清 华 大 学

综 合 论 文 训 练

题目：IC3+IA 模型检测框架中语法制导抽象的设计与实现

系　　别：计算机科学与技术系

专　　业：计算机科学与技术

姓　　名：杨　淇

指导教师：王生原　副教授

2021 年 6 月 16 日

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：学校有权保留学位论文的复印件，允许该论文被查阅和借阅；学校可以公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存该论文。

(涉密的学位论文在解密后应遵守此规定)

签　名：杨 淇　导师签名：王生原　日　期 2021.6.15

中文摘要

在模型检测领域中，状态空间抽象是一种通用的降低模型状态空间复杂度的方法。IC3+IA 模型检测算法，在已获广泛应用的 IC3 算法的基础上，引入了隐式抽象技术和谓词抽象方法，让 IC3 得以被应用于无限状态系统。语法制导抽象，是不同于谓词抽象的一种新的抽象方法，其核心思想在于从模型检测问题的语法中抽取子谓词和子项，并依据子谓词的真值以及子项之间的相等关系形成抽象状态空间。本文探讨将语法制导抽象应用于 IC3+IA 的可能思路，并在已有的一套支持多种抽象方法的 IC3+IA 框架中对语法制导抽象进行了实现。实验结果表明，当前对语法制导抽象的实现的性能整体上比谓词抽象更低，但仍然存在一些问题场景，当前实现能够在其中取得比谓词抽象更好的结果。

关键词：模型检测；IC3；抽象

ABSTRACT

In the field of model checking, state space abstraction is a general approach to reduce model state space complexity. Based on the widely used IC3 model checking algorithm, IC3+IA introduces implicit abstraction technique and predicate abstraction, enabling IC3 to be applied to infinite-state systems. Syntax-guided abstraction, a new abstraction approach different from predicate abstraction, has the core idea of extracting sub-predicates and sub-terms from the syntax of the model checking problem, then forming abstract state space according to truth values of these sub-predicates along with equality relations between these sub-terms. In this paper, we discuss possible ideas to apply syntax-guided abstraction to IC3+IA, and implement syntax-guided abstraction within an existing IC3+IA framework which supports multiple kinds of abstraction approaches. Experiments show that our current implementation has an overall lower performance compared with predicate abstraction, but there still exist some problem scenarios in which our implementation performs better.

Keywords: Model Checking; IC3; Abstraction

目　录

主要符号对照表

X	变量的集合
$I(X)$	初始状态公式
$T(X, X')$	状态转移公式
$P(X)$	代表待验证属性的公式

第 1 章　引言

1.1　背景

长期以来，缺陷（bug）是软件和硬件开发中的一个非常常见的问题。缺陷通常源于开发人员在设计系统时的疏忽——大到对整个系统应有的运行逻辑没有考虑周全，比如忽视了某些异常输入出现的可能性，小到数组越界访问，甚至仅仅是一处对变量的使用的变量名笔误，都有可能成为缺陷的来源。缺陷的后果可能相对较小，比如仅仅对用户体验造成少许影响，但也有可能是灾难性的——于 2018 年被披露的熔断（Meltdown）漏洞[1]和幽灵（Spectre）漏洞[2]，就是近几年造成重大信息安全问题的硬件漏洞的两个知名例子。除了缺陷后果本身，这些造成重大影响的缺陷的严重性还有可能体现在以下两个方面：

- 难以被发现，缺陷可能已经存在了很长时间并持续造成潜在影响。以于 2014 年被披露的心脏出血（Heartbleed）漏洞[3]为例，就有一些迹象表明，攻击者对于该漏洞的利用，最晚从该漏洞被公开披露的五个月前就开始了。
- 即使被发现，也难以被完全妥善地修复。以 WannaCry 勒索病毒所利用的软件漏洞为例，尽管该漏洞在 2017 年 3 月就已经在较新的 Windows 操作系统版本被修复，但在 2017 年 5 月病毒爆发时，还是有很多 Windows 用户因为未安装修复补丁而受害[4]。硬件漏洞的情况则更为严重，与向用户推送更新以修复软件漏洞的可能性相比，修复用户手中的硬件的漏洞更为麻烦，甚至可能根本不可行。

为了减小缺陷所带来的负面影响，人们提出了许多方法，测试，就是其中常见的一种，用以应对那些与系统行为正确性有关的缺陷。在测试过程中，开发人员设计出若干测试样例，分别以这些样例作为所要测试的软件或硬件系统的输入或环境，考察系统运行时的行为是否符合预期。如果在测试过程中没有发现问题，那么可以认为，系统在给定的若干样例下的行为是正确的。测试所能确保的这样的正确性往往是不完全的——对于绝大多数系统，可能的输入或环境的数量是较巨大甚至无限的，为了证明系统在所有可能的输入或环境下都正确，而对每一种可能性都进行测试，往往是完全不现实的。在多数场景下，综合考虑开发成本等因素，这样的不完全的正确性其实已经足够。但也存在一些系统，尤其在医疗、航天等领域，它们对完全的正确性是有所要求的。为了确保这样的完全的正确性，一种思

路是用数学（形式化）的语言去对系统以及预期的行为进行建模。这样，对系统行为是否符合预期的判断，就转化为了可以通过形式化的过程去证明或证否的问题。

1.1.1 模型检测

与上述思路相对应的形式化方法有很多，模型检测（model checking）[5]就是其中之一。在模型检测中，开发者通常需要将待验证的系统和预期的行为建模成一个变迁系统（transition system）和与之相对应的希望被证明为正确的属性（properties）。一般地，一个变迁系统由一个状态集合，以及该集合的一个子集作为初始状态集合，以及一个状态转移关系组成。模型检测问题的具体表示形式有多种，这些形式的表达能力也各有差异，本文只涉及以下的基于一阶逻辑（一阶理论）的形式，该形式包含以下几个要素：

- 变量的集合，记为 X。每个变量都有相应的类型，可以是布尔，也可以是整数、4 位位向量，等等。一个状态，就是一组对每个变量的赋值。
- 初始状态公式，记为 $I(X)$。我们使用形如 $\varphi(\cdots)$ 的形式，来表明 $I(X)$ 中所出现的所有自由变量都来自 X。
- 状态转移公式，记为 $T(X, X')$。X' 代表给 X 中的每个变量都加上撇号所得到的集合。
- 代表待验证属性的公式，记为 $P(X)$。

一条路径，是一个状态序列 $s_1, \cdots, s_n$，其中 $n \geqslant 1$，满足 $s_1 \vDash I(X)$，且对于任意的 $1 \leqslant i \leqslant n-1$，有 $(s_i, s'_{i+1}) \vDash T(X, X')$。一条反例路径（counterexample path），是指最后一个状态不满足 $P(X)$ 的路径。该模型检测问题是安全的，当且仅当反例路径不存在。

对上述形式，我们举一个例子以便理解：

例 1.1： 令

$$X = \{x, y\},\ x : \mathbb{Z},\ y : \mathbb{Z},$$
$$I(X) : (x = 1) \wedge (y = 1),$$
$$T(X, X') : (x' = 2x - y) \wedge (y' = y),$$
$$P(X) : x = 1.$$

该问题的可能路径只能是类似这样的形式：$s_1 : x = 1, y = 1,\ s_2 : x = 1, y = 1,\ s_3 : x = 1, y = 1, \cdots$，因此该问题是安全的。但如果我们把 $I(X)$ 中的 $y = 1$ 改为 $0 \leqslant y \leqslant 1$，该问题就变为不安全了，因为存在反例 $s_1 : x = 1, y = 0,\ s_2 : x = 2, y = 0$。 ∎

1.1.2 状态空间抽象

随着模型检测所要解决的实际问题变得越来越复杂（例如将单线程问题扩展成并行问题），模型的状态变量也变得越来越多，进而导致模型的状态数（状态空间大小）以指数速度增长，给模型检测算法带来很大负担——这样的问题被称为状态空间爆炸。目前已经有多种方法致力于降低状态数，状态空间抽象（state space abstraction）被认为是其中最通用的一种[6]。

一般地，状态空间抽象会显式或隐式地定义一个抽象状态空间（这里记为 S_{abs}）和一个将具体状态（即原问题状态空间中的状态）映射到抽象状态的抽象函数（这里记为 h）。状态空间抽象会略去原模型在行为上的一些细节，存在型抽象（existential abstraction）[7]和全称抽象（universal abstraction）[8]则是两种简化细节的方式。给定原模型检测问题 $\langle X, I(X), T(X, X'), P(X)\rangle$、抽象状态空间 S_{abs} 和抽象函数 h，分别基于存在型抽象和全称抽象，我们可以这样定义新的抽象模型检测问题：

存在型抽象

$$\hat{I}(\hat{s}) : \exists X, (h(X) = \hat{s}) \wedge I(X), \tag{1.1}$$

$$\hat{T}(\hat{s}, \hat{s}') : \exists X, X', (h(X) = \hat{s}) \wedge (h(X') = \hat{s}') \wedge T(X, X'), \tag{1.2}$$

$$\hat{P}(\hat{s}) : \forall X, (h(X) = \hat{s}) \rightarrow P(X). \tag{1.3}$$

全称抽象

$$\hat{I}(\hat{s}) : \forall X, (h(X) = \hat{s}) \rightarrow I(X), \tag{1.4}$$

$$\hat{T}(\hat{s}, \hat{s}') : \forall X, X', (h(X) = \hat{s}) \wedge (h(X') = \hat{s}') \rightarrow T(X, X'), \tag{1.5}$$

$$\hat{P}(\hat{s}) : \exists X, (h(X) = \hat{s}) \wedge P(X). \tag{1.6}$$

可以说，存在型抽象所得模型的行为是原模型行为的超集，而全称抽象所得模型的行为是原模型行为的子集。对于存在型抽象，如果存在型抽象所得问题是安全的，那么原问题也是安全的；对于全称抽象，如果全称抽象所得问题是不安全的，那么原问题也是不安全的。

后续我们只考虑存在型抽象。在本小节的最后，我们举一个存在型抽象的例子。

例 1.2： 令

$$X = \{x, y\},\ x : \mathbb{Z},\ y : \mathbb{Z},$$

$$I(X) : (x = 1) \wedge (y = 2),$$

$$T(X, X') : (x' = y - x) \wedge (y' = x^2 + y^2),$$
$$P(X) : (x + y) \bmod 3 = 0.$$

为证明该问题是安全的，我们可以定义这样的存在型抽象问题：

$$\hat{X} = \{\hat{x}, \hat{y}\}, \hat{x} : \{0, 1, 2\},\ \hat{y} : \{0, 1, 2\},\ \hat{x} = x \bmod 3,\ \hat{y} = y \bmod 3,$$
$$\hat{I}(\hat{X}) : (\hat{x} = 1) \wedge (\hat{y} = 2),$$
$$\hat{T}(\hat{X}, \hat{X}') : ((\hat{x} = 1) \wedge (\hat{y} = 2) \rightarrow (\hat{x}' = 1) \wedge (\hat{y}' = 2)) \wedge \cdots,$$
$$\hat{P}(\hat{X}) : (\hat{x} + \hat{y}) \bmod 3 = 0.$$

该抽象问题是安全的，因此原问题也是安全的。 ∎

1.1.3 一阶逻辑术语

在背景部分的最后，补充介绍几个一阶逻辑的术语[9]，以便后续讨论。

原子（atom） ⊤（true）、⊥（false），或将某 n 个项（term）代入到某个谓词（predicate）所得到的公式，如 $x = 1$。

文字（literal） 一个原子，或一个原子的非。

立方体（cube） 零至多个文字的合取（conjunction）。

子句（clause） 零至多个文字的析取（disjunction）。另一个等价的定义是，一个立方体的非。

1.2 本文贡献

本文的主要贡献在于：

- 讨论了一些将语法制导抽象[10]（见 2.4 节）应用于 IC3+IA 模型检测算法[11]（见 2.3 节）所要注意的理论问题；
- 提出了 IC3+IA 的一种对语法制导抽象更加友好的等价形式；
- 提出了将语法制导抽象应用于 IC3+IA 的多种可能方案；
- 在已有的一套支持多种抽象的 IC3+IA 框架中，对这些可能方案进行了代码实现；
- 对目前实际可用的两种方案进行了实验，说明了两种方案及其实现的正确性，并比较了二者连同谓词抽象共三者之间的性能差异。

1.3 本文结构

第一章，介绍本文背景，包括模型检测和状态空间抽象，以及本文贡献和本文结构。

第二章，介绍本文的相关工作和理论基础，包括 IC3+IA 和语法制导抽象，以及二者所基于的更基础的工作和理论。

第三章，对 IC3+IA 和语法制导抽象展开讨论，以形成将语法制导抽象应用于 IC3+IA 的理论依据。

第四章，提出将语法制导抽象应用于 IC3+IA 的多种可能方案（统称为 IC3+IA+SA），介绍当前针对这些方案所进行的代码实现，并讨论这些方案目前的实际可用性。

第五章，对目前实际可用的两种方案进行了实验，以评估当前实现的正确性和性能。

第六章，对本文的工作进行总结，并提出后续工作的可能方向。

第 2 章　相关工作

2.1　CEGAR

前文提到，存在型抽象给我们提供了一条能够帮助我们验证原模型检测问题是否安全的性质：“如果存在型抽象所得问题是安全的，那么原问题也是安全的。”该命题的否命题不一定成立，这是因为存在型抽象会使模型行为增多，对于抽象问题中的一条（抽象）反例路径，在原问题中与之对应的（具体）反例路径可能存在，也可能不存在。如果是后者，这条抽象反例路径就被称为伪反例（spurious counterexample）。将例 1.2 中的抽象状态空间更换为由零个抽象变量形成，即将所有具体状态都映射到同一抽象状态，就可以得到一个抽象问题不安全而原问题安全的例子（存在一条长度为 1 的伪反例）。

遇到伪反例后，一种自然的思路是，构造更精细的抽象，并希望伪反例能够在新抽象中消失。原抽象和新抽象可以人工构造，但在实际场景中，我们更希望让抽象的构造成为一个自动化的流程。反例制导的抽象精化（counterexample-guided abstraction refinement，CEGAR）[6]就提出了这样一个自动化的流程框架，它主要由以下几个步骤组成：

1. 依据原问题，自动地构造出一个较粗糙的初始抽象。
2. 对当前抽象问题执行模型检测算法，如果抽象问题是安全的，那么原问题也是安全的。
3. 否则，检查所遇到的抽象反例是否是伪反例，如果不是，则原问题是不安全的。
4. 否则，依据该伪反例的有关信息和当前抽象，自动地构造出一个更精细的新抽象，以消除该伪反例，然后回到第二步并继续。生成新抽象的过程被称为精化（refinement）。

本文即将讨论的所有涉及抽象的算法，都基于 CEGAR 流程。

2.2　IC3

接下来我们讨论具体的模型检测算法。IC3（**I**ncremental **C**onstruction of **I**nductive **C**lauses for **I**ndubitable **C**orrectness）[12]，是一类基于不变式属性的高效的模型检测算法，在硬件模型检测领域得到了广泛的应用，其一开始是针对由有限个

布尔变量所形成的状态空间的，后来则被扩展到了基于一阶理论的情形[13]。另外，Niklas Een 等人给 IC3 起了另一个名字，称为 PDR（property directed reachability），并针对原始 IC3 算法提出了若干改进[14]。本节将大致介绍改进后的算法。

给定模型检测问题 $\langle X, I(X), T(X, X'), P(X)\rangle$，IC3 的根本目标是，找到一个能够表明原问题为安全的归纳不变式（inductive invariant），或是找到一条反例路径。归纳不变式，是满足以下三条性质的公式 $F(X)$：

1. $I(X) \to F(X)$；
2. $F(X) \wedge T(X, X') \to F(X')$；
3. $F(X) \to P(X)$。

其中，$F(X')$ 代表给 $F(X)$ 中的每一处来自 X 的自由变量都加上撇号所得到的公式。

为达到这样的目标，IC3 维护一个公式（这里也称为帧（frame））的序列 $frames: F_1(X), F_2(X), \cdots, F_n(X)$，其中 $n \geqslant 1$，满足以下四条性质：

1. $F_1(X) \leftrightarrow I(X)$；
2. 对于任意的 $1 \leqslant i \leqslant n-1$，$F_i(X) \to F_{i+1}(X)$；
3. 对于任意的 $1 \leqslant i \leqslant n-1$，$F_i(X) \wedge T(X, X') \to F_{i+1}(X')$；
4. 对于任意的 $1 \leqslant i \leqslant n-1$（即除了 $i = n$），$F_i(X) \to P(X)$。

从这四条性质中，我们可以看出：

- 每个 $F_i(X)$ 都包含了从初始状态集合出发，走不超过 $i-1$ 步（含零步）所能到达的所有状态。
- 如果存在 $1 \leqslant k \leqslant n-1$，使得 $F_k(X) \leftrightarrow F_{k+1}(X)$，那么 $F_k(X)$ 是一个归纳不变式。

初始，令 $frames: I(X)$。另外，在 IC3 的整个流程中，我们让每个帧都以合取范式（CNF）的形式表示。IC3 接下来将交替进行阻塞阶段（blocking phase）和传播阶段（propagation phase），直到找到归纳不变式或反例路径。

2.2.1 阻塞阶段

为方便理解，我们首先定义该阶段所要进行的基本操作：对于任意的公式 $c(X)$ 和 $1 \leqslant k \leqslant n-1$，如果 $I(X) \to c(X)$，且 $F_k(X) \wedge c(X) \wedge T(X, X') \to c(X')$，那么对于任意的 $2 \leqslant i \leqslant k+1$，令 $F_i(X) := F_i(X) \wedge c(X)$。可以证明，完成该操作后，帧序列的四条性质仍满足。每次完成该操作后，帧序列对不超过 $i-1$ 步可达状态集合的近似都将变得更精细。

在每次进行该阶段的起始，我们首先检查 $F_n(X) \wedge \neg P(X)$ 是否可满足（satisfiable）。若不可满足，则当前阶段结束，进入传播阶段。否则，取出一个令该公式可满足的状态 s（$s \vDash F_n(X)$，且 $s \vDash \neg P(X)$），并生成与该状态相对应的一个（基本）立方体 $c(X)$。如果 $n = 1$，则反例存在，IC3 算法结束。否则，我们希望从 $F_n(X)$ 中“切掉”这个立方体，即令 $F_n(X) := F_n(X) \wedge \neg c(X)$。我们考虑上述基本操作：可以证明，$I(X) \rightarrow \neg c(X)$；剩下来需要我们检查的是，$F_{n-1}(X) \wedge \neg c(X) \wedge T(X, X') \rightarrow \neg c(X')$ 是否成立，或等价地，$F_{n-1}(X) \wedge \neg c(X) \wedge T(X, X') \wedge c(X')$ 是否可满足。若不可满足，则操作可以进行，然后我们需要重新检查 $F_n(X) \wedge \neg P(X)$ 的可满足性。否则，根据被检查的公式，在 $F_{n-1}(X)$ 中必然存在一个状态 s_{pre}，其走一步可以到达 s，我们进而需要对 s_{pre} 考虑与 s 类似的事情。

这是一个递归和循环的过程，出于效率考虑，我们使用优先级队列实现。我们维护一个三元组的小顶堆 *obligations*，每个三元组都包含一个立方体和一个帧号，以及一个可为空的指向另一个三元组（父三元组）的指针。该三元组也称为证明义务（proof obligation），代表我们希望从该帧中切掉该立方体。证明义务在小顶堆中以帧号为排序依据。算法 2.1 给出了阻塞阶段的伪代码流程。

算法 2.1 IC3，阻塞阶段

```
1:  while there exists state s, s.t. s ⊨ F_n(X) ∧ ¬P(X) do
2:      obligations := Heap({(Cube(s), n, Null)})
3:      while obligations is not empty do
4:          (c(X), k, parent) := obligations.top()
5:          if k = 1 then
6:              counterexample := [GetOneState(c(X))]
7:              while parent ≠ Null do
8:                  (c(X), k, parent) := *parent
9:                  counterexample.push_back(GetOneState(c(X)))
10:             end while
11:             return counterexample
12:         else if F_{k-1}(X) ∧ ¬c(X) ∧ T(X, X') ∧ c(X') is satisfiable then
13:             get a state s_pre in F_{k-1}(X) ∧ ¬c(X) from satisfiable solution
14:             obligations.push((Cube(s_pre), k − 1, pointer to current top))
15:         else
16:             for i = 2, ··· , k do
17:                 F_i(X) := F_i(X) ∧ ¬c(X)
18:             end for
19:             obligations.pop()
20:         end if
21:     end while
22: end while
```

2.2.1.1 泛化

算法 2.1 的主要问题在于，每次切掉的立方体都只有单个状态这么小，相比单个帧来说可以说微乎其微，这会让算法变得很低效。对此，论文[14]提出了几种被统称为泛化（generalization）的改进方法：

1. 在算法第 16 行开始执行前，$I(X) \to \neg c(X)$ 是有效的，且 $F_{k-1}(X) \wedge \neg c(X) \wedge T(X, X') \wedge c(X')$ 是不可满足的。在此我们可以考虑设法缩短 $c(X)$ 的长度，即移除掉 $c(X)$ 中的若干文字，同时确保阻塞基本操作所要求的两个条件仍成立，这样，我们在第 16 行至第 18 行，就可以切掉更大的立方体了。至于具体如何缩短 $c(X)$ 的长度，我们可以采用启发式方法，参见[15]。
2. 在算法第 13 行，如果此时 $k \geqslant 3$，则我们可以考虑用一个更大的立方体（记为 $c_{pre}(X)$）来代替 Cube(s_{pre})，只要 $c_{pre}(X)$ 满足 Cube(s_{pre}) $\to c_{pre}(X)$，且 $c_{pre}(X)$ 中的任一状态都可以走一步到达 $c(X)$。后一个条件在语法制导抽象原文[10]中被称为严格连续性（strict continuity），该条件的其中一个目的是确保在第 5 行后，反例路径仍然有办法通过证明义务链条构造出来。论文[14]提出，对于状态空间由有限个布尔变量所形成的情形，我们可以采用三值模拟（ternary simulation）方法来构造 $c_{pre}(X)$。
3. 类似于 2.，在算法第 2 行，如果 $n \geqslant 2$，则我们可以考虑用一个更大的立方体（记为 $c_{bad}(X)$）来代替 Cube(s)，只要 $c_{bad}(X)$ 满足 Cube(s) $\to c_{bad}(X)$，且 $c_{bad}(X) \to \neg P(X)$。

2.2.2 传播阶段

一轮阻塞阶段结束后，如果阻塞阶段没有返回反例，那么由循环条件，我们知道此时 $F_n(X) \to P(X)$ 成立。我们接下来进行传播阶段，其包含两个步骤：

1. 将 $\top$ 添加到帧序列末尾。由于在此之前我们有 $F_n(X) \to P(X)$，在进行添加操作后，帧序列的四条性质仍满足。
2. 对于任意的 $1 \leqslant k \leqslant n-1$，和 $F_k(X)$ 的 CNF 表示中的任一子句 $c(X)$，如果 $F_k(X) \wedge T(X, X') \to c(X')$ 成立，则令 $F_{k+1}(X) := F_{k+1}(X) \wedge c(X)$。
3. 如果存在 $1 \leqslant k \leqslant n-1$，使得 $F_k(X) = F_{k+1}(X)$，那么 $F_k(X)$ 是一个归纳不变式，模型检测问题是安全的，IC3 算法结束。否则，开始新的一轮阻塞阶段。

传播阶段相当于一个“继承”子句的过程，同样能够让帧序列对可达状态集合的近似变得更精细。

2.3 IC3+IA

IC3+IA（IC3 modulo theories via implicit (predicate) abstraction）[11]模拟了 IC3 在谓词抽象状态空间上的运行方式，并使用了隐式抽象技术来避免显式计算抽象状态转移公式（$\hat{T}(\hat{s}, \hat{s}')$），进而使得其相比当时已有的其他 IC3 抽象算法，能够适用于更广范围的背景理论。本节我们首先介绍 IC3+IA 所基于的谓词抽象和隐式抽象。

2.3.1 谓词抽象

谓词抽象（predicate abstraction）[16] 使用若干描述模型状态的布尔公式来形成抽象状态空间。给定谓词（公式）集合 $\mathbb{P} = \{P_1(X), P_2(X), \cdots, P_n(X)\}$，其中 $n \geqslant 0$，我们定义 n 个布尔抽象变量 $\hat{X} = \{\hat{x}_1, \hat{x}_2, \cdots, \hat{x}_n\}$，并定义相应的抽象函数 $h(X)$，使得 $h(X) = (\hat{x}_1, \hat{x}_2, \cdots, \hat{x}_n)$，当且仅当对于任意的 $1 \leqslant i \leqslant n$，$\hat{x}_i \leftrightarrow P_i(X)$。

2.3.2 隐式抽象

在 1.1.1 小节，我们以反例路径的不存在性给出了模型检测问题 $\langle X, I(X), T(X, X'), P(X)\rangle$ 的安全性定义。该定义也可以通过一阶公式（可基于一阶理论）的可满足性来等价地给出：该问题是安全的，当且仅当对于任意的 $n \geqslant 1$，

$$I(X_1) \wedge \left(\bigwedge_{i=1}^{n-1} T(X_i, X_{i+1})\right) \wedge \neg P(X_n) \tag{2.1}$$

是不可满足的。该表示被称为展开（unrolling）[17]。

我们接下来考虑带有抽象的情形。设抽象函数为 h，那么公式 2.1 的抽象版本为：

$$\hat{I}(\hat{s}_1) \wedge \left(\bigwedge_{i=1}^{n-1} \hat{T}(\hat{s}_i, \hat{s}_{i+1})\right) \wedge \neg \hat{P}(\hat{s}_n). \tag{2.2}$$

将公式 1.1、1.2 和 1.3 代入公式 2.2，并消去存在量词，可得

$$I(X_1) \wedge \left(\bigwedge_{i=1}^{n-1} T(\overline{X}_i, X_{i+1})\right) \wedge \neg P(\overline{X}_n) \wedge \left(\bigwedge_{i=1}^{n} h(X_i) = h(\overline{X}_i) = \hat{s}_i\right). \tag{2.3}$$

可以看到，在公式 2.3 中，对于任意的 $1 \leqslant i \leqslant n$，$X_i$ 与 $\overline{X}_i$ 之间的联系在于它们对应同一个抽象变量。我们可以定义以下的基于抽象的具体状态的等价关系，并不妨称其为抽象等价关系：

$$EQ(X, \overline{X}) : h(X) = h(\overline{X}). \tag{2.4}$$

借助抽象等价关系，我们可以消去公式 2.3 中的抽象变量，然后得到：

$$I(X_1) \wedge \left(\bigwedge_{i=1}^{n-1} T(\overline{X}_i, X_{i+1}) \right) \wedge \neg P(\overline{X}_n) \wedge \left(\bigwedge_{i=1}^{n} EQ(X_i, \overline{X}_i) \right). \tag{2.5}$$

以上就是隐式抽象（implicit abstraction，IA）[18] 的基本思想了。隐式抽象的一个好处是，我们不需要去计算 $\hat{I}(\hat{s})$、$\hat{T}(\hat{s}, \hat{s}')$ 和 $\hat{P}(\hat{s})$ 了。

对于谓词抽象 $\{P_1(X), P_2(X), \cdots, P_n(X)\}$，抽象等价关系可以表示为：

$$EQ(X, \overline{X}) : \bigwedge_{i=1}^{n} (P_i(X) \leftrightarrow P_i(\overline{X})). \tag{2.6}$$

2.3.3 IC3+IA

在 IC3+IA 中，每个帧仍然是基于具体状态变量的。对于每个帧 $F_i(X)$，我们用 $\hat{F}_i(\hat{X})$ 来标记其抽象版本：$\hat{F}_i(\hat{X}) : \exists X, (h(X) = \hat{X}) \wedge F_i(X)$。我们希望 IC3 在谓词抽象状态空间上运行，所以在 IC3 帧序列的四条性质中，我们要分别用公式 1.1、1.2、1.3 中的定义来替代 $I(X)$、$T(X, X')$、$P(X)$，并用 $\hat{F}_i(\hat{X})$ 来替代 $F_i(X)$，进而得到 IC3+IA 帧序列所要满足的四条基本性质：

1. $\hat{F}_1(\hat{X}) \leftrightarrow \hat{I}(\hat{X})$；
2. 对于任意的 $1 \leqslant i \leqslant n-1$，$\hat{F}_i(\hat{X}) \rightarrow \hat{F}_{i+1}(\hat{X})$；
3. 对于任意的 $1 \leqslant i \leqslant n-1$，$\hat{F}_i(\hat{X}) \wedge \hat{T}(\hat{X}, \hat{X}') \rightarrow \hat{F}_{i+1}(\hat{X}')$；
4. 对于任意的 $1 \leqslant i \leqslant n-1$，$\hat{F}_i(\hat{X}) \rightarrow \hat{P}(\hat{X})$。

按照隐式抽象的思想，我们不计算 $\hat{I}(\hat{X})$、$\hat{T}(\hat{X}, \hat{X}')$、$\hat{P}(\hat{X})$ 和各 $\hat{F}_i(\hat{X})$。我们所要直接满足的是以下几条不含抽象公式的要求：

1. $I(X)$、$P(X)$ 和各 $F_i(X)$ 的所有原子都在 $\mathbb{P}$ 中，或者按照 IC3+IA 原文 [11] 的提法，这些公式都是 $\mathbb{P}$-公式；
2. $F_1(X) \leftrightarrow I(X)$；
3. 对于任意的 $1 \leqslant i \leqslant n-1$，$F_i(X) \rightarrow F_{i+1}(X)$；
4. 对于任意的 $1 \leqslant i \leqslant n-1$，$F_i(X) \wedge EQ(X, \overline{X}) \wedge T(\overline{X}, \overline{X}') \wedge EQ(\overline{X}', X') \rightarrow F_{i+1}(X')$；
5. 对于任意的 $1 \leqslant i \leqslant n-1$，$F_i(X) \rightarrow P(X)$。

可以证明，上述五条要求蕴涵了上文的四条性质。接着，我们大致介绍 IC3+IA 的流程，并关注上述五条要求是如何被满足的。首先，IC3+IA 接受用户所提供的谓词集合，并往该集合中加入 $I(X)$ 和 $P(X)$ 中的所有原子，形成初始谓词集合（考虑要求 1），并进而形成初始抽象状态空间，如同 CEGAR 那样。接下来的流程则

如同 IC3，同时主要有以下几处不同：

- 在阻塞阶段，我们不使用 Cube()，而是从一个状态衍生出一个“基本抽象立方体”。例如，对于状态 $(x=0, y=1, z=2)$，和 $\mathbb{P}=\{x+y=1, x-y=1\}$，所衍生出的基本抽象立方体不是 $(x=0)\wedge(y=1)\wedge(z=2)$，而是 $(x+y=1)\wedge\neg(x-y=1)$。
- 在阻塞阶段，我们所求解的公式不是 $F_{k-1}(X)\wedge\neg c(X)\wedge T(X,X')\wedge c(X')$，而是 $F_{k-1}(X)\wedge\neg c(X)\wedge EQ(X,\overline{X})\wedge T(\overline{X},\overline{X}')\wedge EQ(\overline{X}',X')\wedge c(X')$（考虑要求 4）。
- 在阻塞阶段，我们所获得的反例路径不是在具体状态空间下的，而是在抽象状态空间下的。如同 CEGAR 那样，我们需要考虑伪反例检查和精化。精化过程会往谓词集合中添加新谓词。
- 在传播阶段，我们所检查的公式不是 $F_k(X)\wedge T(X,X')\rightarrow c(X')$，而是 $F_k(X)\wedge EQ(X,\overline{X})\wedge T(\overline{X},\overline{X}')\wedge EQ(\overline{X}',X')\rightarrow c(X')$。

2.4 语法制导抽象

2019 年，Aman Goel 等人提出了一种被称为语法制导抽象（syntax-guided abstraction，SA）[10] 的新方法，以应对位级（bit-level）IC3 算法在求解硬件模型检测问题时所遇到的源于字宽的复杂度问题。该方法从给定的模型检测问题的语法中提取子谓词和子项，进而形成抽象状态空间，然后结合 IC3 和可选的未解释函数方法[19]，在 CEGAR 框架下求解问题。相比谓词抽象，语法制导抽象对子项之间的相等关系做出了额外的关注。

我们暂时跳过从语法中提取子谓词和子项的具体方式（原文[10]对此仅有简单举例，未作足够说明），并假设已经直接给定了谓词集合

$$\mathbb{P}=\{P_1(X), P_2(X), \cdots, P_n(X)\},$$

其中 $n\geqslant 0$，以及非布尔变量类型的集合

$$\mathbb{T}=\{T_1, T_2, \cdots, T_m\},$$

其中 $m\geqslant 0$，以及对于每个类型 T_i，再给定一个类型为 T_i 的项的集合

$$\mathbb{A}_i=\{A_{i;1}(X), A_{i;2}(X), \cdots, A_{i;l_i}(X)\},$$

其中 $l_i\geqslant 1$。语法制导抽象所定义的抽象状态空间中的抽象状态，相当于在被映射到该抽象状态的具体状态下，每个 $P_i(X)$ 的真值，以及对每个 $\mathbb{A}_i$ 的相等关系划分。

我们举一个例子：

例 2.1： 令 x, y, z 为二位位向量，$\mathbb{P} = \{x \leqslant y, y \leqslant z\}$，其中“$\leqslant$”相当于无符号整数比较，$\mathbb{A}_1 = \{01, x, y, z, x + y\}$，$\mathbb{A}_2 = \{0000, \text{CONCAT}(x, z)\}$，那么状态 s：$(x = 00, y = 01, z = 00)$ 对应的抽象状态为：

$$\begin{aligned}\hat{s}：&(x \leqslant y) \wedge \neg(y \leqslant z)\\ &\wedge \{\{x, z\}, \{01, y, x + y\}\}\\ &\wedge \{\{0000, \text{CONCAT}(x, z)\}\}.\end{aligned}$$

相应的基本抽象立方体为：

$$\begin{aligned}\text{AbsCube}(s)：&(x \leqslant y) \wedge \neg(y \leqslant z)\\ &\wedge (z = x) \wedge (01 \neq x) \wedge (y = 01) \wedge (x + y = 01)\\ &\wedge (\text{CONCAT}(x, z) = 0000).\end{aligned}$$ ∎

2.4.1 IC3+SA

原文[10] 同时提出了一种将语法制导抽象与 IC3 相结合的方法。与 IC3+IA 不同，该方法所维护的帧序列的性质和非抽象 IC3 完全一样。该方法与非抽象 IC3 的一个不同之处在于，在原阻塞阶段算法（算法 2.1）的第 14 行，我们用 AbsCube(s_{pre}) 来代替 Cube(s_{pre})，并采用基于 cone of influence 的方法进行进一步泛化。这会导致我们所得到的反例（若有）是抽象的而非具体的。原文采用了一种基于前像计算（forward image computation）[20]的方法来进行伪反例检查和精化。

第 3 章　对 IC3+IA 和语法制导抽象的讨论

3.1　对 IC3+IA 的推广

前文提到，在 IC3+IA 的执行过程中，我们需要确保 $I(X)$、$P(X)$ 和各 $F_i(X)$ 都是 $\mathbb{P}$-公式。该要求是针对谓词抽象的，因此我们希望将其推广至任意抽象的情形。跟随原文[11]的算法正确性证明过程，我们发现，$\mathbb{P}$-公式的要求可以被推广至“整洁性”。另外，作者对 IC3+SA 进行了分析，发现其尽管在流程上与 IC3+IA 有所不同，但也同样需要满足上述整洁性的要求，尽管其原文[10]并没有提到这一点。以下给出整洁性的定义：

定义 3.1 (整洁性)： 给定具体状态空间 S 和抽象 $\langle S_{abs}, h\rangle$，我们说一个关于 S 的公式 $F(s)$ 相对于 $\langle S_{abs}, h\rangle$ 是整洁的，当且仅当 $EQ(s,\overline{s}) \to (F(s) \leftrightarrow F(\overline{s}))$。 ∎

通俗地说，如果 $F(s)$ 是整洁的，那么对于任意一个抽象状态，其所对应的所有具体状态，要么全让 $F(s)$ 为真，要么全让 $F(s)$ 为假。

以下是整洁性的几条基本性质，以及它与 $\mathbb{P}$-公式的关系：

定理 3.1： $\top$ 和 $\bot$ 相对于任意抽象是整洁的。 ∎

定理 3.2： 谓词抽象和语法制导抽象的所有基本抽象立方体都是整洁的。 ∎

定理 3.3： 如果 $F_1(s)$ 和 $F_2(s)$ 是整洁的，那么 $\neg F_1(s)$ 和 $F_1(s) \wedge F_2(s)$ 是整洁的。即，逻辑运算能够保持整洁性。 ∎

定理 3.4： 在谓词抽象中，包括 $I(X)$ 和 $P(X)$ 在内的所有 $\mathbb{P}$-公式都是整洁的。 ∎

3.2　语法制导抽象的抽象等价关系表示

在 IC3+IA 中，我们需要求解 $F_{k-1}(X) \wedge \neg c(X) \wedge EQ(X,\overline{X}) \wedge T(\overline{X},\overline{X}') \wedge EQ(\overline{X}',X') \wedge c(X')$，因此我们需要考虑如何表示语法制导抽象的抽象等价关系。给定 $\mathbb{P} = \{P_1(X), P_2(X), \cdots, P_n(X)\}$、$\mathbb{T} = \{T_1, T_2, \cdots, T_m\}$ 和 m 个 $\mathbb{A}_i = \{A_{i;1}(X), A_{i;2}(X), \cdots, A_{i;l_i}(X)\}$（见 2.4 节），以下是一种可能的表示：

$$\begin{aligned} EQ(X,\overline{X}) : & \left(\bigwedge_i (P_i(X) \leftrightarrow P_i(\overline{X}))\right) \\ & \wedge \left(\bigwedge_{i;j\neq k} ((A_{i;j}(X) = A_{i;k}(X)) \leftrightarrow (A_{i;j}(\overline{X}) = A_{i;k}(\overline{X})))\right). \end{aligned} \tag{3.1}$$

在最坏情况下，该公式的长度正比于项个数的平方。考虑到较长的抽象等价关系表示会影响公式求解性能，这样的复杂度并不是非常理想，但是目前来看，这已经是最简单的表示了。

从谓词抽象和语法制导抽象的抽象等价关系表示中，我们能够看到，谓词抽象和语法制导抽象的表达能力是相同的。对于由 $\mathbb{P}_{PA}$ 所确定的谓词抽象，我们令 $\mathbb{P}_{SA} = \mathbb{P}_{PA}$ 且 $\mathbb{T} = \varnothing$，即可得到一个等价的语法制导抽象；对于由 $\mathbb{P}_{SA}$、$\mathbb{T} = \{T_1, T_2, \cdots, T_m\}$ 和 m 个 $\mathbb{A}_i = \{A_{i;1}(X), A_{i;2}(X), \cdots, A_{i;l_i}(X)\}$ 所确定的语法制导抽象，我们令 $\mathbb{P}_{PA} = \mathbb{P}_{SA} \cup \{A_{i;j} = A_{i;k} : j \neq k\}$，即可得到一个等价的谓词抽象。

3.3 语法制导抽象子谓词和子项的抽取

语法制导抽象的子谓词和子项的抽取方式会影响初始抽象状态空间的构建以及精化的结果。前文提到，原文[10]对抽取方式未作足够说明，对此我们在本节以如何为语法制导抽象构建初始抽象状态空间为出发点展开讨论。

根据已有讨论，我们需要确保在初始抽象状态空间下，$I(X)$ 和 $P(X)$ 是整洁的。由于语法制导抽象涉及到项，我们首先将整洁性的定义从公式推广至公式和项：

定义 3.2 (项的整洁性)： 给定具体状态空间 S 和抽象 $\langle S_{abs}, h\rangle$，我们说一个关于 S 的项 $A(s)$ 相对于 $\langle S_{abs}, h\rangle$ 是整洁的，当且仅当 $EQ(s, \overline{s}) \to (A(s) = A(\overline{s}))$。 ∎

推广后，整洁性的基本性质可以作以下补充：

定理 3.5： 如果公式 $F(s)$ 和项 $A_1(s), A_2(s)$ 是整洁的，那么 $A_1(s) = A_2(s)$ 和 $\mathrm{ITE}(F(s), A_1(s), A_2(s))$ 是整洁的，其中 ITE 代表 *if-then-else*。 ∎

事实上，不论是谓词抽象还是语法制导抽象，我们都可以令初始抽象状态空间仅由 $I(X)$ 和 $P(X)$ 两个谓词形成。这已经足够确保 $I(X)$ 和 $P(X)$ 的整洁性，但该抽象太过粗糙，并不实用。类似于谓词抽象对公式原子的抽取，根据整洁性的性质，我们可以为语法制导抽象定义一种“最小”抽取方式，见算法 3.1。

例 3.1： $\mathrm{MinExtract}((x \geqslant 0) \wedge (y + z = 1)) = \{x \geqslant 0, 1, y + z\}$. ∎

我们将由包含了谓词和项的集合 G 所确定的语法制导抽象记为 $\mathrm{SA}(G)$。以下是最小抽取的两条性质：

定理 3.6： 对于任意的公式 $F(X)$，$F(X)$ 相对于 $\mathrm{SA}(\mathrm{MinExtract}(F(X)))$ 是整洁的。 ∎

定理 3.7： 存在某些公式 $F(X)$，对于 $\mathrm{MinExtract}(F(X))$ 的任意真子集 G'，$F(X)$

算法 3.1 语法制导抽象，最小抽取

```
1:  function MinExtract(A(X): a formula or a term)
2:      if A(X) is ⊤ or ⊥ then
3:          return ∅
4:      else if A(X) has the form ¬A_1(X) then
5:          return MinExtract(A_1(X))
6:      else if A(X) has the form A_1(X) ∧ A_2(X), A_1(X) ∨ A_2(X), A_1(X) → A_2(X), A_1(X) ↔
        A_2(X) or A_1(X) = A_2(X) then
7:          return MinExtract(A_1(X)) ∪ MinExtract(A_2(X))
8:      else if A(X) has the form ITE(A_1(X), A_2(X), A_3(X)) then
9:          return MinExtract(A_1(X)) ∪ MinExtract(A_2(X)) ∪ MinExtract(A_3(X))
10:     else
11:         return {A(X)}
12:     end if
13: end function
```

相对于 SA(G') 是非整洁的。 ∎

综上，我们令初始抽象为 SA(MinExtract($I(X)$)∪MinExtract($P(X)$))，就可以确保 $I(X)$ 和 $P(X)$ 相对于初始抽象的整洁性了。

除了最小抽取本身，我们也可以通过对最小抽取进行扩展来设计新的抽取规则，例如从 $(x \geqslant 0) \wedge (y + z = 1)$ 中额外抽取出 y 和 z。另外，除了 $I(X)$ 和 $P(X)$，我们也可以额外对 $T(X, X')$ 作抽取（要去除带撇号变量的参与），如同语法制导抽象原文[10]所提供的例子那样。抽取出的子谓词和子项越多，所形成的语法制导抽象就越精细，这会减少伪反例的出现，但同时也会让抽象状态空间变得更大，进而增大求解抽象问题的开销。抽取更多的子谓词和子项所带来的利弊，是需要我们来权衡的。

3.4 IC3+IA 的等价形式

本节将论证，在 IC3+IA 的算法流程中，抽象等价关系的参与可以被去除，同时确保去除后所得到的新算法做的仍是和原算法完全等价的事情。这让我们有机会避免涉及复杂度较高的语法制导抽象的抽象等价关系表示，同时也为我们将 IC3+SA 的泛化和精化方法直接应用于 IC3+IA 提供了可能。具体地，新算法（不妨称其为等价形式）主要产生了以下两点变更：

- 在新算法中，我们不再关注帧序列的“$F_i(X) \wedge EQ(X, \overline{X}) \wedge T(\overline{X}, \overline{X}') \wedge EQ(\overline{X}', X') \rightarrow F_{i+1}(X')$”性质。我们把该公式替换为 $F_i(X) \wedge T(X, X') \rightarrow F_{i+1}(X')$，与非抽象 IC3 相同。

- 在阻塞阶段，我们不再求解 $F_{k-1}(X) \wedge \neg c(X) \wedge EQ(X,\overline{X}) \wedge T(\overline{X},\overline{X}') \wedge EQ(\overline{X}',X') \wedge c(X')$。我们把该公式替换为 $F_{k-1}(X) \wedge \neg c(X) \wedge T(X,X') \wedge c(X')$，与非抽象 IC3 相同。

为说明新旧算法之间的等价性，以下给出三条相关定理：

定理 3.8：如果 $F(X)$ 和 $c(X)$ 是整洁的，那么：$F(X) \wedge \neg c(X) \wedge EQ(X,\overline{X}) \wedge T(\overline{X},\overline{X}') \wedge EQ(\overline{X}',X') \wedge c(X')$ 可满足，当且仅当 $F(X) \wedge \neg c(X) \wedge T(X,X') \wedge c(X')$ 可满足。

证明： 必要性。如果 $F(X) \wedge \neg c(X) \wedge EQ(X,\overline{X}) \wedge T(\overline{X},\overline{X}') \wedge EQ(\overline{X}',X') \wedge c(X')$ 可满足，那么存在状态 $X_0, \overline{X}_0, X_0', \overline{X}_0'$，使得 $F(X_0) \wedge \neg c(X_0) \wedge EQ(X_0,\overline{X}_0) \wedge T(\overline{X}_0,\overline{X}_0') \wedge EQ(\overline{X}_0',X_0') \wedge c(X_0')$ 求值为真，其中 $F(X_0)$ 代表 $F(X)$ 中的自由变量被状态 X_0 对 X 的赋值所代入。从中我们知道，X_0 和 $\overline{X}_0$ 是抽象等价的，X_0' 和 $\overline{X}_0'$ 也是抽象等价的，考虑到 $F(X)$ 和 $c(X)$ 的整洁性，我们可以用 $F(\overline{X}_0)$ 来代替 $F(X_0)$，用 $c(\overline{X}_0)$ 来代替 $c(X_0)$，用 $c(\overline{X}_0')$ 来代替 $c(X_0')$，并去掉 $EQ(X_0,\overline{X}_0)$ 和 $EQ(\overline{X}_0',X_0')$，进而得到 $F(\overline{X}_0) \wedge \neg c(\overline{X}_0) \wedge T(\overline{X}_0,\overline{X}_0') \wedge c(\overline{X}_0')$ 求值为真，进而 $F(X) \wedge \neg c(X) \wedge T(X,X') \wedge c(X')$ 可满足。

充分性。如果 $F(X) \wedge \neg c(X) \wedge T(X,X') \wedge c(X')$ 可满足，那么存在状态 X_0 和 X_0'，使得 $F(X_0) \wedge \neg c(X_0) \wedge T(X_0,X_0') \wedge c(X_0')$ 求值为真。考虑到抽象等价关系的自反性，我们有 $EQ(X_0,X_0)$ 和 $EQ(X_0',X_0')$ 求值为真，进而得到 $F(X_0) \wedge \neg c(X_0) \wedge EQ(X_0,X_0) \wedge T(X_0,X_0') \wedge EQ(X_0',X_0') \wedge c(X_0')$ 求值为真，进而 $F(X) \wedge \neg c(X) \wedge EQ(X,\overline{X}) \wedge T(\overline{X},\overline{X}') \wedge EQ(\overline{X}',X') \wedge c(X')$ 可满足。 ∎

定理 3.9： 如果 $F(X)$ 和 $G(X)$ 是整洁的，那么：$F(X) \wedge EQ(X,\overline{X}) \wedge T(\overline{X},\overline{X}') \wedge EQ(\overline{X}',X') \rightarrow G(X')$ 成立，当且仅当 $F(X) \wedge T(X,X') \rightarrow G(X')$ 成立。

证明： 证明 $F(X) \wedge EQ(X,\overline{X}) \wedge T(\overline{X},\overline{X}') \wedge EQ(\overline{X}',X') \wedge \neg G(X')$ 可满足当且仅当 $F(X) \wedge T(X,X') \wedge \neg G(X')$ 可满足即可。证明过程与定理 3.8 类似。 ∎

定理 3.10：如果 $F(X)$ 和 $c(X)$ 是整洁的，且 $F(X) \wedge \neg c(X) \wedge EQ(X,\overline{X}) \wedge T(\overline{X},\overline{X}') \wedge EQ(\overline{X}',X') \wedge c(X')$ 和 $F(X) \wedge \neg c(X) \wedge T(X,X') \wedge c(X')$ 均为可满足，那么我们可以令 A_{EQ} 和 A 分别为能够从前者和后者的可满足解中衍生出的基本抽象立方体的集合，即 $A = \{\text{AbsCube}(X_0) : \text{IsSat}(F(X_0) \wedge \neg c(X_0) \wedge T(X_0,X') \wedge c(X'))\}$，其中 IsSat 代表该公式可满足。我们有：$A = A_{EQ}$。

证明： 令 $B = \{X_0 : \text{IsSat}(F(X_0) \wedge \neg c(X_0) \wedge T(X_0,X') \wedge c(X'))\}$，$B_{EQ} = \{X_0 : \text{IsSat}(F(X_0) \wedge \neg c(X_0) \wedge EQ(X_0,\overline{X}) \wedge T(\overline{X},\overline{X}') \wedge EQ(\overline{X}',X') \wedge c(X'))\}$，则 $A = \{\text{AbsCube}(X_0) : X_0 \in B\}$，$A_{EQ} = \{\text{AbsCube}(X_0) : X_0 \in B_{EQ}\}$。我们证明 $B_{EQ} = \{X_0 : \exists \overline{X}_0 \in B, EQ(X_0,\overline{X}_0)\}$ 即可，因为抽象等价的两个状态所衍生出的基本抽

象立方体是相同的。

如果状态 X_0 使得 $F(X_0)\wedge\neg c(X_0)\wedge EQ(X_0,\overline{X})\wedge T(\overline{X},\overline{X}')\wedge EQ(\overline{X}',X')\wedge c(X')$ 可满足，那么，存在状态 $\overline{X}_0, X_0', \overline{X}_0'$，使得 $F(X_0)\wedge\neg c(X_0)\wedge EQ(X_0,\overline{X}_0)\wedge T(\overline{X}_0,\overline{X}_0')\wedge EQ(\overline{X}_0',X_0')\wedge c(X_0')$ 求值为真，进而 $F(\overline{X}_0)\wedge\neg c(\overline{X}_0)\wedge T(\overline{X}_0,\overline{X}_0')\wedge c(\overline{X}_0')$ 求值为真，进而 $\overline{X}_0$ 使得 $F(\overline{X}_0)\wedge\neg c(\overline{X}_0)\wedge T(\overline{X}_0,X')\wedge c(X')$ 可满足，进而 $\overline{X}_0\in B$。所以，$B_{EQ}\subseteq\{X_0:\exists\overline{X}_0\in B, EQ(X_0,\overline{X}_0)\}$。

如果状态 X_0 使得 $\exists\overline{X}_0\in B, EQ(X_0,\overline{X}_0)$，那么存在状态 $\overline{X}_0,\overline{X}_0'$，使得 X_0 和 $\overline{X}_0$ 是抽象等价的，且 $F(\overline{X}_0)\wedge\neg c(\overline{X}_0)\wedge T(\overline{X}_0,\overline{X}_0')\wedge c(\overline{X}_0')$ 求值为真，进而 $F(X_0)\wedge\neg c(X_0)\wedge EQ(X_0,\overline{X}_0)\wedge T(\overline{X}_0,\overline{X}_0')\wedge EQ(\overline{X}_0',\overline{X}_0')\wedge c(\overline{X}_0')$ 求值为真，进而 $F(X_0)\wedge\neg c(X_0)\wedge EQ(X_0,\overline{X})\wedge T(\overline{X},\overline{X}')\wedge EQ(\overline{X}',X')\wedge c(X')$ 可满足。所以，$\{X_0:\exists\overline{X}_0\in B, EQ(X_0,\overline{X}_0)\}\subseteq B_{EQ}$。

综上，$B_{EQ}=\{X_0:\exists\overline{X}_0\in B, EQ(X_0,\overline{X}_0)\}$，进而 $A=A_{EQ}$。 ∎

事实上，新算法和原算法的性能还是会有差异的，尽管二者等价，这是因为二者传给 SMT（satisfiability modulo theories，可满足性模理论）[21]求解器的公式在形式上有差异。本文认为，二者之间谁的性能更好不能一概而论，这是因为新算法虽然将几个带有抽象等价关系的公式简化了，但同时也让简化后的公式的解集变得更小了（见定理 3.10 的证明过程，$B_{EQ}=\{X_0:\exists\overline{X}_0\in B, EQ(X_0,\overline{X}_0)\}$），进而潜在地增加了求解器找到解的难度。二者性能的差异将主要取决于当前模型检测算法实现所依赖的求解器的实现。

3.5 谓词抽象泛化和精化方法对语法制导抽象的适用性

本节将论证，在我们把语法制导抽象应用于 IC3+IA 时，除了选择 IC3+SA 的泛化和精化方法，我们也可以考虑采用和谓词抽象相同或相似的泛化和精化方法。

3.5.1 泛化

由于谓词抽象状态空间本身相当于一个布尔变量状态空间，且 IC3+IA 在算法起始就确保了 $I(X)$、$P(X)$ 和各 $F_i(X)$ 的整洁性，IC3+IA 可以采用在本质上和非抽象 IC3 一样的泛化方法。对于语法制导抽象，我们通过以下定理来说明非抽象 IC3 泛化方法对语法制导抽象的适用性：

定理 3.11： 设 $c(X)$ 是一个语法制导抽象的基本抽象立方体，则从 $c(X)$ 中移除掉任意多个文字后，所得公式相对于当前抽象仍是整洁的。 ∎

3.5.2 精化

IC3+IA 原文[11]所附带的参考实现采用了基于插值（interpolation）[22]的精化方法。简单地说，我们可以把插值方法当作一个黑箱，在发现伪反例后，插值方法能够给出 $n\,(n \geqslant 1)$ 个公式 $Q_1(X), \cdots, Q_n(X)$，然后谓词抽象从这 n 个公式中提取原子，就可以确保消除当前伪反例。考虑到语法制导抽象的表达能力与谓词抽象相同，我们同样可以把插值方法应用于语法制导抽象。我们可以直接把这 n 个公式中的原子加入到语法制导抽象的谓词集合，但为了更好地体现语法制导抽象对子项相等关系的关注，我们也可以选择对这 n 个公式进行最小抽取，该做法所形成的新抽象会比前者（抽取原子）更精细，因此也能够确保消除当前伪反例。

第 4 章　IC3+IA+SA 算法和实现

在上一章中，我们讨论了几个关于 IC3+IA 和语法制导抽象的重要问题。基于上一章的讨论，我们在本章将给出我们把语法制导抽象应用于 IC3+IA 的具体方案，包括默认方案和一些可能的变体，所得算法不妨称为 IC3+IA+SA。

4.1　算法描述

与 IC3+IA 类似，IC3+IA+SA 首先构建初始抽象状态空间，然后令初始帧序列 $frames : I(X)$，接着交替进行阻塞阶段和传播阶段。

4.1.1　初始抽象状态空间的构建

在默认方案中，对 $I(X)$ 和 $P(X)$ 作最小抽取。按照上一章的讨论，一种可能的变体是，对 $T(X, X')$ 作最小抽取，并把 X 中的所有变量添加至谓词集合或项集合。

4.1.2　阻塞阶段

算法 4.1 给出了 IC3+IA+SA 阻塞阶段的伪代码流程。该流程中有几处步骤存在变体：

- 在算法第 5 行，默认方案是进行泛化。另一种变体是，不进行泛化，即直接令 $c(X) := \text{AbsCube}(s)$。
- 在算法第 18 行和第 28 行，默认方案是采用带有抽象等价关系的公式。另一种变体是，采用不带抽象等价关系的公式。
- 在算法第 23 行，默认方案是不进行泛化（原因在 4.2 节解释）。可能的变体有，采用非抽象 IC3 泛化方法，如同谓词抽象那样，或是采用 IC3+SA 的泛化方法。
- 在算法第 12 行至第 17 行，默认方案是采用插值精化方法。另一种变体是，采用 IC3+SA 的精化方法。

4.1.3　传播阶段

在默认方案中，与 IC3+IA 相同。一种可能的变体是，与 IC3+IA 的等价形式相同。

算法 4.1 IC3+IA+SA，阻塞阶段

while there exists state s, s.t. $s \vDash F_n(X) \wedge \neg P(X)$ **do**
  if $n = 1$ **then**
    $c(X) := \text{AbsCube}(s)$
  else
    find a cube $c(X)$, s.t. AbsCube$(s) \rightarrow c(X)$, and $c(X) \rightarrow \neg P(X)$
  end if
  $obligations := \text{Heap}(\{(c(X), n, \text{Null})\})$
  while $obligations$ is not empty **do**
    $(c(X), k, parent) := obligations.\text{top}()$
    if $k = 1$ **then**
      get cube chain $c_1(X), \cdots, c_n(X)$ along parent pointers
      if there exists a concrete $counterexample$ in cube chain **then**
        return $counterexample$
      else
        perform refinement with cube chain
        break
      end if
    else if $F_{k-1}(X) \wedge \neg c(X) \wedge EQ(X, \overline{X}) \wedge T(\overline{X}, \overline{X}') \wedge EQ(\overline{X}', X') \wedge c(X')$ (or the version without EQ) is satisfiable **then**
      get a state s_{pre} in $F_{k-1}(X) \wedge \neg c(X)$ from satisfiable solution
      if $k = 2$ **then**
        $c_{pre}(X) := \text{AbsCube}(s_{pre})$
      else
        get a cube $c_{pre}(X)$, which may or may not have been generalized, from s_{pre}
        (see discussion in subsection 4.1.2 for more details)
      end if
      $obligations.\text{push}(c_{pre}(X), k-1, pointer\ to\ current\ top)$
    else
      find a cube $c_g(X)$, s.t. $c(X) \rightarrow c_g(X)$, and $F_{k-1}(X) \wedge \neg c_g(X) \wedge EQ(X, \overline{X}) \wedge T(\overline{X}, \overline{X}') \wedge EQ(\overline{X}', X') \wedge c_g(X')$ (or the version without EQ) is unsatisfiable
      for $i = 2, \cdots, k$ **do**
        $F_i(X) := F_i(X) \wedge \neg c_g(X)$
      end for
      $obligations.\text{pop}()$
    end if
  end while
end while

4.2 代码实现

对 IC3+IA+SA 的实现基于本文作者进行本课题时所在课题组的一位研究生所设计的一套支持多种抽象方法的 IC3+IA 框架，本文作者在该框架中实现了本文所涉及的语法制导抽象的所有特有逻辑。该框架基于 IC3+IA 原文[11]所附带的参考实现，该参考实现基于 MathSAT 5[23] 求解器。IC3+IA+SA 实现、IC3+IA 框架和 IC3+IA 原文参考实现都基于 C++ 编程语言。另外，IC3+IA 原文参考实现的前像泛化（对应算法 4.1 第 23 行）会导致 IC3+IA 算法在个别样例下给出错误结果，因此 IC3+IA+SA 对前像泛化暂时不予考虑。

另外，考虑到 MathSAT 的库 API 是 C 形式的，为提高开发体验，本文作者还对 MathSAT 5 进行了一层轻量的 C++ 封装，作为本课题的一个副产品，主要达到了以下几个效果：

- 最大限度地避免了潜在的由于忽略库 API 执行出错（例如，假定返回非空指针，实际返回的却就是空指针）所引发的未定义行为；
- 利用 C++ RAII 特性，实现了 MathSAT 对象的自动资源回收；
- 利用 C++ 特性，令封装后的 API 更符合工效学；
- 在达到以上几个效果的同时，不引入任何不必要的计算或内存开销。

4.3 变体的可用性

在上文所提到的算法步骤变体中，并不是所有的变体都实际可行或可用。目前，经分析和测试发现，以下几种变体是不可用的：

- 在初始抽象状态空间的构建过程中，对 $T(X, X')$ 作最小抽取，和（或）把 X 中的所有变量添加到谓词集合或项集合的方案是不可行的。实际测试发现，这会引入过多的子谓词和子项，导致抽象过于精细，进而导致 IC3+IA+SA 能在有限一段时间内给出验证结果的样例非常少。
- IC3+SA 提供了一种精化方法，然而本文作者对该方法的实现目前还存在一些问题：
  - 求 unsat core（见语法制导抽象原文[10]）会出错。根据目前的测试和判断，这是由 MathSAT 导致的。
  - 如果不使用 unsat core，在某些样例下就会出现精化失败的情况，即精化过程没有获得任何新的子谓词和子项，其原因目前暂未完全清楚。
- IC3+SA 也提供了泛化方法，然而该泛化方法对 IC3+SA 的精化方法存在依

赖，这是因为该泛化方法不确保证明义务链条的严格连续性（见 2.2.1.1 小节），进而使得该泛化方法不能直接和插值精化方法相结合。因此，该泛化方法目前也是不可用的。

目前可用的方案主要有两个：默认方案（后文称“语法制导抽象（有 ***EQ*** 版本）”，或直接称“语法制导抽象”），以及去除了抽象等价关系参与的变体（后文称“语法制导抽象（无 ***EQ*** 版本）”）。这两个版本将参与下一章的实验。

第 5 章　实验和分析

本章我们将对语法制导抽象的两个版本（见上一章）进行正确性测试和性能测试，并进行相应的分析。

5.1　实验设定

实验运行在以下环境：

- CPU：AMD EPYC 7H12 64-Core (x2)；
- 内存：64 GiB (x16)；
- 操作系统：Ubuntu 20.04.2 LTS；
- Docker version 20.10.6, build 370c289。

我们选用来自 2020 年硬件模型检测大赛（HWMCC'20）的 324 个基于 BV（bit vector）理论的样例，并以 IC3+IA 框架的谓词抽象版本（不使用前像泛化）为对照。三个版本同时开始运行，同一时刻同一版本最多并行运行 32 个样例，每个运行实例（即对应一个版本和一个样例）等效地占用一个 CPU 逻辑核心，并最多能够运行 1 小时。

5.2　实验结果和分析

5.2.1　正确性

有多个模型检测工具参加了 2020 年硬件模型检测大赛。对于一个样例，我们这样定义 ground truth：

- 如果存在一个工具所给出的验证结果为不安全，那么我们认为该样例是不安全的；
- 否则，如果存在一个工具所给出的验证结果为安全，那么我们认为该样例是安全的；
- 否则（即所有工具都未能给出结果，原因可能是超时、超出内存限制，等等），我们不对该样例的实际安全性作出判断。

本文实验完成后，按照以上 ground truth，我们没有发现任何正确性问题。即，所有的版本对所有样例所给出的结果，都没有和 ground truth 产生矛盾。

5.2.2 性能

我们对三个版本进行两两对比，并关注以下三个问题：

- （Solved）前一版本和后一版本分别对多少个样例给出了验证结果；
- （Better）有多少个样例，前一版本比后一版本验证耗时更短，以及相反；
- （Unique）有多少个样例，前一版本给出了结果而后一版本没有，以及相反；

表 5.1 对这三个问题给出了回答。另外，图 5.1、图 5.2 和图 5.3 以散点图形式呈现了两两对比的原始数据，其中每个点代表一个样例，其横纵坐标分别代表前一版本和后一版本的运行时间，单位为秒。我们可以得出以下几个结论：

- 不论是有 *EQ* 版本的语法制导抽象还是无 *EQ* 版本，其总体效果都不如谓词抽象好。这可能是因为，语法制导抽象在 IC3+IA 的场景下引入了偏多的对求解问题没有帮助的精度。
- 尽管如此，两个版本的语法制导抽象还是在部分样例下取得了比谓词抽象更好的效果。这说明，至于是语法制导抽象的效果更好，还是谓词抽象的效果更好，要取决于问题（样例）的具体特征。因此可以认为，语法制导抽象在 IC3+IA 的场景下仍然对部分问题有较好的适用性。
- 无 *EQ* 版本的语法制导抽象的效果相比有 *EQ* 版本有一定提升，但也相当有限，尽管语法制导抽象的抽象等价关系表示具有较高的复杂度。这验证了前文的观点：对于 IC3+IA 原始算法和去除掉抽象等价关系参与的等价形式，谁的性能更好不能一概而论。

表 5.1　三个版本的两两对比结果

Comp	Solved	Better	Unique
PA VS. SA	88:69	63:29	24:5
PA VS. SA (without *EQ*)	88:71	58:36	23:6
SA VS. SA (without *EQ*)	69:71	28:46	4:6

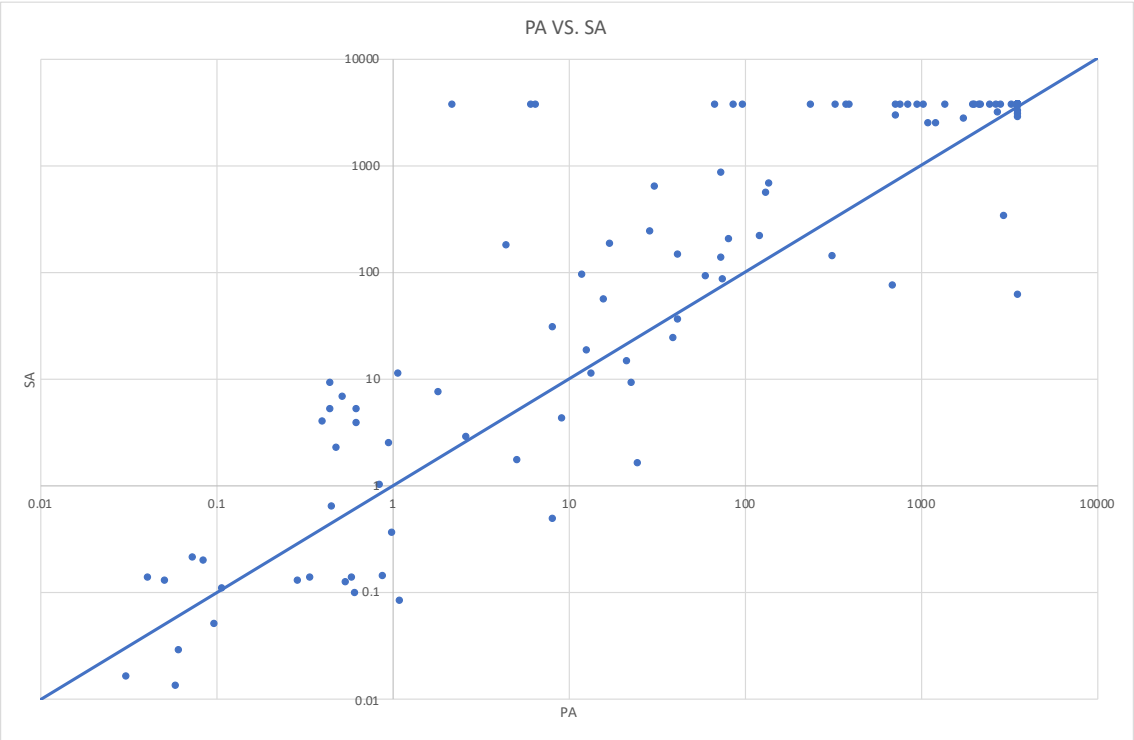

图 5.1　谓词抽象与语法制导抽象的对比

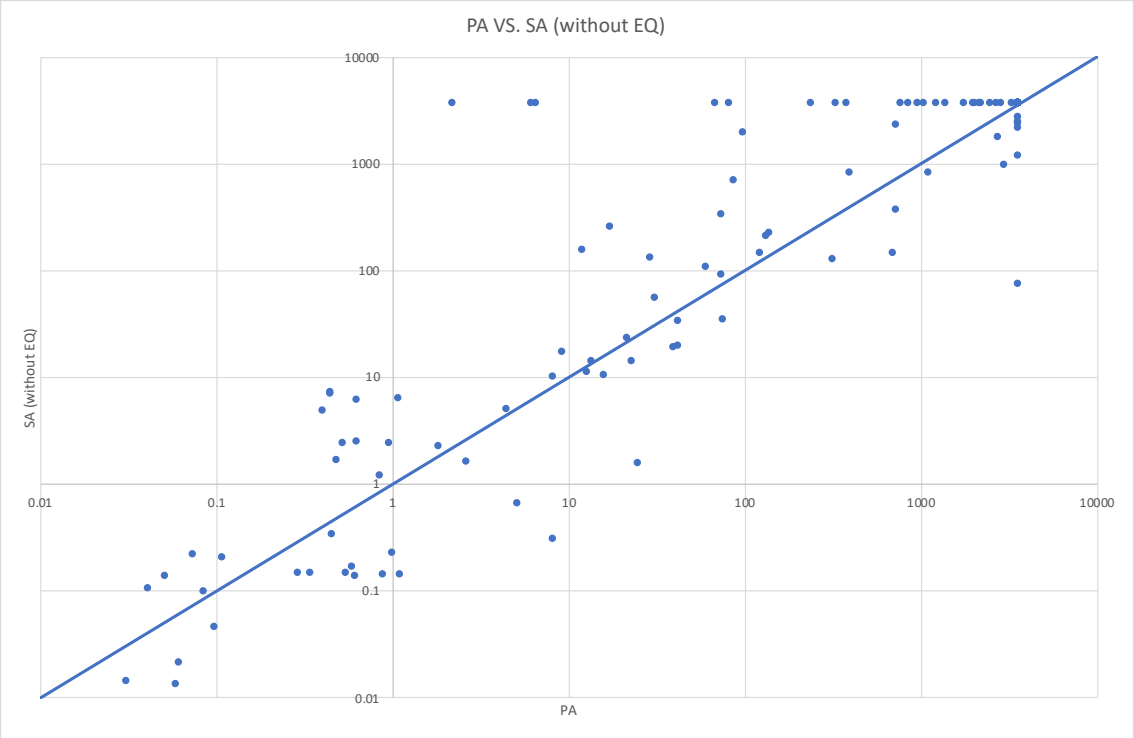

图 5.2　谓词抽象与语法制导抽象（无 ***EQ*** 版本）的对比

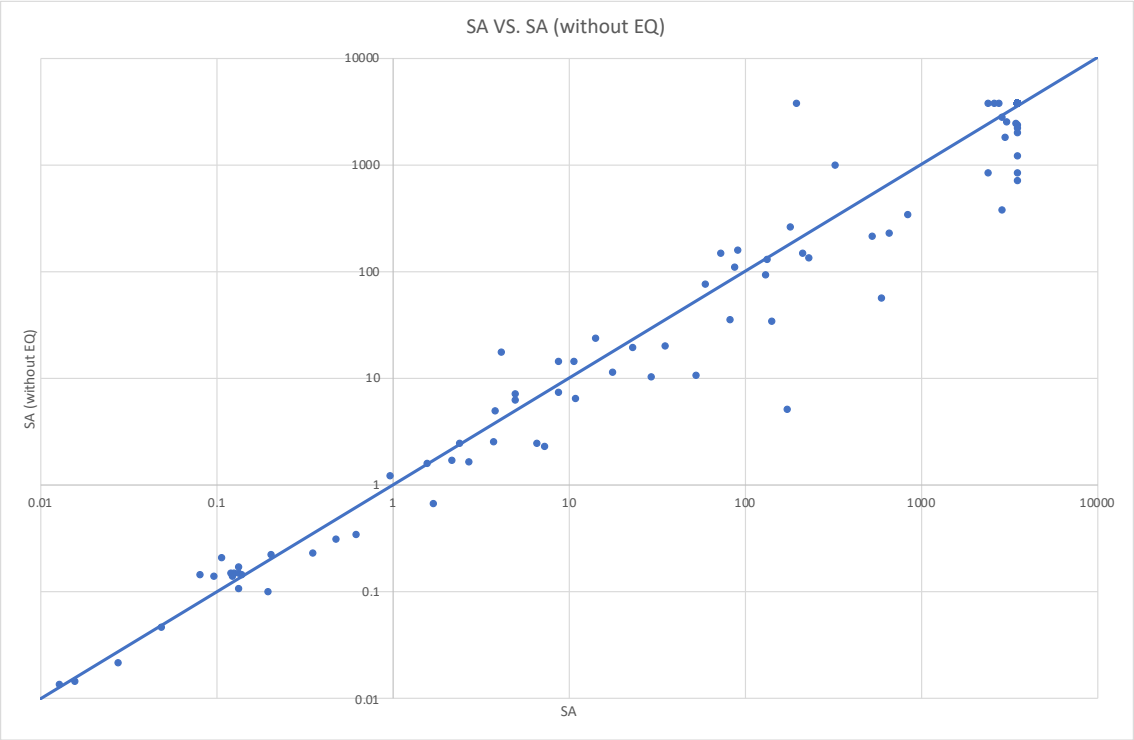

图 5.3　语法制导抽象与语法制导抽象（无 ***EQ*** 版本）的对比

第 6 章　总结和展望

模型检测，是一种形式化的验证系统行为正确性的方法。在模型检测领域中，人们已经提出了 IC3 等多种多样的模型检测算法。对于较复杂的问题，模型检测算法往往会遇到状态空间爆炸问题，因此我们常常需要考虑对原问题进行状态空间抽象，以降低状态空间的复杂度。本文对将语法制导抽象应用于 IC3+IA 所需要注意的问题进行了讨论，提出了多种可能的具体方案，并在一套 IC3+IA 框架中进行了代码实现。目前实际可用的方案有两种，即保留抽象等价关系参与的方案和将其去除的方案。本文最后对这两种方案连同谓词抽象进行了实验，得到了整体上不太理想但仍有部分可取之处的结果。

本文后续的一大工作方向是，解决目前所遇到的一些有关 IC3+SA 泛化方法和精化方法的代码实现问题。我们认为，如果能将二者成功应用于当前体系，我们的语法制导抽象代码实现的性能将有机会获得进一步提升。

插图索引

表格索引

算法索引

参考文献

[1] Lipp M, Schwarz M, Gruss D, et al. Meltdown: Reading kernel memory from user space[C]// 27th {USENIX} Security Symposium ({USENIX} Security 18). 2018: 973-990.

[2] Kocher P, Horn J, Fogh A, et al. Spectre attacks: Exploiting speculative execution[C]//2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019: 1-19.

[3] Durumeric Z, Li F, Kasten J, et al. The matter of heartbleed[C]//Proceedings of the 2014 conference on internet measurement conference. 2014: 475-488.

[4] Ehrenfeld J M. Wannacry, cybersecurity and health information technology: A time to act[J]. Journal of medical systems, 2017, 41(7): 104.

[5] Clarke E M. Model checking[C]//International Conference on Foundations of Software Technology and Theoretical Computer Science. Springer, 1997: 54-56.

[6] Clarke E, Grumberg O, Jha S, et al. Counterexample-guided abstraction refinement for symbolic model checking[J]. Journal of the ACM (JACM), 2003, 50(5): 752-794.

[7] Clarke E M, Grumberg O, Long D E. Model checking and abstraction[J]. ACM transactions on Programming Languages and Systems (TOPLAS), 1994, 16(5): 1512-1542.

[8] Lee W, Pardo A, Jang J Y, et al. Tearing based automatic abstraction for ctl model checking [C]//Proceedings of international conference on computer aided design. IEEE, 1996: 76-81.

[9] Bradley A R, Manna Z. The calculus of computation: decision procedures with applications to verification[M]. Springer Science & Business Media, 2007.

[10] Goel A, Sakallah K. Model checking of verilog rtl using ic3 with syntax-guided abstraction [C]//NASA Formal Methods Symposium. Springer, 2019: 166-185.

[11] Cimatti A, Griggio A, Mover S, et al. Ic3 modulo theories via implicit predicate abstraction [C]//International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, 2014: 46-61.

[12] Bradley A R. Sat-based model checking without unrolling[C]//International Workshop on Verification, Model Checking, and Abstract Interpretation. Springer, 2011: 70-87.

[13] Cimatti A, Griggio A. Software model checking via ic3[C]//International Conference on Computer Aided Verification. Springer, 2012: 277-293.

[14] Eén N, Mishchenko A, Brayton R. Efficient implementation of property directed reachability [C]//2011 Formal Methods in Computer-Aided Design (FMCAD). IEEE, 2011: 125-134.

[15] Bradley A R, Manna Z. Checking safety by inductive generalization of counterexamples to induction[C]//Formal Methods in Computer Aided Design (FMCAD'07). IEEE, 2007: 173-180.

[16] Graf S, Saidi H. Construction of abstract state graphs with pvs[C]//International Conference on Computer Aided Verification. Springer, 1997: 72-83.

[17] Biere A, Cimatti A, Clarke E, et al. Symbolic model checking without bdds[C]//International conference on tools and algorithms for the construction and analysis of systems. Springer, 1999: 193-207.

[18] Tonetta S. Abstract model checking without computing the abstraction[C]//International Symposium on Formal Methods. Springer, 2009: 89-105.

[19] Lee S, Sakallah K A. Unbounded scalable verification based on approximate property-directed reachability and datapath abstraction[C]//International Conference on Computer Aided Verification. Springer, 2014: 849-865.

[20] Gupta A, Yang Z, Ashar P, et al. Sat-based image computation with application in reachability analysis[C]//International Conference on Formal Methods in Computer-Aided Design. Springer, 2000: 391-408.

[21] Barrett C, Tinelli C. Satisfiability modulo theories[M]//Handbook of Model Checking. Springer, 2018: 305-343.

[22] Henzinger T A, Jhala R, Majumdar R, et al. Abstractions from proofs[J]. ACM SIGPLAN Notices, 2004, 39(1): 232-244.

[23] Cimatti A, Griggio A, Schaafsma B, et al. The MathSAT5 SMT Solver[C]//Piterman N, Smolka S. LNCS: volume 7795 Proceedings of TACAS. Springer, 2013.

致　　谢

感谢软件学院贺飞老师所开设的《软件分析与验证》课程，我在该课程中学到了许多完成该课题所必需的基础知识。

感谢实验室杨柳学姐，为我解答了我在研究过程中的许多困惑。

声　明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：杨　淇　日　期：2021年6月15日

附录 A　外文资料的调研阅读报告

Model Checking with IC3+IA and Syntax-Guided Abstraction

A.1　Preliminaries

A.1.1　Model Checking[1]

Bugs are a common problem in hardware or software development. Testing is a common way to check for bugs, but it is not guaranteed that all bugs will be found. In some scenarios, it is critical to ensure *correctness*, and this is where *model checking* can be useful.

Model checking is a technique for verifying behavior correctness of hardware or software systems in an automated manner. The system developer uses formal language to model the verification problem of interest as 1) a *transition system*, which typically involves a set of *initial states* and a state *transition relation*, and 2) *properties* that need to be proven correct, i.e., they always hold during the running of the system.

In this overview, we only focus on model checking problems expressed by first-order logic in the following form. First, we have a set of *state variables* denoted by X, of which a *state* is an assignment to each variable. Then, we have a formula representing initial states denoted by $I(X)$. Here we use the form $\varphi(\cdots)$ to indicate that all free variables occurring in $I(X)$ are from X. We also have a formula representing transition relation denoted by $T(X, X')$, where X' is the "primed" version of X, and a formula representing the property to verify denoted by $P(X)$. We say that the system is *safe* if and only if there does not exist a state sequence $s_1, \cdots, s_n$, where $n \geqslant 1$, such that $s_1 \vDash I(X)$, $(s_i, s'_{i+1}) \vDash T(X, X')$ (for all $1 \leqslant i \leqslant n-1$), and $s_n \vDash \neg P(X)$. We refer to the sequence above as a *counterexample*.

A.1.2　Abstraction of Model Checking[2]

Practically, the state space of the transition system may be quite large, even infinite, making it hard for the model checking algorithm to give a result, which is sometimes referred to as state space explosion. Therefore, it is essential to consider reducing the number of states. One way to cope with this is *abstraction*.

Given model checking problem $\langle X, I(X), T(X, X'), P(X)\rangle$, an *abstract state space* S_{abs}, and a function h that maps a concrete (i.e. original) state to an abstract state, then we can define the abstract model checking problem as follows (existential abstraction[3]):

$$\hat{I}(\hat{s}) : \exists X, h(X) = \hat{s} \wedge I(X),$$

$$\hat{T}(\hat{s}, \hat{s}') : \exists X, X', h(X) = \hat{s} \wedge h(X') = \hat{s}' \wedge T(X, X'),$$

$$\hat{P}(\hat{s}) : \forall X, h(X) = \hat{s} \rightarrow P(X).$$

If the abstract problem $\langle \hat{I}(\hat{s}), \hat{T}(\hat{s}, \hat{s}'), \hat{P}(\hat{s})\rangle$ is safe, then the original problem is safe. However, the inverse does not necessarily holds. When the abstract problem is unsafe, a method called counterexample-guided abstraction refinement (CEGAR)[2] can be performed, which repeatedly checks whether there exists a concrete counterexample corresponding to the abstract one, makes the abstraction finer if such concrete one does not exist (i.e., the abstract one is *spurious*), and checks the new abstract problem again. We do not discuss CEGAR further here for simplicity.

A.2 Property Directed Reachability (IC3)[4-5]

Property directed reachability (PDR), also known as *incremental construction of inductive clauses for indubitable correctness (IC3)*, is an extremely effective model checking algorithm. Given model checking problem $\langle X, I(X), T(X, X'), P(X)\rangle$, IC3 tries to find an *inductive invariant* ($F(X)$ s.t. $I(X) \rightarrow F(X)$, $F(X) \wedge T(X, X') \rightarrow F(X')$, and $F(X) \rightarrow P(X)$) to show that the problem is safe. IC3 maintains a sequence of formulas (also called *frames*) $F_1(X), \cdots, F_n(X)$, where $n \geqslant 1$, such that 1) $F_1(X) = I(X)$, 2) $F_i(X) \rightarrow F_{i+1}(X)$ (for all $1 \leqslant i \leqslant n-1$), 3) $F_i(X) \wedge T(X, X') \rightarrow F_{i+1}(X')$ (for all $1 \leqslant i \leqslant n-1$), and 4) $F_i(X) \rightarrow P(X)$ (for all $1 \leqslant i \leqslant n-1$, i.e. except $i = n$). We use the form $\varphi(X')$ to denote the formula that comes from $\varphi(X)$ with each occurrence of variables in X replaced with the "primed" version. In the sequence, each $F_i(X)$ represents an over-approximation of states reachable in $i-1$ transition steps or less. In the *blocking* phase, we strengthen $F(X)s$ by considering: for any $1 \leqslant i \leqslant n-1$ and formula $c(X)$, if $I(X) \rightarrow c(X)$, and $F_i(X) \wedge c(X) \wedge T(X, X') \rightarrow c(X')$, then we update $F_j(X)$ to $F_j(X) \wedge c(X)$ for each $2 \leqslant j \leqslant i+1$. If $F_n(X)$ cannot be strengthened so that $F_n(X) \rightarrow P(X)$, then a counterexample can be found. If there are two consecutive

$F(X)$s becoming equivalent (i.e. $F_i(X) \leftrightarrow F_{i+1}(X)$ for some $1 \leqslant i \leqslant n-1$) during the algorithm procedure, then we obtain an inductive invariant and thus the problem is safe. Other details of IC3 have been omitted for simplicity.

A.3 IC3 via Implicit Predicate Abstraction (IC3+IA)[6]

A naive way to combine IC3 with abstraction is to directly apply IC3 to the abstract model checking problem. However, existence of quantifier in the abstract problem is sometimes unfriendly to the underlying SMT (satisfiable modulo theories)[7] solver of the model checking algorithm. Before introducing the main idea of IC3+IA, we first introduce the concept of *predicate abstraction* and *implicit abstraction* with model checking problem $\langle X, I(X), T(X, X'), P(X)\rangle$.

A.3.1 Predicate Abstraction (PA)[8]

Given formula (predicate) set $\mathbb{P} = \{P_1(X), \cdots, P_n(X)\}$, we define n abstract state variables $\hat{x}_1, \cdots, \hat{x}_n$, and the abstract function h such that $h(X) = (\hat{x}_1, \cdots, \hat{x}_n)$ if and only if for each i, $\hat{x}_i \leftrightarrow P_i(X)$. IC3+IA additionally requires that $\mathbb{P}$ contains all subpredicates of $I(X)$ and $P(X)$.

A.3.2 Implicit Abstraction (IA)[9]

Given abstract function h and the corresponding abstract problem $\langle \hat{I}(\hat{s}), \hat{T}(\hat{s}, \hat{s}'), \hat{P}(\hat{s})\rangle$, the condition for the abstract problem to be safe can be equivalently transformed as follows, getting rid of the existence quantifiers introduced by definition of abstraction: there does not exist a state sequence s_1, $\overline{s}_1$, s_2, $\overline{s}_2$, $\cdots$, s_n, $\overline{s}_n$, where $n \geqslant 1$, such that $s_1 \vDash I(X)$, $(\overline{s}_i, s'_{i+1}) \vDash T(X, X')$ (for all $1 \leqslant i \leqslant n-1$), $\overline{s}_n \vDash \neg P(X)$, and $h(s_i) = h(\overline{s}_i)$ (for all $1 \leqslant i \leqslant n$).

Here, we define the formula representing the equivalence relation between concrete states with abstraction as $EQ(X, \overline{X}) \leftrightarrow h(X) = h(\overline{X})$. In the predicate abstraction case, $EQ(X, \overline{X}) \leftrightarrow (\bigwedge_i P_i(X) \leftrightarrow P_i(\overline{X}))$. Then, we can replace $h(s_i) = h(\overline{s}_i)$ with $(s_i, \overline{s}_i) \vDash EQ(X, \overline{X})$.

A.3.3 Main Idea of IC3+IA

IC3+IA differs to IC3 mainly in the following aspects:

- All frames are limited to only have subpredicates in $\mathbb{P}$.
- The third condition of frames is replaced with $F_i(X) \wedge EQ(X, \overline{X}) \wedge T(\overline{X}, \overline{X}') \wedge EQ(\overline{X}', X') \rightarrow F_{i+1}(X')$.
- Instead of checking whether $F_i(X) \wedge c(X) \wedge T(X, X') \rightarrow c(X')$ is valid, IC3+IA checks whether $F_i(X) \wedge c(X) \wedge EQ(X, \overline{X}) \wedge T(\overline{X}, \overline{X}') \wedge EQ(\overline{X}', X') \rightarrow c(X')$ is valid.
- If $F_n(X)$ cannot be strengthened so that $F_n(X) \rightarrow P(X)$, then an *abstract* counterexample can be found. At this time, a CEGAR procedure will be executed to check whether the counterexample is spurious.

A.4 Syntax-Guided Abstraction (SA)[10]

Syntax-guided abstraction (SA) captures equality relations between terms in the representation of the model checking problem. Given predicate set $\mathbb{P} = \{P_1(X), \cdots, P_n(X)\}$, and one or more non-boolean term set(s) $T_1, T_2, \cdots, T_m$, s.t. any two terms in the same T_i have the same sort (type), and any two that are not in the same T_i have different sorts. For any concrete state, we can obtain from the state the truth value of each P_i, and the partition of each T_i according to equality of terms under current state. We define the abstraction h so that for any two concrete states X_1, X_2, $h(X_1) = h(X_2)$ holds if and only if the truth values and the partitions of two states are all same.

参考文献

[1] Edmund M Clarke. Model checking. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 54–56. Springer, 1997.

[2] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM (JACM)*, 50 (5):752–794, 2003.

[3] Edmund M Clarke, Orna Grumberg, and David E Long. Model checking and abstraction. *ACM transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, 1994.

[4] Aaron R Bradley. Sat-based model checking without unrolling. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 70–87. Springer, 2011.

[5] Niklas Eén, Alan Mishchenko, and Robert Brayton. Efficient implementation of property directed reachability. In *2011 Formal Methods in Computer-Aided Design (FMCAD)*, pages 125–134. IEEE, 2011.

[6] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. Ic3 modulo theories via implicit predicate abstraction. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 46–61. Springer, 2014.

[7] Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. Springer, 2018.

[8] Susanne Graf and Hassen Saidi. Construction of abstract state graphs with pvs. In *International Conference on Computer Aided Verification*, pages 72–83. Springer, 1997.

[9] Stefano Tonetta. Abstract model checking without computing the abstraction. In *International Symposium on Formal Methods*, pages 89–105. Springer, 2009.

[10] Aman Goel and Karem Sakallah. Model checking of verilog rtl using ic3 with syntax-guided abstraction. In *NASA Formal Methods Symposium*, pages 166–185. Springer, 2019.

综合论文训练记录表

学生姓名	杨淇	学号	2017011471	班级	计 71
论文题目	IC3+IA 模型检测框架中语法制导抽象的设计与实现				

主要内容以及进度安排

我们将在一个支持多种抽象方法的IC3+IA模型检测框架中实现语法制导抽象。

进度安排：

- 2020 年 12 月 – 2021 年 1 月：调研研究现状，理解相关工作。
- 2021 年 2 月 – 2021 年 3 月：完成理论分析，并初步完成代码实现。
- 2021 年 4 月：进行调试和实验，改进代码实现。
- 2021 年 5 月：撰写论文。
- 2021 年 6 月：完成最终答辩。

指导教师签字：王生原

考核组组长签字：喻文建

2021年 1 月 6 日

中期考核意见

工作进展顺利.

考核组组长签字：喻文建

2021 年 4 月 9 日

指导教师评语	杨淇同学的主要工作包括：1）对 IC3+IA 和语法制导抽象进行了理论分析；2）在 IC3+IA 框架中对语法制导抽象进行了代码实现；3）对代码实现进行了实验，并解决了在实验过程中所发现的许多问题。 杨淇同学的研究工作和毕业论文体现出杨淇同学已经具备计算机系本科毕业生应有的理论知识和技能。论文结构完整，叙述清楚，是一篇合格的本科毕业论文。 指导教师签字：王生原 2021 年 6 月 16 日
评阅教师评语	杨淇同学的毕业论文面向硬件代码验证，给出了一种语法制导的模型检测框架。论文清楚全面地叙述了研究背景和主要工作，结构完整，叙述清楚，是一篇合格的本科毕业论文。 评阅教师签字：贺飞 2021 年 6 月 15 日
答辩小组评语	答辩流畅顺利，回答问题正确。 答辩小组组长签字：陈小康 2021年 6 月 11 日

总成绩：B

教学负责人签字：

2021年 6 月 15 日