Computer Science and Application 计算机科学与应用, 2023, 13(10), 1837-1845
Published Online October 2023 in Hans. https://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2023.1310182

Hans汉斯

基于队列的树形加法优化认证器

冯天烁，史美琦，齐　爽，江建国*

辽宁师范大学数学学院，辽宁　大连

收稿日期：2023年9月10日；录用日期：2023年10月9日；发布日期：2023年10月16日

摘　要

目前验证乘法器电路依然是一个极其重要的问题，形式验证给出系统的正确性，但为了增加形式验证结果的可靠性，我们在验证过后加入了认证过程，以检验其结果的正确性。同时运用Amulet工具生成Nullstellensatz (NSS)证明来提高对自动化推理的信心，并由独立的认证器Nuss-Checker进行检查。本文在认证器生成规范多项式过程中，运用基于队列的树形加法优化该认证器，降低了求和过程中形成多项式的项的数目，提高了认证器的检查效率。

关键词

形式验证，认证器，NSS证明，队列，树形加法

Queue-Based Tree Addition Optimization Authenticator

Tianshuo Feng, Meiqi Shi, Shuang Qi, Jianguo Jiang*

School of Mathematics, Liaoning Normal University, Dalian Liaoning

Received: Sep. 10th, 2023; accepted: Oct. 9th, 2023; published: Oct. 16th, 2023

Abstract

At present, the verification of the multiplier circuit is still an extremely important problem. The formal verification gives the correctness of the system, but in order to increase the reliability of the formal verification results, we add the authentication process after the verification to check the correctness of the results. The AMULET tool is also used to generate Nullstellensatz (NSS) proofs to increase confidence in automated reasoning and to be checked by the independent certifier USS Checker. In this paper, the authenticator is optimized by tree addition based on queue, which reduces the number of polynomials and improves the checking efficiency of the authenticator.

*通讯作者。

文章引用: 冯天烁, 史美琦, 齐爽, 江建国. 基于队列的树形加法优化认证器[J]. 计算机科学与应用, 2023, 13(10): 1837-1845. DOI: 10.12677/csa.2023.1310182

Keywords

Formal Verification, Authenticator, NSS Proof, Queue, Tree Addition

1. 引言

基于计算机代数的自动推理技术被广泛的应用于算术电路的形式验证，是门级乘法电路形式化验证的重要组成部分。形式验证的目的是保证系统的正确性，然而为了增加验证过程和实现过程的可靠性，一种常见的技术是生成证明文件，该方法监视整个验证过程并形成证明文件，同时该证明文件可以由独立的认证器进行检查。

形式验证的很多应用可以运用可满足性(SAT)求解，电路被转化为合取范式(CNF)来进行解算，这些结果可以通过产生并检查分辨率证明或子句证明进行验证。而在某些应用中，SAT 求解无法应用，例如算数电路的形式验证，验证乘法器的斜接需要指数大小的分辨率证明，即使很小的乘法器，斜接也会产生非常困难的 SAT 问题[1]，这对于复杂的乘法器体系结构来说明显不适用，因此乘法器电路是 SAT 求解的难点。

目前有效的方法是基于计算机代数，将电路中所有逻辑门以及其规范都用多项式表示，并生成 Gröbner 基进行计算，但乘法器部分，尤其是复杂的末级加法器部分很难用计算机代数进行验证，如今可以将 SAT 和计算机代数结合起来，用简单的纹波进位加法器(RCA)来代替生成和传播(GP)加法器[2]。在预处理之后，通过重写门多项式来减少指定多项式，直到不能减少为止，再通过检查返回值是否为零来判断乘法器电路是否正确。

除了电路验证，计算机代数与 SAT 求解相结合也被用于解决复杂的组合问题，例如，找到矩阵乘法的主方法[3]，计算色数为 5 [4]的小单位距离图，或解决 Williamson 猜想[5]。这些应用都证明了运用代数证明系统进行验证的必要性。在证明复杂度中，主要使用的有两个代数证明系统，多项式演算(PC)和 Nullstellensatz (NSS) [6]，其中由于 PC 不适用于实际的证明检查而引入了可以有效检查的实用代数演算(PAC) [7]。NSS 证明捕获的多项式是否可以表示为多项式集合的线性集合，该证明十分简洁，是由输入多项式和相应的协因子多项式序列组成，进而判断生成多项式是否等同于目标多项式，此外 NSS 证明拥有独立的认证器 Nuss-Checker [8]，对验证结果进行检查。

本文在认证器 Nuss-Checker 中基于队列通过树形加法方式生成多项式，将 NSS 证明中两个连续的多项式相加，并运用队列存储其所得和，同时在队列上反复迭代，总是将两个连续的多项式相加，直到只剩下一个多项式为生成多项式，再与目标多项式相比较，得到检查结果，减少了原本按顺序相加产生的中间的二次型多项式的项，提高了检查的效率。

2. 乘法器验证

计算机代数[9]是目前为门级乘法器做形式验证最有效的技术之一。计算机代数将电路建模成多项式，并将其规范也建模成多项式，这是由电路导出的多项式所表示的，也就是说，电路中的每个门都有其对应的多项式方程，该方程描述了门的输入与输出之间的关系，并将多项式按照与项顺序一致的拓扑顺序

进行排序，这样做能够使门多项式自动生成 Gröbner 基[10]，并运用预处理技术来简化 Gröbner 基。同时，电路实现逻辑功能，运用其实现的逻辑函数计算二进制数字值，输入时给定二进制值，通常由简单布尔函数表示逻辑门，例如 NOT，AND，OR。电路的规格就是电路输入和输出之间的期望关系。若电路的所有输入产生的输出都符合所期望的关系，则证明电路满足其规格，从而得出该电路的正确性。

我们用来验证的工具是 Amulet，它将有符号或无符号的整数乘法器 C 作为输入，电路通常用 And-Inverter-Graphs (AIG) [11]格式表示，考虑门级整数乘法器电路 C 的 $2n$ 个输入变量 $a_0,\cdots,a_{n-1},b_0,\cdots,b_{n-1}\in\{0,1\}$ 和 $2n$ 个输出变量 $s_0,\cdots,s_{2n-1}\in\{0,1\}$。内部门变量由 $g_1,\cdots,g_k\in\{0,1\}$ 表示。设 R 是一个有单位的交换环，并且 $R[a_0,\cdots,a_{n-1},b_0,\cdots,b_{n-1},g_0,\cdots,g_k,s_0,\cdots,s_{2n-1}]=R[X]$。乘法器 C 是正确的当且仅当对于所有可能的输入 $a_i,b_i\in\{0,1\}$ 下面的规范多项式 $L=0$ 成立：

$$L=-\sum_{i=0}^{2n-1}2^i s_i+\left(\sum_{i=0}^{n-1}2^i a_i\right)\left(\sum_{i=0}^{n-1}2^i b_i\right) \tag{1}$$

每一个 AIG 节点都蕴含着一个多项式关系，如图 1。设 $G(C)\subseteq Z[X]$ 为多项式集，其包含对于电路 C 的每个门均有相对应的多项式关系。其中的 v,w 由对应的变量 $x\in X$ 表示，我们称这些多项式为门约束。

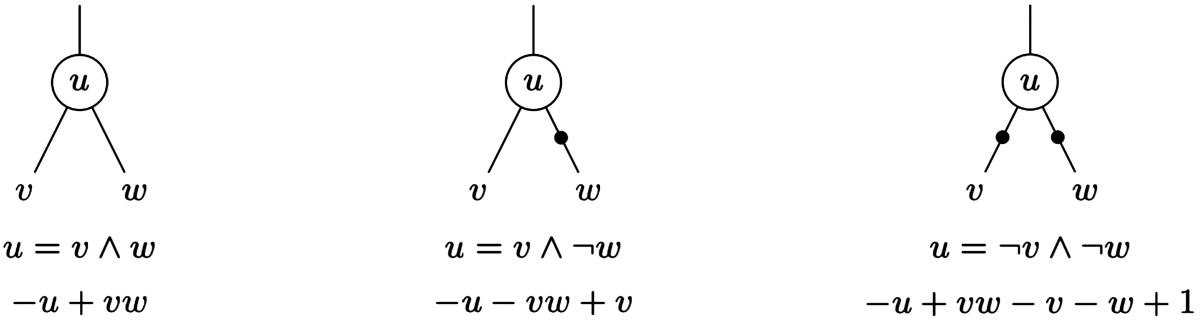

Figure 1. All polynomial encodings covered by AIG nodes
图 1. AIG 节点覆盖的多项式编码

定义 2.1. R 是一个环，X 表示一组变量 $\{x_1,\cdots,x_l\}$。$R[X]$ 表示 X 的系数在 R 中的多项式环。

定义 2.2. 项 $\tau=x_1^{d_1},\cdots,x_l^{d_l}$ 是变量幂的乘积，其中 $d_i\in N$，单项式 $c\tau$ 是项的乘积，其中 $c\in R\setminus\{0\}$，多项式是具有成对不同项的单项式的有限和。

定义 2.3. 在项集 $[X]$ 上的序关系固定所有项 τ,σ_1,σ_2，当 $1\le\tau$ 时，且 $\sigma_1\le\sigma_2\Rightarrow\tau\sigma_1\le\tau\sigma_2$，这种顺序被称为字典序，如果多项式排序为 $x_1>x_2>\cdots>x_l$，对任意两个不同的项 $\sigma_1=x_1^{d_1}\cdots x_l^{d_l}$，$\sigma_2=x_1^{e_1}\cdots x_l^{e_l}$，我们有 $\sigma_1<\sigma_2$ 当且仅当存在变量 i，有 $d_j=e_j$ 时，对于所有的 $j<i$ 有 $d_i<e_i$。我们有 $\sigma_1=\sigma_2$ 当且仅当对于所有 $1\le j\le l$ 有 $d_j=e_j$。

定义 2.4. 对一个多项式 $p=c\tau+\cdots$，其中最大的项 τ 被称为首项 $lt(p)=\tau$，首项系数为 $lc(p)=c$，首项单项式为 $lm(p)=c\tau$，p 的尾部为 $tail(p)=p-lm(p)$。

定义 2.5. 设 $P\subseteq R[X]$，如果对于一个项序，P 所有的前导项都只由一个指数为 1 的变量构成，并且对于所有 $p\in P$，有 $lc(p)\in R^{\times}$，则称 P 有唯一的前导项(UMLT)。

所有变量 $x\in X$ 是布尔值，通过为每个变量添加布尔值约束 $x(x-1)=0$ 来强调，令 $B(Y)=\{y(1-y)\mid y\in Y\}\subseteq R[X]$，其中 $Y\subseteq X$，是 Y 的布尔值约束集合。设 $R=Z$，由于门多项式的 UMLT 性质，对于 $J(C)\subseteq Z[X]$，$G(C)\cup B(X_0)$ 定义了一个 D-Gröbner 基[12]，进一步得知 $J(C)=\langle G(C)\cup B(X)\rangle\subseteq Z[X]$ 即 $J(C)$ 包含 $x\in X$ 所有布尔值约束。因此，可以通过门多项式和所有布

尔约束约简 L，并通过检查结果是否为零来确定电路的正确性。将多项式进行简化的过程为算法 1，该算法隐式处理 $B(X)$，而不是显式地从 $B(X)$ 中减少多项式，同时取消大于 1 比 1 的指数。

算法 1 Reduction (p, p_v, v)

输入：多项式 $p, p_v \in Z[X]$，$lm(p_v) = -v$

输出：多项式 $h, r \in Z[X]$，形如 $p + hp_v = r$

1. $t \leftarrow p, r \leftarrow p, h \leftarrow 0$;
2. **while** $t \neq 0$ **do**
3. **if** $v \in lt(t)$ **then**
4. $h = h + lm(t)/v$;
5. $r = r + p_v lm(t)/t \mod \langle B(X) \rangle$;
6. $t = t - lm(t)$;
7. **return** h, r

然而，乘法器的 GP 加法器部分很难用计算机代数来验证，而加法器电路的等效性检查则很容易由 SAT 求解。因此，将 SAT 和计算机代数相结合，验证工具 Amulet 可以自动将复杂的 GP 加法器替换为简单的纹波进位加法器。用 SAT 求解器可以验证替换的正确性，用计算机代数可以验证重写乘法器的正确性，同时 Amulet 中可以生成 PAC 证明和 NSS 证明。

3. 代数证明系统

已知在乘法器验证期间分为两个阶段，首先在预处理步骤中，从 $G(C)$ 中消除变量来获得更简单的多项式表示 $G(C)'$。其次，通过约简 $G(C)'$ 来确定所给的电路是否正确。这两个阶段都包含在 NSS 证明中，来生成规范多项式 L 作为初始的门多项式 $G(C) \in Z[X]$ 的线性组合。

定义 3.1. 对于一组给定的多项式 $G \subset Z[X]$，令 $\text{base}(r) = \{(p_i, q_i) \mid p_i \in G, q_i \in Z[X]\}$，如果存在多项式 $v_1, \cdots, v_l$，$r = \sum_{(p_i,q_i)\in \text{base}(r)} q_i p_i + \sum_{i=1}^{l} v_i (x_i^2 - x_i)$，则称 base($r$)是关于 G 的基表示。

为了得到 L 的 NSS 证明，需要找到 L 用 $G(C)$ 表示的基表示，对于所有多项式 $g \in G(C)$，则认为 $\text{base}(g) = \{(g, 1)\}$ 是 $G(C)$ 的基表示。由于通过算法 2 导出的重写多项式 r 替换 $G(C)$ 的多项式，以此来重写 $G(C)$，也就是说，可以导出 $G(C)$ 的 r 的基表示，即 base(r)中包含算法 1 中使用的元组 $(p, 1), (p_v, h)$。

算法 2 展示了如何通过添加元组 (f, h) 来更新 base(r)，如果算法 3 的输入多项式 f 是 $G(C)$ 的一个元素，即 $\text{base}(f) = \{(f, 1)\}$，则将 (f, h) 添加到 base(r)中，若 base(r)的元组中没有 f，需将 (f, h) 加到 base(r)的元组中，这样 base(r)的元组 (f, h_i) 则更新为 $(f, h_i + h)$，这是 base(r)中的合并公因式。如果多项式 f 不是初始门多项式，对于某些初始多项式 f_i 和 $h_i' \in Z[X]$，f 可以写成线性组合 $f = h_1' f_1 + \cdots + h_l' f_l$。因此元组 (f, h) 对应为 $hf = hh_1' f_1 + \cdots + hh_l' f_l$，我们遍历元组 $(f_i, h_i') \in \text{base}(g)$，将每个协因子 h_i' 乘以 h，并将对应的元组 (f_i, hh_i) 添加到 base(r)中。

预处理完成后重复应用算法 2，将规范多项式 L 约简 $G(C)$，通过推导 L 的基表示来生成 NSS 证明。因此在每次约简之后都要将元组 (g, h) 添加到 base(L)中，其中 h 是多项式 g 相应的协因子。在完成最后的约简之后，base(L)就表示 NSS 证明并将其打印到文件中。

NSS 证明系统推导出是否多项式 $f \in R[X]$ 可以表示为给定集合 $G = \{g_1, \cdots, g_m\}$，也就是说，对于给定多项式 f 和集合 $G = \{g_1, \cdots, g_m\}$ 是数组 $P = (h_1, \cdots, h_m)$ 的多项式，例如

$$\sum_{i=1}^{m} h_i g_i = f \tag{2}$$

NSS 证明的健全性和完备性论证都可以推广到环 $R[X]$，当 G 具有 UMLT 时，在 NSS 证明中，布尔值约束被隐式处理以产生更短的证明。因此对于给定多项式 $f \in R[X]$ 和多项式集 $G=\{g_1,\cdots,g_m\} \subseteq R[X]$ 是一组协因子 $P=(h_1,\cdots,h_m)$ 的多项式，使得存在多项式 $r_1,\cdots,r_l \in R[X]$，有

$$\sum_{i=1}^{m} h_i g_i + \sum_{i=1}^{l} r_i \left(x_i^2 - x_i\right) = f \tag{3}$$

算法 2 Add-to-basis-representation $(f, h, \text{base}(r))$

输入：多项式 $f, h \in Z[X]$，基表示 base(r)

输出：更新后的 base(r)，内含 (f, h)

1.**if** $\text{base}(f)=\{(f,1)\}$ **then**

2.**if** $(f, h_i) \in \text{base}(r)$ for any h_i **then**

3. $\text{base}(r) \leftarrow \left(\text{base}(r) \setminus \{(f, h_i)\}\right) \cup \{(f, h_i + h)\}$;

4.**else**

5. $\text{base}(r) \leftarrow \text{base}(r) \cup \{(f, h)\}$;

6.**else**

7.**foreach** $(f_i', h_i') \in \text{base}(f)$ **do**

8. base(r) ← Add-to-basis-representation $(f_i', hh_i', \text{base}(r))$

9.**return** base(r)

4. 检查器及优化

4.1. NSS 认证器

NSS 认证器 Nuss-Checker 是运用 C 语言实现的。Nuss-Checker 需要读入三个输出文件<constraints>，<cofact>和<target>，文件<constraints>包含初始约束 $g_i \in G$，<cofact>按同样的顺序包含对应的协因子 h_i。Nuss-Checker 读取文件<constraints>和<cofact>生成乘积，并将乘积进行求和，验证所得的多项式和 f 是否与文件<target>中所给的多项式 L 相等。

认证器通过检查是否 $\sum_{i=1}^{l} h_i g_i = f \in Z[X]$，$p_i \in G \subseteq Z[X]$，$f, h_i \in Z[X]$ 来判断 NSS 证明的正确性。理论上来讲，只需要将初始门约束 g_i 乘以协因子 h_i，然后计算乘积的和。这听起来是很简单的，但是根据实现的不同，分配的时间和最大内存量是会出现数量级变化。

Nuss-Checker 中的多项式在内部存储为单项式的有序链表，系数则用 GMP 库表示，项是变量的有序链表，内部使用散列表来分配项。在运行过程中，Nuss-Checker 动态的生成乘积 $h_i g_i$，也就是说要同时解析文件<constraints>和<cofact>，并从两个文件中分别读取出两个多项式 g_i 和 h_i，并计算 $h_i g_i$。两个多项式的相加是通过将这两个多项式的单项式推入堆栈中实现的，用固定项顺序在堆栈上进行排序，将拥有相同项的单项式进行合并来生成最终的和式，乘法也是运用的类似的方法。在计算多项式求和的过程中，$Z[X]$ 中的加法方式是结合式的，将 l 个乘积 $h_i g_i$ 的所有单项式都推到一个大堆栈中，之后对堆栈上的单项式进行排序和合并，所有的乘积的单项式被推到堆栈上进行存储以及排序和合并，这都增加了 Nuss-Checker 的使用时间。同时按顺序相加，只在内存中存储一个多项式，总是添加最新的

乘积 $h_i g_i$，但目标多项式 L 包含着 n^2 个多项式的中间项 $a_i b_j$，这使得中间和的指数大小为二次，也增加了验证的时间。

4.2. 队列及树状加法

队列(queue)是一种线性结构，与同为线性数据结构的栈不同的是，队列中的元素遵循的是先入先出规则，当将数据存入队列称为“addQueue”，需要两个步骤：

1. 将尾指针往后移：rear + 1，当 front = rear 时为空。

2. 若尾指针 rear 小于队列的最大下标 MaxSize−1，则将数据存入 rear 所指的数组元素中，否则无法存入数据，rear = maxsize − 1，则队列满。

如图 2 所示，在初始化队列时，令 front = rear = 0，当有新元素插入队尾时，尾指针增 1；当元素需从队头出队时，头指针增 1。因此在非空队列中，头指针始终指向队列头元素，而尾指针始终指向队尾元素的下一个位置。在本文中由于需要将每相邻的两个多项式的和作为新元素重新放入存储空间中等待下一次相加，所以选择使用队列对检查器的多项式相加过程进行优化。

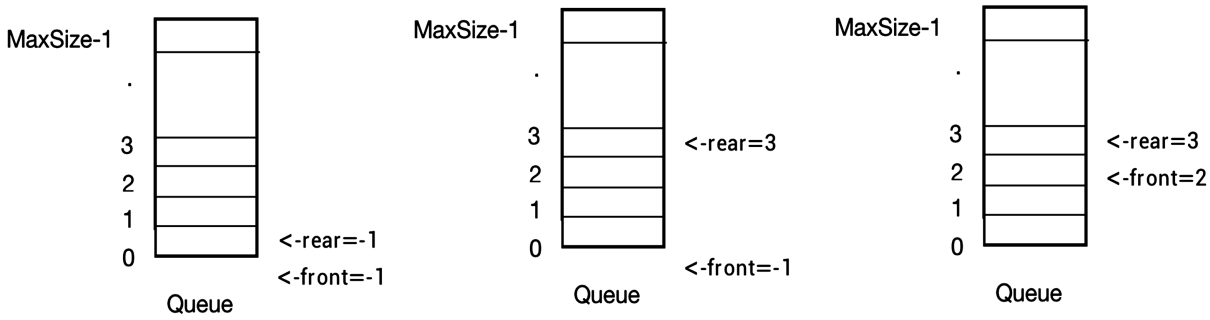

Figure 2. All polynomial encodings covered by AIG nodes
图 2. 队列运行示意图

树形加法是通过在树结构中相加，将 NSS 证明的两个连续的乘积结果相加并加以存储，如图 3 所示。这样在一次遍历相加之后，在队列中会存在 $\frac{l}{2}$ 个多项式，经过在队列中反复迭代，总是对两个连续的多项式求和，并把新的结果推送到队尾，直到最后只剩下一个多项式，这样减少了指数为二次的中间和，因此本文采用树形加法对队列上的多项式求和进行优化。

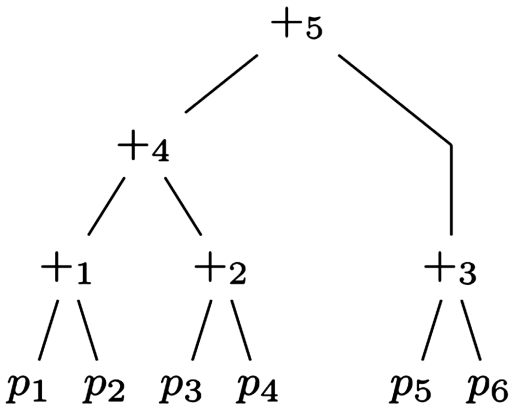

Figure 3. All polynomial encodings covered by AIG nodes
图 3. 树状结构的加法示意图

4.3. 优化算法

基于队列的树形加法优化是将堆栈上存储的多项式 $p_1, p_2, \cdots, p_l$ 重新推送到队列内进行存储，同时运用树状加法将 p_1 与 p_2 相加，p_3 与 p_4 相加，直至 p_{l-1} 与 p_l 相加，相加所得和推送至队尾入列。此时队列内存储了 $\frac{l}{2}$ 个多项式，之后重复该步骤进行反复迭代，直至队列中只剩下一个多项式，即为生成多项式 f，则求和过程结束，随之便可将所得的生成多项式与目标多项式进行对比得出结论。

如下所展示的算法给出了优化的具体过程：(1)初始化一个队列，检查队列是否为空，是否已满；(2)入队，将 l 个多项式 $p_1, p_2, \cdots, p_l$ 存储进队列；(3)获取队列长度，并获取队列前端元素进行打印；(4)取整个队列长度的一半，利用 for 循环，将相邻多项式分别赋值给 p, q，并将 p, q 相加，随后删除 p, q；(5)将相加所得的和添加至队尾重新入列；(6)重复前面步骤进行反复迭代；(7)判断队列长度是否为 1，若为 1，则将唯一的多项式输出。

算法 3 Queue-tree $(h_i g_i, f)$

输入：Nuss-Checker 动态生成的乘积 $h_i g_i$
输出：Nuss-Checker 生成的多项式 f

```
1.initialQueue   front=-1，rear =-1
2.isEmpty   front=-1
3.isFull   rear=MAX_SIZE=-1
4.while   getSize(queue>1)
5.for (i=0；   i<getSize(queue)；   i++)
6.            p= dequeue(queue)
7.            q = dequeue(queue)
8.            add = add_polynomials(p ，  q)
9.       enqueue(queue ，  add)
10.getSize(queue) = 1
11.return   dequeue(queue)
```

基于队列的树形加法优化使得在最终求和的过程中，面对庞大、繁杂的单项式与多项式，总是能尽快的添加同一层的两个多项式来动态的运用树形加法，同时结合队列的灵活性，减少其相加求和时产生的中间项的次数，简化运算大小，更加方便、快捷、高效的完成认证器的检查过程，提高了认证器的检查效率。

5. 实验

本文中的实验使用的是一台带有 Ubuntu 18.04.6 虚拟机的电脑，配置为 Intel i5-6300HQ，2.30GHz 和 2GB 内存，实验用的工具是 Amult 和 Nuss-checker。选择了 btor 乘法器[13]和 sp-ar-rc 乘法器[14]电路进行验证。这两类的乘法器结构略有不同，btor 乘法器是将全加器和半加器用网状结构累积，sp-ar-rc 乘法器是将全加器和半加器用对角线结构累积。如图 4、图 5 所示，是 4 位 btor 乘法器和 4 位 sp-ar-rc 乘法器的结构示意图。

实验时间表示从运行代码开始，到最后得出该乘法器是否为正确的乘法器结论为止所花费的时长，单位为秒，时间限制为 600 秒。内存的单位为 MB，内存限制为 2048 MB。实验结果如表 1 所示。

实验将按顺序相加所得多项式的时间与基于队列的树形加法所得的多项式的时间相对比，可以显著得知，基于同类型同位数的乘法器，相比于顺序相加，优化后所需的时间有了明显减少。随着乘法器复杂度的提升，这种差异会越来越明显。

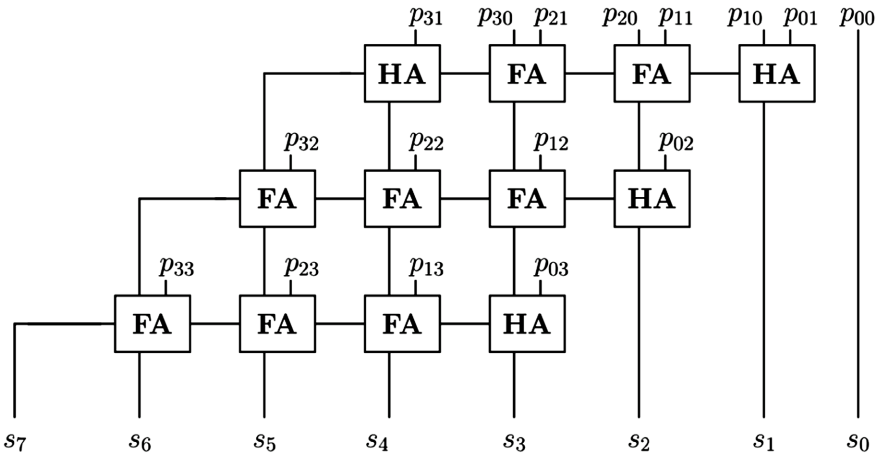

Figure 4. 4-bit btor multiplier structure
图 4. 4 位 btor 乘法器结构

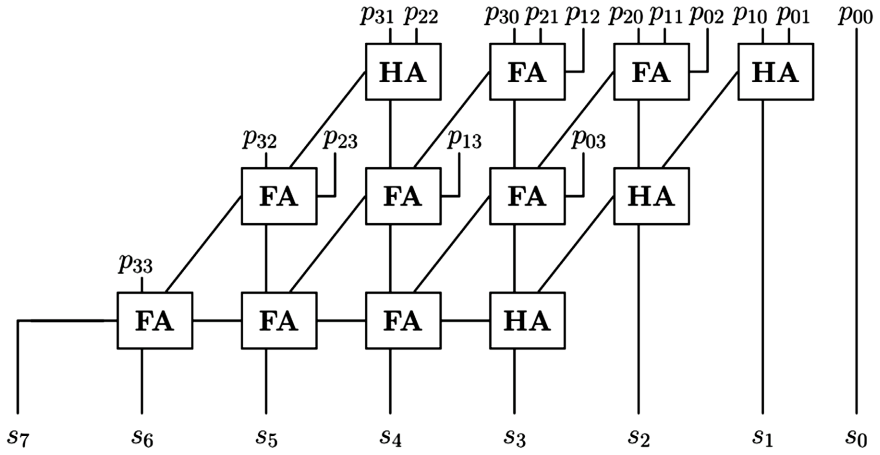

Figure 5. 4-bit sp-ar-rc multiplier structure
图 5. 4 位 sp-ar-rc 乘法器结构

Table 1. Resulting data of experiment
表 1. 试验结果数据

乘法器类型	位数	Nuss-Checker		Queue-tree	
		时间/s	内存/MB	时间/s	内存/MB
botr	16	0.071	2.5	0.030	2.5
botr	32	0.755	2.8	0.102	2.8
botr	64	13.284	5.7	0.447	5.9
btor	128	338.629	18.6	3.390	18.8
sp-ar-rc	16	0.115	2.5	0.042	2.5
sp-ar-rc	32	1.357	3.2	0.173	3.4
sp-ar-rc	64	20.707	7.4	0.787	7.8
sp-ar-rc	128	339.583	24.5	3.681	25.1

6. 结论

本文对验证 NSS 证明时 NSS 认证器检验其生成多项式的过程进行优化。原本需要不断添加最新的乘积 $h_i g_i$ 到多项式中，但因为目标多项式 L 包含 n^2 个多项式的中间项，导致中间和的指数大小为二次，而基于队列的树形加法改变了多项式的求和方式，将两个连续的乘积相加并求和存储，这样不会产生有 n^2 个多项式的中间项的情况，使得认证器的检查时间有了明显的减少，提高了认证器的检验效率。未来的工作中，希望该算法方式也会运用于不同的应用程序。

参考文献

[1] Kaufmann, D., Biere, A. and Kauers, M. (2019) Verifying Large Multipliers by Combining SAT and Computer Algebra. 2019 *Formal Methods in Computer Aided Design* (*FMCAD*), San Jose, 22-25 October 2019, 28-36. https://doi.org/10.23919/FMCAD.2019.8894250

[2] Parhami, B. (2000) Computer Arithmetic—Algorithms and Hardware Designs. Oxford University Press, Oxford.

[3] Heule, M.J.H., Kauers, M. and Seidl, M. (2019) Local Search for Fast Matrix Multiplication. In: Janota, M. and Lynce, I., Eds., *Theory and Applications of Satisfiability Testing*, Vol. 11628, Springer, Cham, Berlin, 155-163. https://doi.org/10.1007/978-3-030-24258-9_10

[4] Bright, C., Kotsireas, I. and Ganesh, V. (2019) SAT Solvers and Computer Algebra Systems: A Powerful Combination for Mathematics.

[5] Bright, C., Kotsireas, I. and Ganesh, V. (2019) Applying Computer Algebra Systems and SAT Solvers to the Williamson Conjecture. *Journal of Symbolic Computation*, **100**, 187-209. https://doi.org/10.1016/j.jsc.2019.07.024

[6] Beame, P., Impagliazzo, R., Krajíček, J., Pitassi, T. and Pudlák, P. (1996) Lower Bounds on Hilbert's Nullstellensatz and Propositional Proofs. *The Proceedings of the London Mathematical Society*, **s3-73**, 1-26. https://doi.org/10.1112/plms/s3-73.1.1

[7] Ritirc, D., Biere, A. and Kauers, M. (2018) A Practical Polynomial Calculus for Arithmetic Circuit Verification. CEUR-WS, 61-76.

[8] Kaufmann, D. (2021) Nullstellensatz-Proofs for Multiplier Verification. http://fmv.jku.at/nussproofs

[9] Mahzoon, A., Große, D., Scholl, C. and Drechsler, R. (2020) Towards Formal Verification of Optimized and Industrial Multipliers. 2020 *Design, Automation & Test in Europe Conference & Exhibition*, Grenoble, 9-13 March 2020, 544-549. https://doi.org/10.23919/DATE48585.2020.9116485

[10] Buchberger, B. (1965) Ein Algorithmus zum Auffifinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. Ph.D. Thesis, University of Innsbruck, Innsbruck.

[11] Kuehlmann, A., Paruthi, V., Krohm, F. and Ganai, M. (2002) Robust Boolean Reasoning for Equivalence Checking and Functional Property Verifification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **21**, 1377-1394. https://doi.org/10.1109/TCAD.2002.804386

[12] Lichtblau, D. (2012) Effective Computation of Strong Gröbner Bases over Euclidean Domains. *Illinois Journal of Mathematics*, **56**, 177-194. https://doi.org/10.1215/ijm/1380287466

[13] Niemetz, A., Preiner, M., Wolf, C. and Biere, A. (2018) BTOR$_2$, BtorMC and Boolector 3.0. In: Chockler, H. and Weissenbacher, G., Eds., *Lecture Notes in Computer Science*, Vol. 10981, Springer, Berlin, 587-595. https://doi.org/10.1007/978-3-319-96145-3_32

[14] Homma, N., Watanabe, Y., Aoki, T. and Higuchi. T. (2006) Formal Design of Arithmetic Circuits Based on Arithmetic Description Language. *IEICE Transactions*, **E89-A**, 3500-3509. https://doi.org/10.1093/ietfec/e89-a.12.3500