数组遍历程序内存安全的完备性阈值

Tobias Reinhard
tobias.reinhard@kuleuven.be
imec-DistriNet，鲁汶大学，比利时鲁汶

Justus Fasse
justus.fasse@kuleuven.be
imec-DistriNet，鲁汶大学，鲁汶，比利时

Bart Jacobs
bart.jacobs@kuleuven.be
imec-DistriNet，鲁汶大学，比利时鲁汶

arXiv:2305.03606v1 [cs.LO] 5 May 2023

摘要

我们报告了阶段性研究结果。据我们所知，这是首个针对（部分）有界内存安全证明的完备性阈值研究。具体而言，我们考虑遍历数组且不分配或释放内存的堆操作程序。在这一设置下，我们首次提出了程序验证完备性阈值的概念，可将无界内存安全证明转化为（部分）有界证明。此外，我们证明可以刻画简单数组遍历程序类别的完备性阈值。最后，我们提出了从理论上（扩展至更广泛的程序类别，如堆操作和树形数据结构）和实践上（突出自动化机会）扩展这一技术的研究方向。

CCS 概念：• **计算理论** → **程序验证**；**分离逻辑**；**通过模型检测进行验证**。

关键词：程序验证、完备性阈值、内存安全、有界证明、模型检测、分离逻辑

1 引言

无界证明与有界证明。人们已经开发出许多技术，以确信软件值得信赖。内存安全是任何更高层次属性的基础支柱。在内存不安全语言中，证明责任落在程序员身上。然而，证明内存安全仍然很困难，通常需要编写繁琐的归纳证明。自动化验证过程的一种方法是采用有界证明，并接受有界保证。

考虑一个程序 c，它会搜索大小为 s 的数组。对于 c 的无界内存安全证明可以表明，该程序对任何可能的输入都是安全的，尤其是对任意数组大小都安全，即 $\forall s.\,\mathrm{memsafe}(c(s))$。只考虑输入大小不超过 10 的有界证明，只能保证程序对这类有界数组都是安全的，即$\forall s \leq 10.\,\mathrm{memsafe}(c(s))$。

完备性阈值。通过有界证明近似无界证明，是一种常用于模型中的技术。检查。因此，关于有限状态转换系统的有界证明与无界证明之间的关系已得到广泛研究 [1, 4, 7, 13, 20, 30, 35, 38]。对于有限状态转移系统 T 和感兴趣的属性 ϕ，完备性阈值是指任何一个数值 k，使得我们只需检查 T 中长度为 k 的路径前缀，就能证明 ϕ，即 $T \models_k \phi \Rightarrow T \models \phi$ [20] [1]。多年来，许多研究针对不同类型的属性 ϕ，刻画了最小完备性阈值的上界。这些上界通常根据转换系统 T 的关键属性来描述，例如递归直径（最长无环路径）[35]。例如，考虑有限状态转移系统T中形式为 $G p$ 的全局安全属性类，其中 p 是一个局部属性。我们知道，仅用 T 的直径表示的该类属性的最小完备性阈值恰好为 $\mathrm{diam}(T)$ [7, 34]。由于形式为 $G p$ 的安全属性也适用，因此这同样适用于有限状态转移系统的内存安全。

一般而言，堆操作程序的状态空间可能是无限的。这是因为程序的输入数据可能任意大，并且执行过程也可能任意长。因此，上述关键属性通常也会是无限的。我们所关注的程序与已研究完备性阈值的转换系统之间存在巨大的结构差异，这使我们无法直接复用任何现有定义或结果。

2 有界证明的局限性

有界模型检测。 一般而言，如果我们希望获得无界的内存安全保证，就必须考虑所有可能的输入大小和所有可能的执行过程。这通常很困难，并且需要我们编写繁琐的归纳证明。另一种方法是放弃无界保证这一理念，转而接受有界保证。在关键软件开发过程中，已被证明很有用的一种方法是有界模型检查（BMC） [15]。

其基本思路是用一个可以自动检查的有限模型来近似原始验证问题。采用这种方法时，我们选择一个大小界 S

[1]请注意，文献中对术语完备性阈值的使用并不一致。一些论文，例如 [20]，采用上述定义，因此完备性阈值并非唯一。另一些论文，例如[35]，则将其定义为满足 $T \models_k \phi \Rightarrow T \models \phi$的最小数量 k，因此完备性阈值是唯一的。

```
对于 i 属于 ⟦L : s-R⟧ ，执行
  !a⟦i+Z⟧
```
$\Big\}=:\ trav^{Z}_{L,\,R}(a,s)$

$$Trav \;:=\; \{trav^{Z}_{L,R} \;\mid\; L,R,Z \in \mathbb{Z}\}$$

图 1. 程序类别 $Trav$ $trav^{Z}_{L,\,R}$遍历大小为 s 的数组a，尝试读取元素。L，R，Z是常量。

并且只考虑大小不超过 S的输入。此外，我们也只检查有限的执行前缀。一种常见方法是将循环和递归展开到一定深度。

这种方法背后的直觉是，如果程序包含错误，那么这些错误很可能已经出现在较小的输入大小和较早的循环迭代中。只要有界模型检查不执行抽 [16], 象，所有报告的反例都是真实的错误。然而，我们应当谨慎，不要忘记，通过这种方式，我们只能获得一个有界证明，从而得到有界保证。

数组遍历模式。考虑图$Trav$ 1 所示的、使用指针算术的WHILE语言中的程序类$Trav$ 。给定指针 a 和变量 s，使得 a 指向一个大小为 s的数组，每个程序都会遍历该数组并尝试读取元素。该类模拟了一种基本的编程模式以及常见的差一错误 [23]。我们使用大写字母表示常量，使用小写字母表示（程序）变量。此类中的程序 $trav^{Z}_{L,R}$ 从 $i = L$ 迭代到 $i = s - R$（包括界限），并尝试读取数组在索引 $i + Z$处的元素。我们使用 $!x$ 来表示对堆位置 x的访问。对于具体实例$trav^{Z}_{L,R}$，是否发生内存错误取决于常量 L、R、Z的取值方式。本文将其作为贯穿全文的最小示例。

有界证明可能会出现什么问题？为了说明这一问题，我们使用有界模型检查来检查数组遍历模式的几种不同实例：(i) 遍历整个数组：$trav^{0}_{0,1}$, (ii) 遍历数组时，访问位置相对于索引偏移两个单位：$trav^{2}_{0,1}$ 以及 (iii) 将索引变量的上界额外减小一个单位：$trav^{2}_{0,2}$。显然，(i) 是内存安全的，而 (ii) 和 (iii) 则不是。然而，在运行模型检测算法之前，我们必须选择合适的界限。我们所考察的模式相当简单。因此，我们为有界模型检查过程选择大小界 $S = 1$ 和展开深度 $D = 1$。请注意，后者实际上意味着：对于我们所选择的输入大小，不限制循环深度。

对于标准变体 (i)，我们在界限内找不到任何错误。这是合理的，因为该程序是安全的。在变体 (ii) 中，数组访问 $a[i + 2]$ 被错误地向右偏移。这会导致大小为 1 的数组发生越界错误。该大小处于我们选定的界限内，因此有界模型检查会报告此错误，我们可以据此进行修正。最后，变体 (iii) 将索引变量的取值范围缩减为 $[0, s - 2]$，这意味着该程序对大小为 $s \geq 2$的数组只执行循环迭代。因此，对于大小为 0 和 1 的数组，程序显然是安全的。这正是我们的有界证明所检查的大小。因此，有界模型检查不会报告任何错误，导致我们错误地认为$trav^{2}_{0,2}$ 是安全的。

3 完备性阈值

如上所示，有界证明通常是无界证明的不可靠近似。具体近似只有在我们选择足够大的界限时才是可靠的，这样才能确保不会遗漏任何错误。我们重点讨论对输入大小（在我们的示例中为数组大小）设定界限，而忽略不依赖这些参数的循环界限。

回顾 § 1，完备性阈值是有限状态转移系统模型检测中的一个概念 [20]。我们借用这一术语，并将其应用于内存安全验证。因此，对于输入参数为 $x \in X$ 的程序 $c(x)$，如果只考虑来自 Q 的输入就能证明 c 的内存安全性，即 $\forall x \in Q.\,\mathrm{memsafe}(c(x)) \Rightarrow \forall x \in X.\,\mathrm{memsafe}(c(x))$，那么我们称任何子定义域 $Q \subseteq X$ 为 x 在 c 上的完备性阈值（CT）。

直观的 ***CT*** 提取。回到我们举例的那类实现数组遍历模式的程序$Trav$ 。其中有些是内存安全的，有些则不是。因此，让我们尝试为这些程序计算完备性阈值。首先，看看可能发生哪些错误。这能让我们了解可靠的有界证明必须涵盖哪些大小。任意实例 $trav^{Z}_{L,R}(a, s)$ 按升序迭代 $i = L, \ldots, s-R$并访问 $a[i + Z]$。对于使升序范围$L, \ldots, s - R$为空的大小 s，循环根本不会执行。任何这类运行显然都是内存安全的。因此，任何有意义的 $trav^{Z}_{L,R}(a, s)$ 有界证明都必须包含满足 $\{L, \ldots, s - R\} \neq \emptyset$ 的大小 s，也就是说，$s \geq L + R$。

假设 $s \geq L + R$。如果索引 $i + Z$违反数组边界，即 $i + Z < 0$ 或 $i + Z \geq s$，则会发生错误。考虑到索引范围，我们可以得出，当 $L + Z < 0$ 或 $s - R + Z \geq s$ 成立时，就会发生错误。我们可以将后者简化为 $Z - R \geq 0$。

请注意，$L + Z < 0$ 和 $Z - R \geq 0$ 均不依赖于数组大小 s。这意味着，只要我们关注大于阈值 $s \geq L + R$ 的大小，s 的具体取值就不会影响是否发生错误。换句话说，对于我们的有界证明，只需检查单个（任意选取的）大小 $q \geq L + R$，然后即可将结果外推，即，

$$\forall a.\,\mathtt{memsafe}(trav^{Z}_{L,R}(a, q)) \;\Rightarrow\; \forall s.\forall a.\,\mathtt{memsafe}(trav^{Z}_{L,R}(a, s))$$

因此，任何集合 $\{q\}$ for $q \geq L + R$ 都是数组大小参数 s 在$trav^{Z}_{L,R}$中的一个完备性阈值。我们刚刚为整个类$Trav$的 CT 找到了统一的刻画。请注意，$\{q\}$不一定是最小的 CT。对于 $trav^{0}_{0,1}$, 这类安全实例，空集 $\emptyset$ 也是有效的 CT。

我们的方法。我们通过研究 x 在 $c(x)$ 中的验证条件（VC），来研究其 CT。后者是一个自动生成的逻辑公式，其形式为 $\forall x \in X.\, vc(x)$，并证明对于 x 的所有取值，该公式蕴含 $c(x)$ 的内存安全性。接下来，我们目前通过手动简化 $vc(x)$，直到能够明确看出 x 的选择如何影响 $vc(x)$ 的有效性。了解这一点使我们能够将定义域划分为 $X = \text{Đ} Q_i$。对于每个子定义域，我们得到 $vc_i(x) = \forall x \in Q_i.\, vc(x)$。如果可能，我们根据受限子定义域 Q_i 将每个 $vc'_i(x)$ 简化为 $vc''_i(x)$，目标是消除 x 的出现。如果 vc''_i 不涉及 x，我们就从 Q_i 中任意选取一个元素作为代表 Q'_i。否则，$Q'_i = Q_i$。因此，$\text{Đ}\, Q'_i$ 是 $c(x)$ 中 x 的一个 CT。接下来，我们将对此进行更详细的阐述。

3.1 通过验证条件近似 CT

现在我们已经对*Trav*的 CT 有了直观认识，接下来将上文的非正式论证转化为正式论证。我们所考虑的语言和逻辑的形式化定义，以及文中引理的证明，可见技术报告 [44]。

霍尔三元组。我们使用霍尔三元组 [31] 来表示程序规约。三元组 $\{A\}\ c\ \{B\}$ 表示，对于每个从满足前置条件 A的状态开始的执行，下列性质均成立：首先，执行不会遇到任何运行时错误。其次，执行要么 (i) 不终止，要么 (ii) 终止于满足后置条件 B的状态。

在本研究中，我们考察不会改变其所处理数据结构形状的程序的内存安全性。因此，我们选择的前置条件仅描述程序输入数据结构的内存布局。对于数组遍历程序，我们选择谓词 $\mathsf{array}(a,\ s)$ 作为前置条件，它表示 a指向一个大小为 s 的连续内存块。鉴于目标程序不会改变内存布局，规约可简化为 $\{A\}\ c\ \{A\}$。对于数组遍历，我们得到 $\{\mathsf{array}(a,\ s)\}\ trav^{Z}_{L,\ R}\{\mathsf{array}(a,\ s)\}$。

定义 3.1（程序的完备性阈值）。设 $\{A\}\ c\ \{B\}$ 为一个包含自由变量 x 且其定义域为 X 的程序规约。若子定义域 $Q \subseteq X$ 满足以下条件，则称其为 x 在 $\{A\}\ c\ \{B\}$ 中的完备性阈值 if

$$\models \forall x \in Q.\ \{A\}\ c\ \{B\} \quad \Rightarrow \quad \models \forall x \in X.\ \{A\}\ c\ \{B\}$$

当上下文清晰时，我们会省略前置条件和后置条件的具体表述。相反，我们称 Q 是程序 c中 x 的完备性阈值。

分离逻辑。我们使用一种带递归谓词 [40, 41, 45]的一阶仿射／直觉主义分离逻辑来描述内存。由于我们关注堆操作程序，因此使用断言来描述堆。分离逻辑包含几个特殊运算符：(i) 指向块 $x \mapsto v$ 描述一个包含位置 x 的堆，该位置存储值 v。我们用 $x \mapsto _$ 表示不关心堆单元中存储的值。(ii) 分离合取 $a_1 * a_2$ 表示 a_1 和 a_2 描述的是不相交的堆。因此，$x \mapsto _ * y \mapsto _$ 蕴含$x \neq y$。(iii) 分离蕴含 $a_1 \mathrel{-\!\!*} a_2$ 可读作 a_2 不含 a_1。也就是说，将 $a_1 \mathrel{-\!\!*} a_2$ 描述的堆与 a_1 描述的不相交堆组合起来，得到的堆符合 a_2。(iii) 在我们的逻辑中，持久性模态 $\Box a$ 表示 a 不描述资源，因此在空堆下成立（参见 [33]）。

我们假设 array 表示一个（递归定义的）谓词，对于每个固定大小 s，都可以将其表示为迭代的分离合取：$\mathsf{array}(a, s) \equiv \ast_{0 \le k < s} a[k] \mapsto _$.

验证条件。验证程序的一种常见方法是通过验证条件 [29, 43]。对于任意规约 $\{A\}\ c\ \{B\}$，验证条件（验证条件）是指满足以下条件的任意逻辑公式 vc：我们可以通过证明 vc 来验证 $\{A\}\ c\ \{B\}$，即 $\models vc \Rightarrow \models \{A\}\ c\ \{B\}$。

定义 3.2（验证条件）。如果断言 a 满足以下条件，我们就称其为关于 $\{A\}\ c\ \{B\}$ 的验证条件：

$$\models a \quad \Rightarrow \quad \models \{A\}\ c\ \{B\}.$$

定义 3.3（断言的完备性阈值）。设 a为一个断言，其自由变量 x的定义域为 X。若子定义域 $Q \subseteq X$ 满足以下条件，我们就称其为 x 在 a中的完备性阈值：

$$\models \forall x \in Q.\ a \quad \Rightarrow \quad \models \forall x \in X.\ a.$$

考虑一个规约 $\{A\}\ c\ \{B\}$，它包含一个自由变量$x \in X$，并有一个对应的验证条件 $\forall x \in X.\, vc$。假设我们得到了 x在 vc中的完备性阈值 Q。已知这个阈值后，规约的正确性就可以归约为有界验证条件，即 $\models \forall x \in Q.\, vc \Rightarrow \{A\}\ c\ \{B\}$。也就是说，我们可以从有界证明推导出无界保证。当量化定义域从上下文中可以明确得知时，我们通常会省略它们。

最弱自由前置条件。生成验证条件的一种常见方法是使用最弱自由前置条件 [24, 29]。对于任意程序 c 和后置条件 B，最弱自由前置条件 $\mathsf{wlp}(c, \lambda r.\, B)$ 是一个断言，满足

$$\forall A.\ \left(A \models \mathsf{wlp}(c, \lambda r.\, B) \ \Rightarrow \models \{A\}\ c\ \{B\} \right)$$

成立。也就是说，如果初始状态满足最弱自由前置条件，那么 c 要么不会终止，要么会在一个满足后置条件 B 的状态下终止。特别是，执行过程中不会发生内存错误。对于 $\{A\}\ c\{B\}$，其规范验证条件为 $\forall \overline{x}.\, A \to \mathsf{wlp}(c,\ \lambda r.\, B)$，其中$x$ 是 A、c 和 B 中自由出现的变量所组成的元组。

***CT* 的局限性。** 一般而言，验证条件是过度近似。因此，从验证条件推导出的 CT 并不总是适用于相应的程序。考虑以下规约

$\{\mathsf{array}(a,s)\}$ 遍历$^2_{0,2}$ $\{\mathsf{array}(a,s)\}$，以及任意不可满足的断言 $vc_{\text{false}} \equiv \mathsf{False}$。那么 $\forall s \in \mathbb{N}.\, vc_{\text{false}}$ 是一个过度近似的验证条件，因为 $\mathsf{False} \Rightarrow \models \{\mathsf{array}(a,s)\}$ 遍历$^2_{0,2}$ $\{\mathsf{array}(a,s)\}$ 如 § 2 所述，仅涵盖大小 0、1 的有界证明无法发现 $\mathrm{trav}^2_{0,2}$ 中的错误。因此，集合 $\{0\}$ 不是 $\{\mathsf{array}(a,s)\}$ 中 x 的 CT：$\mathrm{trav}^2_{0,2}\{\mathsf{array}(a,s)\}$。然而，对于 s in vc_{false}，$\{0\}$ 是 CT，因为 $\forall s \in \{0\}.\, vc_{\text{false}} \equiv \mathsf{False}$ 且 $\mathsf{False} \Rightarrow \models \forall s \in \mathbb{N}.\, vc_{\text{false}}$。我们看到，只有当 vc 在关于 x 时不会过度近似，才能将从 VC vc 推导出的、针对某个变量 x 的 CT Q 转移到对应的程序。（我们目前正在研究这种联系。）不过，值得注意的是，本文讨论的示例确实满足这一条件。即便 vc 会过度近似，CT Q 仍适用于任何考虑至少与 vc 一样强的性质的证明。

顾名思义，由最弱前置条件推导出的验证条件所蕴含的性质非常弱。合理的假设是，有界证明通常会蕴含基于 wlp 的验证条件的有界版本。因此，在研究 CT 时使用它们是合理的。

通过验证条件提取 *CT*。 将上述方法应用于我们的规约 $\{\mathsf{array}(a,s)\}\ \mathit{Trav}^Z_{L,R}(a,s)\ \{\mathsf{array}(a,s)\}$，我们得到一个验证条件 $\forall a.\, \forall s.\, vc_{\text{trav}}$，其中 vc_{trav} 如下 [2]：

$$
\begin{aligned}
vc_{\text{trav}} \;:=\; & \mathsf{array}(a,s) \rightarrow && \text{(\#1)}\\
& \mathsf{array}(a,s) && \text{(\#2)}\\
& * \ \square(\forall i.\ (L \le i \le s-R) \wedge \mathsf{array}(a,s) \mathrel{-\!\!*} && \text{(\#3)}\\
& \qquad \exists v.\ a[i+Z] \ \mapsto\ v \wedge \mathsf{array}(a,s))\\
& * \ (\mathsf{array}(a,s) \mathrel{-\!\!*} \mathsf{array}(a,s)) && \text{(\#4)}
\end{aligned}
$$

此处，(#1) 是描述我们内存布局的前置条件。分离合取 (#2)-(#4) 是从我们的规约中推导出的最弱前置条件，并且 vc_{trav} 表明它应当由前置条件 (#1) 推出。

最弱前置条件指出，内存布局在循环执行期间必须保持不变。(#2) 表明，这一条件必须在循环开始之前成立。(#3) 要求每次循环迭代都保持该布局，即布局描述是一个循环不变式。(#4) 表明，该不变式蕴含规约中的后置条件；而后置条件同样是未发生变化的内存布局。

请记住，我们使用的是仿射分离逻辑。显然，这个验证条件包含许多显然冗余的部分。我们可以将 vc_{trav} 化简为 vc_1：

$$
\begin{aligned}
vc_{\text{trav}} \ \equiv\ & \texttt{// Eliminate (\#2), (\\#4)}\\
& \mathsf{array}(a,s) \rightarrow\\
& \quad \square(\forall i.\ (L \le i \le s-R) \wedge \mathsf{array}(a,s) \mathrel{-\!\!*}\\
& \qquad \exists v.\ a[i+Z] \ \mapsto\ v \wedge \mathsf{array}(a,s))\\
\equiv\ & \texttt{// Persistency makes pre. obsolete}\\
& \quad \square(\forall i.\ (L \le i \le s-R) \wedge \mathsf{array}(a,s) \mathrel{-\!\!*}\\
& \qquad \exists v.\ a[i+Z] \ \mapsto\ v \wedge \mathsf{array}(a,s))\\
\equiv\ & \texttt{// } \mathsf{array}(a,s) \text{ equiv. to } \mathop{\Large *}_{0\le k<s} a[k] \ \mapsto\ _\\
& \quad \forall i.\ (L \le i \le s-R) \rightarrow (0 \le i+Z < s)\\
=:\ & vc_1
\end{aligned}
$$

这个等价的验证条件 $\forall a.\, \forall s.\, vc_1$ 确实反映了我们在非正式分析程序时形成的直觉：对于大小为 $s < L+R$的数组，程序不会进行任何循环迭代，因此显然是内存安全的。对于更大的数组，当且仅当索引 $i+Z$ 超出数组边界时，才会发生内存错误。

我们可以通过将 s 的定义域划分为 $\mathbb{N} = \{0,\dots,L+(R-1)\} \cup \{L+R,\dots\}$ 来证明这一直觉。让我们分别分析这两个子域。对于大小为 $s_- < L+R$ 的情况，我们得到

$$vc_1(s_-) \quad\equiv\quad \forall i.\, \mathsf{False} \rightarrow (0 \le i+Z < s_-) \quad\equiv\quad \mathsf{True}$$

因此，我们无需费心检查大小为 $s < L+R$ 的情况。对于更大的大小 $s_+ \ge L+R$，我们得到

$$
\begin{aligned}
& vc_1(s_+) \ \equiv\ \forall i.\ (L \le i \le s_+ - R) \rightarrow (0 \le i+Z < s_+)\\
\equiv\ & \forall i.\ (L \le i \rightarrow 0 \le i+Z) \ \wedge\ (i \le s_+ - R \rightarrow i+Z < s_+)\\
\equiv\ & \forall i.\ (L \le i \rightarrow 0 \le i+Z) \ \wedge\ (i \le -R \rightarrow i+Z < 0)\\
=:\ & vc_2.
\end{aligned}
$$

由于 s_+ 在 vc_2中不自由出现，$vc_1(s_+)$的真假不取决于 s_+的选择。请记住，我们有$vc_{\text{trav}}(s_-) \equiv \mathsf{True}$，且 $vc_{\text{trav}}(s_+) \equiv vc_2$。因此，

$$\models \forall a.\, \forall s.\, vc_{\text{trav}} \quad\Leftrightarrow\quad \models \forall a.\, vc_{\text{trav}}(s_+)$$

我们可以看到，只需针对任意大小 $s_+ \ge L-R$ 检查原始验证条件 vc_{trav}，即可证明我们的数组遍历程序 $\mathit{trav}^Z_{L,R}$ 内存安全。也就是说，$\{s_+\}$ 是一个完备性阈值。

通过约束刻画完备性阈值。 请注意，我们刚刚推导出的约束 $s \ge L-R$ 是对整个 Trav 类的完备性阈值的一种统一表示。我们经常使用约束来简洁地刻画完备性阈值。约束描述的性质必须足够强，使得每个包含该性质的子定义域都是一个完备性阈值。对于每个程序 $\mathit{trav}^Z_{L,R} \in \mathit{Trav}$，如果 $Q' \cap \{s \in \mathbb{N} \mid s \ge L-R\} \neq \emptyset$，那么每个子定义域 $Q' \subseteq \mathbb{N}$ 都是数组大小 s 的一个完备性阈值。

3.2 完备性阈值的模块化

无关数据结构。 考虑程序求和$^Z_{L,R}(a,\ s,\ n)$，它尝试对数组 a 的所有元素求和，并将结果写入堆位置 n：

[2]最弱前置条件演算要求我们使用循环不变式对循环进行注释。在我们研究的场景中，初始内存布局在程序执行过程中保持不变。我们考虑的前置条件恰好描述了初始内存布局，不包含其他内容。因此，在最弱前置条件计算过程中，我们可以将前置条件重用为循环不变式。

```
对于 i，满足 [L : s-R] ，执行 (
  n := a[i+Z];
  复杂_fct (n, y,        k)
)
```

$$=: \quad comp^{Z}_{L,R}(a, s, y, k)$$

$M(a, s)$:= 数组$(a,\ s)$ $*$复杂_数据$(y,\ k)$

$Comp$:= $\{\{M\}comp^{Z}_{L,\,R}\{M\} \mid L, R, Z \in \mathbb{Z}\}$

图 2. 涉及复杂数据结构和不依赖于数组大小 s的计算的程序类。

```
for i in [L : s-R] do
  !n := !n + !a[i+Z]
```

$$=: \quad sum^{Z}_{L,R}(a, s, n)$$

与 $trav_{L,\,R}$ 类似，它遍历数组并尝试读取数组元素。此外，它使用读取的值更新存储在堆位置 n处的总和。假设数组 a 与结果变量 n 不别名。我们得到规约 $\{A\}$ 求和$^{Z}_{L,\,R}\{A\}$，针对 $A :=$ array$(a,\ s)*n \mapsto _$。我们假设采用结构化内存。因此，无法通过数组访问 $a[...]$来访问 n。

直观来看，数组大小显然不会影响对堆位置 n的内存访问。因此，$sum^{Z}_{L,R}$ 中 s 的 CTs 应与$trav^{Z}_{L,R}$中的相同。事实上，对$sum^{Z}_{L,R}$的基于 wlp 的验证条件进行分析，证实了这一直觉。它具有$\forall a.\forall s.\forall r.\ vc_{\text{sum}}$的形式，并且我们可以将$vc_{\text{sum}}$ 改写为

$$vc_{\text{sum}} \quad \equiv \quad vc_{\text{trav}} * (n \mapsto _ \to A)$$

其中 vc_{trav} 是 § 3.1 中数组遍历程序 $trav^{Z}_{L,R}$ 的验证条件。此外，freeVars$(A) = \{n\}$。由于 s 在 $n \mapsto _ \to A$中不是自由出现的，因此在为 vc_{sum} 中的 s 寻找完备性阈值时，我们可以忽略它。

引理 3.4（验证条件切片）。设 a、a_x、a_y 为断言，满足 $x \in$自由变量(a_x)、$x \notin$自由变量(a_y) 以及 $a \equiv a_x * a_y$。设 $Q \subseteq X$ 是 a_x 中 x 的一个完备性阈值（CT）。那么，Q 也是 a 中 x 的一个完备性阈值，即，

$$\models \forall x \in Q.\ \forall \overline{y} \in \overline{Y}.\ a \quad \Rightarrow \quad \models \forall x \in X.\ \forall \overline{y} \in \overline{Y}.\ a$$

我们可以将求和$^{Z}_{L,\,R}$示例中的观察结果推广到更复杂的程序类别。考虑图2中介绍的$Comp$类。程序$comp^{Z}_{L,\,R} \in Comp$ 接收两个不相互别名的数据结构：一个大小为 s 的数组 a，以及一个大小为 k 的复杂数据结构 y，后者由谓词 complex_data$(y,\ k)$ 描述。$comp^{Z}_{L,\,R}$ 从数组 a 中读取元素，将结果存储在一个局部变量 n 中，然后调用一个复杂函数 complex_fct(n,y,k)，该函数既不依赖于 a，也不依赖于 s。验证条件（VC）vc_{comp} 将体现这一点。也就是说，类似于上面的示例，它应当能够表示为 $vc_{comp} \equiv vc_{\text{trav}} *$ (complex_data$(y,\ k)$ $\to \ldots$)，其中右侧的合取项不依赖于 s。

验证条件切片引理3.4 告诉我们，每当要刻画特定参数的完备性阈值时，都可以忽略所有不涉及该参数的独立验证条件合取项。实际上，这意味着我们可以忽略所有复杂的与搜索 s 的 CT 时与数组大小无关的 完备性阈值$^{Z}_{L、\,R}$ 部分。这使我们能够将搜索缩小到 $Trav$ 的 CT。

组合性。我们可以用复杂程序的构建块的完备性阈值来描述这些程序的完备性阈值。考虑程序 $c_1; c_2$，并假设 c_1 和 c_2 是我们之前研究过的模式实例。由此可知，每个 c_i 都对应一个验证条件 $\forall x.\ vc_i$，其针对 x 的完备性阈值为 Q_i。设 $vc_{1;2}$ 是我们要证明的 $c_1; c_2$ 的验证条件。假设它可以改写为 $\forall x.\ vc_1 \wedge vc_2$。那么，我们知道 $Q_1 \cup Q_2$ 是 $vc_{1;2}$ 的完备性阈值。因此，我们研究完备性阈值的方法是研究模式和组合子。

基本模式。我们将基本模式（例如上文讨论的数组遍历模式）视为基本构建块。它们往往频繁出现在程序中，而且足够简洁，因此可以通过研究其验证条件来提取完备性阈值。具体而言，我们关注能够保持内存布局的遍历模式和访问模式。目前，我们聚焦于数组，但之后会将其推广到任意归纳数据结构。

管理复杂性。我们的主要目标之一是描述有趣程序类别的完备性阈值。随着程序变得更加复杂，验证条件往往会迅速变得非常复杂。因此，我们需要一种处理这种复杂性的方式，并将完备性阈值分析拆解为更简单的问题。遵循我们想要推理的程序结构，是一种自然的方法。

组合子模式。为了在分析 CT 时利用程序结构，我们需要研究控制结构如何影响 CT。例如，如上所述，我们可以将顺序结构 $c_1; c_2$ 的 CT 描述为从 c_1 和 c_2 得出的 CT 的并集。此外，考虑命令 if $e$then c_1 else c_2，并假设我们可以通过约束 Ó K_i 分别描述每个 c_i 中大小为 s 时的 CT。那么，我们可以通过约束 $(e \wedge \text{Ó}K_1) \ \vee (\neg e \wedge \text{Ó}K_2)$ 描述整个命令的 CT。

4 结论

以往的程序验证方法，要么针对无界保证，并依赖无界（通常为归纳式）证明；要么针对有界保证，并尝试利用有界模型检测等技术来近似程序行为。然而，这两个领域之间的互动很少。

在这项工作中，我们提出了一种关于内存安全证明的新视角，将无界证明与有界证明联系起来。我们证明，对于某些遍历数组并保持内存布局不变的程序，可以将无界内存安全证明简化为有界证明。对于任何此类程序，只需考虑少数几个经过选择的数组大小，即可获得与考虑所有可能大小的数组相同的保证。我们

将这一概念称为完备性阈值，以此类比有限迁移系统模型检测中的一个类似概念。此外，我们表明，研究验证条件是研究完备性阈值的一种恰当方式。

5 相关工作

完备性阈值最初由 Kroening 和 Strichman [35]提出。迄今为止，对 CT 的研究仅限于有限状态系统。事实上，针对各类 LTL 性质的知名 CT，是根据相关有限状态系统的（递归）直径定义的（例如，[7, 34]）。通过迭代展开来确定程序的最坏执行时间并发现循环的上界，也可用于确定 CT[19, 26]。对于可能具有无限状态的系统，这些 CT 自然也可能是无限的。通过专门研究内存安全这一性质，我们也能够刻画并可能找到适用于这些系统的有用 CT。针对相同（例如，互模拟 [12] 或同构 [28]）进程的参数化网络拓扑进行模型检测时，会涉及一种与完备性阈值相关的概念，称为 截断界限。也就是说，在截断界限范围内进行模型检测，即可推断将拓扑扩展至无限多个进程时仍然正确。对于这类令牌环的性质，已有正面结果 [12, 28]，其他拓扑也有相关结果 [2, 22]。

模型检测文献（参见 [18]）提出了大量替代方法，以获得有限状态系统的无界保证，例如 k归纳 [8, 46]、Craig 插值 [38] 和属性导向可达性 [10, 27]，并将其调整用于软件验证场景（例如 [5, 6, 25]）。

数组操作程序在不同定义域中都得到了充分研究 [9, 11, 14, 32]。不过，我们认为，本文的主要贡献在于提出了一种新方法，将关于内存安全的无界证明与有界证明联系起来。最终，正如展望部分所述，我们希望将此方法推广并自动化，使其适用于树形数据结构。在这方面，Mathur 等人 [37] 也考虑了证明堆操作程序内存安全这一特殊情形。如果初始堆呈森林状，且程序仅对该数据结构进行一次遍历，他们证明了内存安全是可判定的（另见 § 6）。他们未涉及数组和缓冲区溢出，但支持内存的分配与释放。

6 展望

程序的 ***CT***。 考虑任意一个包含自由变量 x 及相应验证条件 vc的程序 c。假设我们在 vc中为 x 推导出了一个 CT Q。一般而言，这并不能让我们得出结论：Q 在 c中也一定是 x 的 CT。直观而言，只有当 vc 在 x方面没有过度近似时，这一结论才成立。本文研究的程序和验证条件符合这一点。尽管验证条件的 CT 仍能为针对这些验证条件的证明提供一些信息，但我们的最终目标是推导程序的阈值。因此，我们目前正在研究这种联系。

可扩展性。 在这项工作中，我们聚焦于一种受限的数组遍历模式，以展示 CT。我们的目标是将这种方法扩展到复杂程序。因此，我们目前正在研究更多数组遍历模式、访问模式和组合子。这将帮助我们更好地理解程序的结构如何影响有界证明与无界证明之间的关系。了解这一点将使我们能够刻画适用于复杂程序类别的 CT，这些类别涵盖差一错误以外的错误。

之后，我们将扩展我们的方法，使其能够涵盖任意归纳数据类型。具体而言，我们计划为一类程序描述完备性阈值，这类程序包括涉及嵌套循环的原地排序算法的（安全和不安全）实现。完成这项工作后，我们将研究内存分配和内存释放如何影响完备性阈值。

可判定性。 我们观察到，排序函数中的内存访问通常主要取决于已排序数据结构的大小、遍历策略以及比较关系 $<$。假设我们成功为 *Sort* 类推导出了有限的 CT，那么我们推测，这足以得出 *Sort* 类的内存安全具有可判定性的结论。值得注意的是，这一针对性结果将超出 Mathur 等人 [37]相关工作的研究范围。后者证明了某种特定形式的单遍程序的内存安全具有可判定性，即恰好遍历一次数据结构的程序。

改进有界模型检查的保证。 有界模型检测受到状态空间爆炸问题的影响 [16, 17, 42]，而该问题与所选界限有关。为了使验证时间保持在可接受范围内，通常只能对每个参数取非常小的界限进行检查。我们计划将本文介绍的方法实现自动化，并利用现有的静态分析技术（例如 [3, 36, 39, 47]）来简化生成的验证条件。一旦能够自动计算某个参数（例如被遍历数据结构的大小）的较小完备性阈值，就可以相应地调整界限。这样，我们可以确保不会遗漏任何可能导致内存错误的大小。因此，这将增强有界证明所提供的保证。最终，我们希望与工业级有界模型检测工具 CBMC 集成[19, 21]。

此外，这还将缩短验证时间，并释放资源，以便用于探索程序的其他部分。也就是说，降低大小界可能使我们能够增大那些看起来更为重要的其他界限。我们将能够在保持验证时间稳定且不牺牲保证的情况下实现这一点。

参考文献

[1] Mohammad Abdulaziz、Michael Norrish 和 Charles Gretton。2018年。经形式化验证的算法：为状态空间直径求上界。《自动推理杂志》61（2018年），485–520。

[2] Benjamin Aminof、Swen Jacobs、Ayrat Khalimov 和 Sasha Rubin。2014年。令牌传递系统的参数化模型检测。载于《验证、模型检测与抽象解释——第15届国际会议，VMCAI 2014》，美国加利福尼亚州圣迭戈，2014年1月19日至21日，会议论文集（《计算机科学讲义》，第8318卷），Kenneth L. McMillan 和 Xavier Rival（编）。施普林格，262–281。https://doi.org/ 10 1007 978 3 642 54013 4_15 . / - - - - Irina Mariuca Asavoae[3]、Mihail Asavoae 和 Adrián Riesco。2018年。从形式语义进行切片：Chisel——一种通用程序切片工具。《国际软件工具与技术转移杂志》20，6（2018年），739–769。https://doi.org/10.1007/s10009-018-0500-y [4] Mohammad Awedh 和 F. Somenzi。2004年。通过有界模型检测证明更多性质。载于 CAV。[5] Dirk Beyer 和 Matthias Dangl。2020年。使用 PDR 进行软件验证：前沿技术的实现。载于《系统构建与分析工具和算法——第26届国际会议，TACAS 2020》，作为欧洲软件理论与实践联合会议 ETAPS 2020 的一部分举行，爱尔兰都柏林，2020年4月25日至30日，会议论文集，第一部分（《计算机科学讲义》，第12078卷），Armin Biere 和 David Parker（编）。施普林格，3–21。https://doi.org/ 10 1007 978 3 030 45190 5_1 . / - - - - Dirk Beyer、Nian-Ze Lee[6] 和 Philipp Wendler。2022年。重新审视插值与基于SAT的模型检测：将其应用于软件验证。CoRR abs/2208.05046（2022年）。https://doi.org/ 10 48550 2208 05046 2208 05046 . /arXiv. . arXiv: . [7] Armin Biere、Alessandro Cimatti、Edmund M. Clarke 和 Yunshan Zhu。1999年。无需 BDD 的符号模型检测。载于《系统构建与分析工具和算法国际会议》。[8] Per Bjesse 和 Koen Claessen。2000年。无需遍历状态空间的基于SAT验证。载于《计算机辅助设计形式化方法，第三届国际会议，FMCAD 2000》，美国得克萨斯州奥斯汀，2000年11月1日至3日，会议论文集（《计算机科学讲义》，第1954卷），Warren A. Hunt Jr. 和 Steven D. Johnson（编）。施普林格，372–389。https://doi.org/10.1007/3-540-40922-X_23[9] Marius Bozga、Peter Habermehl、Radu Iosif、Filip Konecný 和 Tomás Vojnar。2009年。整数数组程序的自动验证。载于《计算机辅助验证，第21届国际会议，CAV 2009》，法国格勒诺布尔，2009年6月26日至7月2日，会议论文集（《计算机科学讲义》，第5643卷），Ahmed Bouajjani 和 Oded Maler（编）。施普林格， 157 172 10 1007 978 3 642 02658 4_15 – 。https://doi.org/[10] 。/ - - - - Aaron R. Bradley。2011。。无需展开的基于SAT的模型检测。载于《验证、模型检测与抽象解释——第12届国际会议，VMCAI 2011》，美国得克萨斯州奥斯汀，2011年1月23日至25日，会议论文集（《计算机科学讲义》，第6538卷），Ranjit Jhala 和 David A. Schmidt（编）。施普林格，70–87。https://doi.org/ 10 1007 978 3 642 18275 4_7 . / - - - - Aaron R. Bradley、Zohar Manna[11] 和 Henny B. Sipma。2006年。关于数组，哪些问题是可判定的？载于《验证、模型检测与抽象解释，第7届国际会议，VMCAI 2006》，美国南卡罗来纳州查尔斯顿，2006年1月8日至10日，会议论文集（《计算机科学讲义》，第3855卷），E. Allen Emerson 和 Kedar S. Namjoshi（编）。施普林格， 427 442 10 1007 11609773_28 – 。https://doi.org/[12] 。/ Michael C. Browne、Edmund M. Clarke 和 Orna Grumberg。1989年。关于具有许多相同有限状态进程的网络进行推理。《信息与计算》81，1（1989年），13–31。https://doi.org/ 10 1016 0890 5401 89 90026 6 . / - ( ) - Daniel Bundala、Joël Ouaknine[13] 和 James Worrell。2012年。关于有界模型检测中完备性阈值的大小。第27届 IEEE 逻辑与计算机科学年度研讨会（2012年），155–164。[14] Supratik Chakraborty、Ashutosh Gupta 和 Divyesh Unadkat。2020年。通过全程序归纳验证数组操作程序。载于《系统构建与分析工具和算法——第26届国际会议，TACAS 2020》，作为欧洲软件理论与实践联合会议 ETAPS 2020 的一部分举行，都柏林，爱尔兰，2020年4月25日至30日，会议论文集，第一部分（计算机科学讲义，第12078卷），Armin Biere 和 David Parker（编）。施普林格，22–39。https://doi.org/10.1007/978-3-030-45190-5_2[15] Nathan Chong、Byron Cook、Konstantinos Kallas、Kareem Khazem、Felipe R. Monteiro、Daniel Schwartz-Narbonne、Serdar Tasiran、Michael Tautschnig 和 Mark R. Tuttle。2020。软件开发工作流中的代码级模型检测。载于 ACM/IEEE 第42届软件工程国际会议：软件工程实践会议论文集（韩国首尔）（ICSE-SEIP '20）。计算机协会，纽约，美国，11–20。https://doi.org/ 10 1145 3377813 3381347 。/ 。Edmund Clarke、Orna Grumberg[16]、Somesh Jha、Yuan Lu 和 Hel- mut Veith。2000。反例引导的抽象精化。载于计算机辅助验证，E. Allen Emerson 和 Aravinda Prasad Sistla（编）。施普林格柏林海德堡，柏林，海德堡，154–169。[17] Edmund M. Clarke。2008。模型检测——我为克服状态爆炸问题而奋斗的27年。载于《逻辑编程、人工智能与推理》，Iliano Cervesato、Helmut Veith 和 Andrei Voronkov（编）。施普林格柏林海德堡，柏林，海德堡，182–182。[18] Edmund M. Clarke、Orna Grumberg、Daniel Kroening、Doron A. Peled 和 Helmut Veith。2018。模型检测，第2版。麻省理工学院出版社。https://mitpress.mit.edu/books/model-checking-second-edition [19] Edmund M. Clarke、Daniel Kroening 和 Flavio Lerda。2004。用于检查 ANSI-C程序的工具。载于《系统构建与分析工具和算法》，第10届国际会议 TACAS 2004，作为欧洲软件理论与实践联合会议 ETAPS 2004 的一部分举办，西班牙巴塞罗那，2004年3月29日至4月2日，会议论文集（计算机科学讲义，第2988卷），Kurt Jensen 和 Andreas Podelski（编）。施普林格，168–176。https://doi.org/ 10 1007 978 3 540 24730 2_15 。/ - - - - Edmund M. Clarke 、[20] Daniel Kroening 、 Joël Ouaknine 和 Ofer Strichman。2004。有界模型检测的完备性与复杂度。载于验证、模型检测与抽象解释国际会议。[21] Edmund M. Clarke、Daniel Kroening 和 Karen Yorav。2003。利用有界模型检测验证 C 与 Verilog 程序的行为一致性。载于第40届设计自动化会议（DAC 2003）会议论文集，美国加利福尼亚州阿纳海姆，2003年6月2日至6日。ACM，368–371。https://doi.org/ 10 1145 775832 775928 。/ 。Edmund M. Clarke、Muralidhar Talupur[22]、Tayssir Touili 和 Helmut Veith。2004。通过网络分解进行验证。载于 CONCUR 2004——并发理论，第15届国际会议，英国伦敦，2004年8月31日至9月3日，会议论文集（计算机科学讲义，第3170卷），Philippa Gardner 和 Nobuko Yoshida（编）。施普林格， 276 291 10 1007 978 3 540 28644 8_18 – 。https://doi.org/[23] 。/ - - - - MITRE公司。2006。。CWE-[24]：193 差一错误。https://cwe.mitre.org/data/definitions/193.html [24] Edsger W. Dijkstra。1976。程序设计的学科。Pentice Hall。[25] Alastair F. Donaldson、Leopold Haller、Daniel Kroening 和 Philipp Rümmer。2011。使用 k归纳进行软件验证。载于《静态分析——第18届国际研讨会，SAS 2011》，意大利威尼斯，2011年9月14日至16日。会议论文集（计算机科学讲义，第6887卷），Eran Yahav（编）。施普林格，351–368。https://doi.org/ 10 1007 978 3 642 23702 7_26 。/ - - - - Vijay Victor D' Silva、Daniel Kroening[26] 和 Georg Weissenbacher。2008。软件形式化验证自动化技术综述。《IEEE计算机辅助设计与集成电路系统汇刊》27，7（ 2008 1165 1178 10 1109 2008 923410 ），– 。https://doi.org/[27] 。/TCAD。2011。。Niklas Eén、Alan Mishchenko 和 Robert K. Brayton。。属性导向可达性的高效实现。载于计算机辅助设计形式化方法国际会议 FMCAD '11，02，2011，30 125 134，美国得克萨斯州奥斯汀，10月至11月，Per Bjesse 和 Anna Slobodová（编）。FMCAD公司，– 。

http://dl.acm.org/citation.cfm?id=2157675

[28] E. Allen Emerson 和 Kedar S. Namjoshi。1995。关于环的推理。载于 POPL'95 会议记录：第22届 ACM SIGPLAN-SIGACT 编程语言原理研讨会，美国加利福尼亚州旧金山，1995年1月23日至25日，Ron K. Cytron 和 Peter Lee（编）。ACM出版社，85 94 10 1145 199448 199468 – 。

[29] https://doi.org/ 。/ 。Cormac Flanagan 和 James B. Saxe。2001。避免指数级膨胀：生成紧凑的验证条件。载于 ACM-SIGACT 编程语言原理研讨会。[30] Keijo Heljanko、Tommi A. Junttila 和 Timo Latvala。2005。针对完整 PLTL 的增量式完备有界模型检测。载于计算机辅助验证国际会议。[31] C. A. R. Hoare。1968。计算机程序设计的公理基础。《ACM通讯》12（1968），576–580。https://doi.org/ 10 1145 363235 363259 。/ 。

[32] Ranjit Jhala 和 Kenneth L. McMillan。2007。基于证明的数组抽象。载于计算机辅助验证，第19届国际会议 CAV 2007，7，2007，3 4590 德国柏林，7月至会议论文集（计算机科学讲义，第 卷），Werner Damm 和 Holger Hermanns（编）。施普林格，193–206。https://doi.org/ 10 1007 978 3 540 73368 3_23 。/ - - - - Ralf Jung、

[33] Robbert Krebbers、Jacques-Henri Jourdan、Ales Bizjak、Lars Birkedal 和 Derek Dreyer。2018。Iris 从基础构建：高阶并发分离逻辑的模块化基础。《函数式编程杂志》28（2018），e20。https://doi.org/10.1017/S0956796818000151[34] Daniel Kroening、Joël Ouaknine、Ofer Strichman、Thomas Wahl 和 James Worrell。2011。用于有界模型检测的线性完备性阈值。载于计算机辅助验证——第23届国际会议 CAV 2011，美国犹他州 Snowbird，2011年7月14日至20日。会议论文集（计算机科学讲义，第6806卷），Ganesh Gopalakrishnan 和 Shaz Qadeer（编）。施普林格，557–572。https://doi.org/ 10 1007 978 3 642 22110 1_44 。/ - - - - Daniel Kroening 和

[35] Ofer Strichman。2003。递归直径的高效计算。载于验证、模型检测与抽象解释国际会议。[36] Isabella Mastroeni 和 Damiano Zanardini。2008。数据依赖与程序切片：从语法到抽象语义。载于2008年 ACM SIGPLAN 偏求值研讨会会议论文集和基于语义的程序操作，PEPM 2008，旧金山，加利福尼亚州，美国，2008年1月7日至8日，Robert Glück 和 Oege de Moor（编）。ACM，125 134 10 1145 1328408 1328428 – 。https://doi.org/ 。/ 。

[37] Umang Mathur、Adithya Murali、Paul Krogmeier、P. Madhusudan 和 Mahesh Viswanathan。2020年。判定单遍堆操作程序的内存安全性。ACM程序设计语言会议录 4，POPL（2020 35 1 35 29 10 1145 3371103），：–：。https://doi.org/ 。/ Kenneth L. McMillan。2003。插值与基于SAT的模型检测。载于计算机辅助验证国际会议。[39] Louise E. Moser。1990年。Ada程序的数据依赖图。IEEE软件工程汇刊 16，5（1990），498–509。https://doi.org/ 10 1109 32 52773 。/ 。Peter W. O'Hearn。2019。分离逻辑。

[40] ACM通讯 62，2（2019），86 95 10 1145 3211968 – 。

[41] https://doi.org/ 。/ Peter W. O'Hearn、John C. Reynolds 和 Hongseok Yang。2001年。关于修改数据结构的程序的局部推理。载于CSL。https://doi.org/ 10 1007 3 540 44802 0_1 。/ - - - D.Y.W. Park、U. Stern、J.U.

[42] Skakkebaek 和 D.L. Dill。2000年。Java模型检测。载于ASE 2000会议论文集。第十五届IEEE自动化软件工程国际会议。253–256。https://doi.org/ 10 1109 2000 873671 。/ASE。 。Gaurav Parthasarathy、

[43] Peter Müller 和 Alexander J. Summers。2021年。实践验证条件生成器的形式化验证。载于计算机辅助验证国际会议。[44] Tobias Reinhard。2023年。数组遍历程序内存安全的完备性阈值：早期技术报告。arXiv: 2211 11885 [ ] 。cs.LO John C. Reynolds。2002年。分离逻辑：一

[45] 种用于共享可变数据结构的逻辑。第十七届IEEE计算机科学逻辑年度研讨会论文集（2002年），55–74。https://doi.org/ 10 1109 2002 1029817 。/LICS。 。Mary Sheeran、

[46] Satnam Singh 和 Gunnar Stålmarck。2000年。使用归纳和SAT求解器检查安全性质。载于计算机辅助设计形式化方法，第三届国际会议，FMCAD 2000，美国得克萨斯州奥斯汀，2000年11月1日至3日，会议论文集（计算机科学讲义，第1954卷），Warren A. Hunt Jr. 和 Steven D. Johnson（编）。施普林格，108–125。https://doi.org/ 10 1007 3 540 40922 _8 。/ - - -X Mark

[47] D. Weiser。1984。程序切片。IEEE软件工程汇刊 10，4 1984 352 357 10 1109 1984 5010248（），–。https://doi.org/ 。/TSE。 。