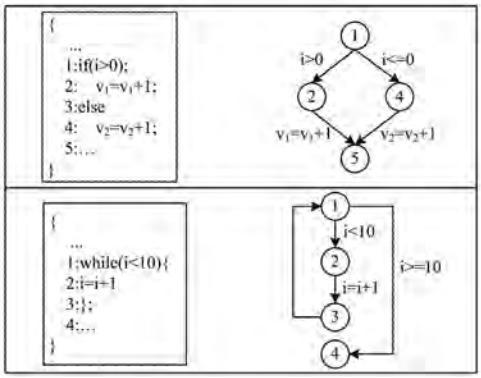

图 3 if-else 语句和 while 语句的控制流图表示

程序具体状态(concrete state)的集合为 $C\{pc,X\}$，其中 $pc\in Pc$ 是程序的某一个结点，X 是在该结点处所有变量的值的集合，一个程序具体状态 $c\in C$ 是指在某一时刻，程序所在结点处每个变量的值的状态。

经过赋值语句 $(pc,x:=e,pc')$ 后，具体状态 $c(pc,X)$ 到达新的具体状态 $c'(pc',X\cup\{x'\}/\{x\})$，其中变量 x 的值改为 e。经过判断语句 $(pc,assume(p),pc')$ 后，如果具体状态 $c(pc,X)$ 中 pc 处变量 $x\cap p=\text{true}$，则到达新的状态 $c'(pc',X)$，新状态 c' 的变量值状态与 c 相同；如果 $x\cap p=\text{false}$，则无法到达新的状态。若记具体状态为 $c(pc,s)$，其中 s 为在结点 pc 处程序的各个变量值信息，则有如下表示：

$$s'=\begin{cases}s, & \text{if } op\equiv assume(p) \text{ and } s\vdash p\\ s[x\vdash\!\!\rightarrow e], & \text{if } op\equiv x:=e\end{cases}$$

抽象状态的迁移关系$\rightsquigarrow$是指一个抽象状态在执行了控制流图上一边的运算后达到另一个抽象状态的过程。抽象状态的集合为 $E(pc,\alpha(X))$，其中 $pc\in Pc$ 是程序某一个结点，α 是具体域到抽象域的抽象函数，$\alpha(X)$是变量之间的关系的抽象表示。经过赋值操作$(pc,x:=e,pc')$后，抽象状态 $e(pc,\alpha(X))$到达新的抽象状态 $e'(pc',\alpha(X)\cup\alpha(\{x'\})/\alpha(\{x\}))$。经过判断操作$(pc,assume(p),pc')$后，若 $\alpha(X)\cup p\neq\varnothing$，则状态 e 可以到达新的状态 $e'(pc',\alpha(X)\cap p)$；若 $\alpha(X)\cup p=\varnothing$，则无法到达新的状态。记抽象状态 $e(pc,s)$，其中 s 为在结点 pc 处程序变量值之间关系的抽象表示，即可有如下表示：

$$s'=\begin{cases}s\cap p, & \text{if } op\equiv assume(p) \text{ and } s\cap p\neq\varnothing\\ s[\alpha(x\vdash\!\!\rightarrow e)], & \text{if } op\equiv x:=e\end{cases}$$

这样，在遍历程序控制流图时，从初始状态 s_0 出发，若 s_0 和 s'之间存在一条赋值语句或判断语句，根据上述定义，可以得到新的抽象状态 s'，再根据与 s'相连的程序结点，得到新的抽象结点，遍历控制流图直至没有新的抽象结点产生，从而得到抽象状态的迁移关系$\rightsquigarrow$。

对于 CPA 形式化体系中的 *merge* 操作的配置，主要根据抽象域域操作的相关定义得到。一般地，在抽象解释中，我们定义 *merge* 操作为两个抽象域的最小上界，即 $merge^{join}(e,e')=e\sqcup e'$，其中$\sqcup$为抽象域的最小上界。为了加速迭代过程，可以定义新的 $merge^{+}$ 使得 $merge^{join}=e\sqcup e'\sqsubset merge^{+}$，这样分析过程将更快到达不动点，然而分析精度可能会有所下降。

对于 CPA 形式化体系中 *stop* 操作的配置，一般以到达不动点为分析结束条件。我们认为如果经过所有的迁移关系，都没有新抽象状态的产生，则整个抽象解释的分析就到达不动点，形式化定义为 $stop^{join}(e,R)=true\rightarrow[e]e\subseteq\bigcup_{e'\in R}[e']$。

5　CPA 的分析实例

本节用一个实例来说明如何使用可配置程序分析 CPA 来对基于抽象解释的静态分析的分析阶段进行建模。本文选择的抽象域是经典的区间抽象域。经典的区间抽象域是最早用于分析变量取值范围的方法，可以发现，变量的上下界信息是一种非关系型抽象域，其最终达到的不动点抽象值表示变量在程序运行过程中的取值范围。

分析对象的源代码是一个开源静态分析工具 INTERPROC[10] 上的样例代码。源代码和该静态分析工具的分析结果如下所示。

```
begin /*(L4 C5) top*/
  x=0; /*((L5 C8)[|x=0|])*/
  y=0; /*((L6 C8)[|x>=0;-x+100>=0|])*/
  while x<=99 do /*((L7 C18)[|x>=0;-x+99>=0|])*/
    if x<=99 then /*((L8 C17)[|x>=0;-x+50>=0|])*/
      x=x+1;
      /*((L9 C14)[|x-1>=0;-x+50>=0|])*/
      y=y+1;
      /*((L10 C14)[|x-1>=0;-x+50>=0|]))*/
    else/*((L11 C8)[|x-50>=0;-x+100>=0|])*/
      x=x+1;
      /*((L12 C14)[|x-51>=0;-x+100>=0|])*/
      y=y+1;
      /*((L13 C14)[|x-51>=0;-x+100>=0|])*/
    endif; /*((L14 C10)[|x-1>=0;-x+100>=0|])*/
  done; /*((L15 C17)[|-x+100>=0|])*/
end;
```

其中，每行代码后的注释部分为该开源工具的分析结果，即变量 x 的取值范围。

首先将源程序转化到一个状态迁移系统，以方便之后的分析。源程序的 CFG 如图 4 所示。

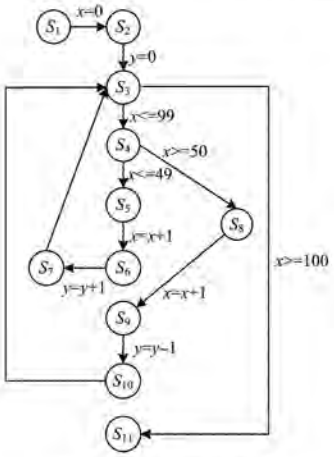

图 4　源程序的控制流图

使用可配置程序分析 CPA $\mathbb{D}=(D,\rightsquigarrow,merge,stop)$对此次静态分析进行建模，分别配置抽象域 D、抽象状态迁移关系$\rightsquigarrow$、抽象状态的合并操作 *merge*、终止判断操作 *stop*。下面给出具体的配置过程。

(1)配置抽象域 $D=(C,\varepsilon,[\cdot])$为经典区间抽象域。$C$ 为程序具体状态的集合$C\{pc,x\}$，其中 $pc\in\{S_1,S_2,\cdots,S_{11}\}$，$X=\{x,y\}$；描述抽象环境的完备格 $\varepsilon=(E,\sqsubseteq,\sqcap,\sqcup,\top)$，其中 E 是抽象状态集合，具体状态和抽象状态之间存在一个 Galois 连接。

$$(R,\leqslant)\underset{\gamma}{\overset{\alpha}{\leftrightarrows}}(Itvs,\sqsubseteq)$$

其中，R 是实数域，$Itvs$ 是实数域上的区间集合$\{[a,b]\mid a\in R\cup\{-\infty\},b\in R\cup\{+\infty\},a\leqslant b\}\cup\{\perp\}$。例如程序具体状态 $c(pc=S_3,\{x=0,y=0\})$转化成的抽象状态是 $e(pc=S_3,\{x\in[0,0],y\in[0,0]\})$。

(2)抽象状态迁移关系$\rightsquigarrow$，表示一个抽象状态通过图 4 所示的程序控制流图上连通的一边到达一个新的状态。例如抽象状态 $e(pc=c,i\in[1,1])$通过边$(pc=c,i=i+1,pc'=d)$到达抽象状态 $e'(pc=d,i\in[2,2])$；抽象状态 $e(pc=c,i\in[1,11])$通过边$(pc=b,assume(i\leqslant10),pc'=c)$到达抽象状态 $e'(pc=c,i\in[1,10])$。以此类推，以遍历寻找所有可能达到的抽象状态。

(3)合并操作符 *merge* 用以表示两个抽象状态的合并，在经典区间抽象域的抽象解释中，表示两个区间抽象域的合并，即 $merge^{join}(e,e')=e\sqcup e'$。两个区间的合并如下：

$$Itvs \sqcup Itvs' = \begin{cases} Itvs, & \text{if } Itvs' = \bot \\ Itvs', & \text{if } Itvs = \bot \\ [\min(a,a'),\max(b,b')], & \text{else} \end{cases}$$

其中，$Itvs=[a,b]$，$Itvs'=[a',b']$。

例如抽象状态 $e(pc=S_3,\{x\in[0,0],y\in[0,0]\})$与抽象状态 $e'(pc=S_3,\{x\in[1,1],y\in[1,1]\})$进行 *merge* 操作形成新的抽象状态 $e''(pc=S_3,\{x\in[0,1],y\in[0,1]\})$。

(4)终止判断操作 *stop* 用以判断迭代过程是否到达不动点。其配置为 $stop^{join}(e,R)=e\sqsubseteq\sqcup_{e'\in R}e'$。

在完成配置 CPA 之后，执行 CPA 执行算法。算法输入 $\mathbb{D}=(D,\rightsquigarrow,merge,stop)$，初始状态 $e_0=(pc=S_1,\{x=\bot,y=\bot\})$，经过 CPA 的迭代算法之后，得到算法的分析结果，如表 2 所列，其中每行表示对应程序阶段处的变量取值范围。

表 2 算法分析结果

结点	变量取值区间抽象域表示
S_1	$x=\bot,y=\bot$
S_2	$x[0,0],y=\bot$
S_3	$x[0,100],y[0,50]$
S_4	$x[0,99],y[0,50]$
S_5	$x[0,49],y[0,49]$
S_6	$x[1,50],y[0,49]$
S_7	$x[1,50],y[1,50]$
S_8	$x[50,99],y[1,50]$
S_9	$x[51,100],y[1,50]$
S_{10}	$x[51,100],y[0,49]$
S_{11}	$x[100,100],y[0,0]$

算法的分析结果与 INTERPROC 的分析结果相同，即通过对 CPA 形式化体系中各个参数的配置，有效地分析出了各个程序结点处变量的取值范围。在实际分析过程中，静态分析器需要根据具体情况调整分析的精度和效率，基于 CPA 的抽象解释分析方法，配置不同需求的分析过程。若需要分析变量的取值范围，则分析过程基于区间抽象域进行；若需要分析变量之间的约束关系，则分析过程基于关系型抽象域进行；若程序中的循环系数较大，分析无法在有效时间范围内完成，则配置 merge 操作，以期加快不动点的产生。

结束语 本文介绍了一种支持抽象解释的参数化的静态分析形式化体系，即可配置程序分析。用其对基于抽象解释的静态分析框架的分析阶段进行建模，并给出了程序状态迁移关系程序控制流图到 CPA 的转换方法，最后用一个实例说明了 CPA 在实际分析过程中的有效性。可配置程序分析可以通过参数的配置满足不同精度和效率需求的分析过程；同时，形式化的分析框架便于进行修改和扩展，例如在静态分析工具 Astree 中采用延时结合(delayed joins)和路径分割(path partitioning)[20]等技术来提高分析过程的效率和精度，这些技术在 CPA 的框架中可以通过 merge 操作的配置来实现。

目前可配置程序分析 CPA 仍有一些不足之处：首先该形式化体系还没有覆盖抽象解释中的 narrowing 算子和 widening 算子的含义，仍需要做一些改进来支持 narrowing 算子和 widening 算子，否则将无法保证分析的计算效率和终止性；其次，可配置程序分析是一种支持抽象解释和模型检测的形式化体系，在对模型检测的建模支持方面，仍然有许多工作需要进一步研究。

参 考 文 献

[1] CHESS B, MCGRAW G. Static Analysis for Security[J]. IEEE Security & Privacy Magazine, 2004, 2(2): 76-79.

[2] ZHANG J H, HUANG Z Q, CAO Z N. Counterexample Generation for Probabilistic Timed Automata Model Checking [J]. Journal of Computer Research and Development, 2008, 45(10): 1638-1645. (in Chinese)
张君华，黄志球，曹子宁. 模型检测基于概率时间自动机的反例产生研究[J]. 计算机研究与发展，2008，45(10)：1638-1645.

[3] JHALA R, MAJUMDAR R. Software model checking[J]. ACM Computing Surveys (CSUR), 2009, 41(4): 1-54.

[4] COUSOT P, COUSOT R, MAUBORGNE L. Theories, solvers and static analysis by abstract interpretation[J]. Journal of the ACM (JACM), 2012, 59(6): 1-56.

[5] COUSOT P, COUSOT R, FERET J, et al. The ASTRéE analyzer[C]// European Symposium on Programming. Springer Berlin Heidelberg, 2005: 21-30.

[6] BEYER D, HENZINGER T A, THÉODULOZ G. Configurable Software Verification: Concretizing the Convergence of Model Checking and Program Analysis[C]// International Conference on Computer Aided Verification. 2007: 504-518.

[7] CHEN L Q. Sound Floating-point and Non-convex Static Analysis Using Interval Linear Abstract Domains [D]. Changsha: National University of Defense Technology, 2010. (in Chinese)
陈立前. 基于区间线性抽象域的可靠浮点及非凸静态分析[D]. 长沙：国防科学技术大学，2010.

[8] LU C, HUANG Z Q, KAN S L, et al. Safety Analysis for Slats and Flaps Control Unit Based on Octagon Abstract Domain [J]. Journal of Chinese Computer Systems, 2016, 37(5): 902-907. (in Chinese)
陆陈，黄志球，阚双龙，等. 基于八边形抽象域的襟缝翼控制系统安全性分析[J]. 小型微型计算机系统，2016，37(5)：902-907.

[9] COUSOT P, CCOUSOT R. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints[C]// ACM Sigact-Sigplan Symposium on Principles of Programming Languages. 1977: 238-252.

[10] KÄSTNER D, FERDINAND C. Applying Abstract Interpretation to Verify EN-50128 Software Safety Requirements[C]// International Conference on Reliability, Safety and Security of Railway Systems. Springer International Publishing, 2016: 191-202.

[11] BLANCHET B, COUSOT P, COUSOT R, et al. A static analyzer for large safety-critical software[C]// Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation 2003. San Diego, California, USA, 2003: 196-207.

[12] CHEN L Q, WANG J, HOU S N. An Abstract Domain of One-Variable Interval Linear Inequalities [J]. Chinese Journal of Computers, 2010, 33(3): 427-439. (in Chinese)
陈立前，王戟，侯苏宁. 单变量区间线性不等式抽象域[J]. 计算机学报，2010，33(3)：427-439.

[13] MINUÉ A. The octagon abstract domain[J]. Higher-Order and Symbolic Computation, 2001, 19(1): 31-100.

(下转第 155 页)

参 考 文 献

[1] MARTIN J. An Information Systems Manifesto[M]. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1986.

[2] HU W S, ZHAO M, WU S Y, et al. Requires analysis based on software maintainability [C] // 2014 International Conference Reliability, Maintainability and Safety(ICRMS). 2014: 354-357.

[3] HECTOR M. Olagu_assessing mainability information theory metrics and iterative software processes [D]. Huntsville: Alabama, 2006: 7-33.

[4] CHIDAMBER R, KEMERER F. A metrics suite of object-oriented design [J]. IEEE Transactions on Software Engineering, 1994, 20(6): 467-493.

[5] ABREU F B E, CARAPUCA R. Object-Oriented software engineering: measuring and controlling the development process[C] // Proceedings of the 4th International Conference on Software Quality. McLean, Va, USA, 1994: 1-8.

[6] TEGARDEN D P, SHEETZ S D, MONARCHI D E. A software complexity model of object-oriented system [J]. Decision Support Systems, 1995, 13(3/4): 241-262.

[7] ETZKORN L, BANSIYA J, DAVIS C. Design and code complexity metrics for OO classes [J]. Journal of Object Oriented Programming, 1999, 12(1): 35-40.

[8] BANSIYA J, DAVIS C G. A Hierarchical Model for Object-Oriented Design Quality Assessment[J]. IEEE Transactions on Software Engineering, 2002, 28(28): 4-17.

[9] SURESH Y, KUMAR L, RATH S K. Statistical and Machine Learning Methods for Software Fault Prediction Using CK Metric Suite: A Comparative Analysis[C] // ISRN Software Engineering. 2014: 1-15.

[10] DUBEY S K, RANA A. Assessment of Maintainability Metrics for Object-Oriented Software System[J]. ACM SIGSOFT Software Engineering Notes Page1, 2011, 36(5): 1-7.

[11] LI X K, LIU Z T, PAN B, et al. Software and Research on Measure Experiments [J]. Chinese Journal of Computers, 2000, 23(11): 1220-1225. (in Chinese)
李心科，刘宗田，潘飚，等. 一个面向对象软件度量工具的实现和度量实验研究[J]. 计算机学报，2000，23(11)：1220-1225.

[12] 刘思峰，谢乃明. 灰色系统理论及其应用(第六版)[M]. 北京：科学出版社，2016.

[13] LIU S F, CAI H, YANG Y J, et al. Advance in grey incidence analysis modelling[J]. Systems Engineering-Theory & Practice, 2013, 33(8): 2041-2046. (in Chinese)
刘思峰，蔡华，杨英杰，等. 灰色关联分析模式研究进展[J]. 系统工程理论与实践，2013，33(8)：2041-2046.

[14] LORENZ M. Object-Oriented Software Development: A Practical Guide [M]. Englewood Cliffs, N. J.: PTR Prentice Hall, 1993.

[15] ROSENBERG L, STAPKO R, GALLO A. Object-oriented Metrics for Reliability [C] // Presentation at IEEE International Symposium on Software Metrics. 1999.

[16] SHATNAWI R, LI W, SWAIN J, et al. Finding software metrics threshold values using ROC curves [C] // Journal of Software Maintenance and Evolution: Reseach and Practice. Res (Pract 2010). 2010: 1-16.

[17] D'AMBROS M, LANZN M. Reverse engineering with logical coupling[C] // IEEE Computer Society Proceedings of the 13th Working Conference on Reverse Engineering. Washington, D C, USA, 2006: 189-198.

[18] KHAN T, BARTHEL H, EBERT A, et al. Visualization and Evoluton of Software Architectures[C] // Visualization of Large and Unstructured Data Sets Workshop. 2011: 25-42.

[19] ZHANG Y, TAO J, QIAN L Q. A Metrics Suite for Class Complexity Based on UML [J]. Computer Science, 2002, 29(10): 128-132. (in Chinese)
张涌，陶隽，钱乐秋. 一种基于 UML 的类复杂性度量方法[J]. 计算机科学，2002，29(10)：128-132.

[20] FU X D, ZOU P. A Measurement Method of structural complexity for UML class diagrams[J]. Computer Applications, 2007, 27(b06): 302-307. (in Chinese)
付晓东，邹平. 一种 UML 类图结构复杂性度量方法[J]. 计算机应用，2007，27(b06)：302-307.

[21] JING F B. Study on Software Complexity Measurement Method and Tool Based on UML [D]. Chongqing: Chongqing University, 2015. (in Chinese)
景富波. 基于 UML 的软件复杂性度量方法和工具的研究[D]. 重庆：重庆大学，2015.

[22] XIE L M. A Study on Class Diagram Design Flaws Detection [D]. Shanghai: East China Normal University, 2011. (in Chinese)
谢玲梅. 类图设计缺陷的检测研究[D]. 上海：华东师范大学，2011.

(上接第 130 页)

[14] COUSOT P, COUSOT R. A galois connection calculus for abstract interpretation[C] // ACM Sigplan-Sigact Symposium on Principles of Programming Languages. ACM, 2014: 3-4.

[15] COUSOT P, COUSOT R. Abstract interpretation: past, present and future[M]. ACM, 2014.

[16] BALL T, PODELSKI A, RAJAMANI S K. Boolean and Cartesian abstraction for model checking C programs[M] // Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2001: 268-283.

[17] HENZINGER T A, JHALA R, MAJUMDAR R, et al. Lazy Abstraction[C] // POPL. 2002: 58-70.

[18] FOSCHER J, JHALA R, MAJUMDAR R. Joining dataflow with predicates[J]. AcmSigsoft Software Engineering Notes, 2005, 30(5): 227-236.

[19] LALIRE G, ARGROUND M, JEANNET B. Interproc[OL]. http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/interproc.

[20] MAUBORGNE L, RIVAL X. Trace Partitioning in Abstract Interpretation Based Static Analyzers [M] // Programming Languages and Systems. Springer Berlin Heidelberg, 2005: 5-20.