Advances in Applied Mathematics 应用数学进展, 2021, 10(11), 3819-3827
Published Online November 2021 in Hans. http://www.hanspub.org/journal/aam
https://doi.org/10.12677/aam.2021.1011405

Hans汉斯

基于动态数组的加法器重写优化算法

张小盈*，吕妍颖，江建国

辽宁师范大学数学学院，辽宁 大连

收稿日期：2021年10月15日；录用日期：2021年11月5日；发布日期：2021年11月16日

摘　要

乘法器电路验证是算术电路验证领域内的一个重大难题。当前最有效的代数验证方法是Gröbner基方法。基于此方法提出的加法器重写算法是一项重大创新，但识别加法器的过程需要穷举遍历电路变量，并且十分低效。为了解决这一问题，本文对加法器重写算法进行了优化，使用动态数组存储搜索加法器所需的门变量，并以逆拓扑序的顺序来遍历，从而消除了冗余。实验结果表明，结合动态数组来识别加法器能够有效提高Gröbner基的生成效率和验证速度。

关键词

乘法器电路，动态数组，Gröbner基，加法器重写

Adder Rewriting Optimization Algorithm Based on Dynamic Array

Xiaoying Zhang*, Yanying Lv, Jianguo Jiang

School of Mathematics, Liaoning Normal University, Dalian Liaoning

Received: Oct. 15th, 2021; accepted: Nov. 5th, 2021; published: Nov. 16th, 2021

Abstract

Verification of multiplier circuits is an important problem in the field of arithmetic circuit verification. Currently, the most effective algebraic verification method is the Gröbner basis method. The adder rewriting algorithm proposed based on Gröbner basis method is a major innovation, but the process of identifying requires exhaustive traversal circuit variables and is very inefficient.

*通讯作者。

文章引用: 张小盈, 吕妍颖, 江建国. 基于动态数组的加法器重写优化算法[J]. 应用数学进展, 2021, 10(11): 3819-3827.
DOI: 10.12677/aam.2021.1011405

To solve this problem, this paper optimizes the adder rewriting algorithm, using a dynamic array to store the gate variables required to search the adders and traverse in reverse topological order, which eliminates redundancy. Experimental results show that combining the dynamic array to identify the adders can effectively improve the generation efficiency of Gröbner basis and the verification efficiency.

Keywords

Multiplier Circuit, Dynamic Array, Gröbner Basis, Adder Rewriting

1. 引言

乘法器作为一种重要的算术电路，其正确性的验证始终是一个挑战。目前用于乘法器电路验证的最主要的方法是代数方法[1]-[7]。该方法将乘法器的规范以及内部的逻辑门建模为多项式集，把乘法器验证问题转化成代数问题来解决。

代数方法包括两类：函数抽取[2]和 Gröbner 基法[3] [4] [5] [6] [7]。函数抽取是将乘法器的输出多项式通过代换和消元，最终化简为输入多项式，之后与电路规范进行比较。Gröbner 基法是将“乘法器电路验证问题”转化为“理想成员问题”，直接将电路建模为一组 Gröbner 基，再验证电路规范是否属于 Gröbner 基生成的理想。事实上，这两种方法都能够验证架构简单的大型乘法器，并且函数抽取和 Gröbner 基法是等价的[2]。

Gröbner 基法是代数方法中最为有效的验证技术，然而随着乘法器位数的增大，Gröbner 基中多项式数目会逐渐增多；同时，在验证过程中，每一步归约得到的余式中单项式数目呈指数性增长。为了减少归约产生的中间单项式的数目，研究者们提出了多种 Gröbner 基优化策略，包括逻辑归约重写方案[3]、逐列验证方法[4]、加法器重写[5]、末级加法器替换[6]、动态替换序[7]等。这些方法都能够在一定程度上减少中间单项式的数目，进而提高验证速度，但无法解决架构复杂的优化的乘法器电路验证问题。

加法器重写[5]的基本思想是利用电路中的进位变量在乘法器电路的门级表示中识别全加器和半加器，并将它们视为单独的电路。再利用变量消去，用加法器的规范多项式替换内部门的逻辑多项式，以简化 Gröbner 基，从而提高验证速度。事实上，加法器重写不仅能应用于逐列验证方法，它还可以应用到乘法器电路的其它验证方法，甚至应用于任何一个包含加法器的算术电路验证中，具有十分重要的意义。然而，加法器重写中的加法器识别速度较慢，随着电路位宽的增加，会导致遍历的变量数目成倍增长，存在很明显的弊端。

本文在加法器重写的基础上，使用动态数组存储电路中所有的和位输出变量，直接利用乘法器的结构性知识，以逆拓扑序遍历数组中的变量来识别全加器和半加器，并且不依赖进位变量。这种设计具有以下几点优点：第一，使用动态数组存储所需变量能够在很大程度上减少变量的遍历数目；第二，使用逆拓扑序遍历变量能够有效地避免重复和冗余；第三，不依赖进位变量识别加法器，能够避免重复遍历。实验结果表明，结合动态数组的加法器重写优化算法在很大程度上提高了加法器的识别速度，进而提高了验证效率。

2. Gröbner 基方法

2.1. 理论基础

本节简单介绍了 Gröbner 基方法的相关理论，更详细的资料请参见[8]。

设 k 是一个数域，$k[x_1,\cdots,x_n]$ 是数域 k 上关于变量 $x_1,\cdots,x_n$ 的 n 元多项式环。$k[x_1,\cdots,x_n]$ 上可以定义单项式序，在这种序关系的约束下，每个多项式都能够按唯一的顺序排列各项，使得在多项式除法算法中所得的余式唯一。

定义 2.1 设 I 是 $k[x_1,\cdots,x_n]$ 的一个非空子集，满足：

1) $0\in I$；
2) 对任意的多项式 $f,g\in I$，都有 $f+g\in I$；
3) 对任意的多项式 $f\in I$，$p\in k[x_1,\cdots,x_n]$ 都有 $pf\in I$。

则称 I 是一个多项式理想。

定义 2.2 设 $f_1,\cdots,f_s\in k[x_1,\cdots,x_n]$ 是多项式，令

$$I=\langle f_1,\cdots,f_s\rangle=\left\{\sum_{i=1}^{s}h_i f_i \mid h_i\in k[x_1,\cdots,x_n]\right\} \tag{1}$$

则 I 是 $k[x_1,\cdots,x_n]$ 的一个多项式理想，称 $\{f_1,\cdots,f_s\}$ 是 I 的一组基。

任一理想可以有很多组基，但在这些基中存在一种具有特定性质的 Gröbner 基：

定义 2.3 设 I 是 $k[x_1,\cdots,x_n]$ 的多项式理想，如果有 $\langle lt(g_1),\cdots,lt(g_m)\rangle=\langle lt(t)\rangle$ 成立，其中 $lt(g_i)$ 表示 g_i 的首项，则称集合 $\{g_1,\cdots,g_m\}$ 是理想 I 关于特定序关系<的一组 Gröbner 基。

接下来引入本文的理论基础，即理想成员算法，这是使用 Gröbner 基理论来验证乘法器电路的关键：

定理 2.1 (理想成员算法) 设 G 是理想 I 的一组 Gröbner 基，多项式 $q\in k[x_1,\cdots,x_n]$，q 除以 G 所得的余式 r 满足 $q-r\in I$，当且仅当 $r=0$ 时，$q\in I$。

2.2. 代数模型

使用 Gröbner 基方法来验证乘法器电路的正确性，首先需要将电路建模为多项式集。假设具有 $2n$ 个输入变量 $a_0,\cdots,a_{n-1},b_0,\cdots,b_{n-1}$ 和 $2n$ 个输出变量 $s_0,\cdots,s_{2n-1}$ 的乘法器 C 是一个无回路电路，其中每个内部门都由变量 $g_0,\cdots,g_{l-1}$ 表示，所有变量均为布尔变量。定义字典序 $<$ 下的变量为

$X=a_0,\cdots,a_{n-1},b_0,\cdots,b_{n-1},g_0,\cdots,g_{l-1},s_0,\cdots,s_{2n-1}$。

将电路中每个逻辑门的输出与输入之间的关系表示为布尔多项式，常见的逻辑门多项式如下：

$$\begin{aligned}
&u=\neg v \Leftrightarrow -u+1-v=0\\
&u=v\wedge w \Leftrightarrow -u+vw=0\\
&u=v\vee w \Leftrightarrow -u+v+w-vw=0\\
&u=v\oplus w \Leftrightarrow -u+v+w-2vw=0
\end{aligned} \tag{2}$$

由于布尔变量的取值均为 0 或 1，故对于所有变量 u 都有 $u(u-1)=0$，形如 $-u^2+u$ 的多项式称为域多项式。所有门变量逻辑多项式(1)和域多项式共同构成了电路的多项式集合 $G\subseteq k[x_1,\cdots,x_n]$，由 G 生成的理想记为 $J(C)$，根据定义 2.3，G 是 $J(C)$ 在序关系 $<$ 下的一组 Gröbner 基。

定义 2.4 设 C 是一个电路，$p\in k[x_1,\cdots,x_n]$ 是多项式。将 C 中所有输入变量、内部门变量和输出变量的任一组给定值代入到 p 中，若结果为 0，则称 p 是电路 C 的一个多项式电路约束，记为 PCC，电路 C 中所有 PCC 构成的集合记为 $I(C)$。

定理 2.2 对于所有的无回路电路 C，都有 $I(C)=J(C)$。

定义 2.5 设 C 是一个电路，如果多项式

$$\sum_{i=0}^{2n-1}2^i s_i-\left(\sum_{i=0}^{n-1}2^i a_i\right)\left(\sum_{i=0}^{n-1}2^i b_i\right) \tag{3}$$

是一个 *PCC*，则称电路 C 是一个乘法器电路，称该多项式为 C 的规范多项式 *sp*。

根据上述定义和定理，可以得到如下结论：

为了验证乘法器电路的正确性，只需使用定理 2.1 来验证 *sp* 是否属于理想 $I(C)=J(C)$。若属于，则 C 是正确的；否则 C 是错误的。

3. 结合动态数组的优化

3.1. 加法器重写

Ritirc 等人在[5]中首次提出了加法器重写。其基本思想是利用电路中的进位变量在乘法器电路的门级表示中以拓扑序识别全加器和半加器，并将它们视为单独的电路。再利用变量消去，用加法器的规范多项式替换 Gröbner 基中对应的内部门的逻辑多项式，以简化整个乘法器电路的 Gröbner 基，从而提高验证速度。

算法 1：加法器重写算法

输入：AIG 格式的乘法器电路 C

输出：可供计算机代数系统计算的输出文件 file

```
1. used();
2. for k ← 0 to 2n−1 do
3.     S ← slice + i
4.     for j ← 0 to S->order − 1 do
5.         if var->used != var->slice then
6.             var->carry = 1
7.         end if
8.     end for
9. end for
10. if full_adder_rewriting = 1 then
11.     fulladder_reduction();
12. end if
13. print_adder();
14. file ← print_spec ;
```

算法 1 描述了加法器重写的过程。此算法以乘法器电路的 AIG 格式[9]作为输入，主要包括以下几个步骤：1) 遍历电路中所有变量，并使用 used 属性表示变量出现的最大切片索引；2) 判断所有变量的 used 值与 slice 值的关系，当二者不同时，将该变量标记为进位变量；3) 利用这些进位变量以拓扑序识别电路中的加法器，并给出相应的规范多项式；4) 用加法器的规范多项式替换电路 Gröbner 基中相应的逻辑门多项式，以简化 Gröbner 基；5) 将生成的 Gröbner 基写入文件 file，再传递给计算机代数系统 Mathematica

或 Singular 进行计算。

我们以全加器(图 1 左侧)为例解释加法器重写中的替换过程。在使用加法器重写之前，全加器电路 A 在逆拓扑序 $a<b<i<g_0<g_1<g_2<s<c$ 下的一组 Gröbner 基如下所示：

$$\begin{aligned} G_A = \{&-c+g_1+g_2-g_1g_2, -s+g_0+i-2g_0i, -g_2+ab, -g_1+g_0i, -g_0+a+b-2ab,\\ &-c^2+c, -s^2+s, -g_2^2+g_2, -g_1^2+g_1, -g_0^2+g_0, -i^2+i, -b^2+b, -a^2+a\} \end{aligned} \tag{4}$$

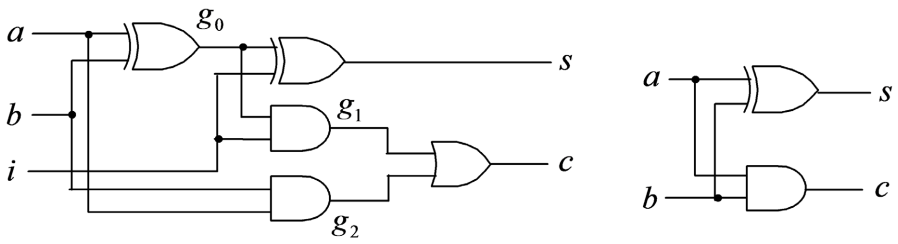
使用加法器重写可以消去内部门变量 g_0, g_1, g_2 和所有的域多项式，使用规范多项式来代替这些门多项式。重写后得到的 Gröbner 基如下：

$$G_A' = \{-2c-s+a+b+i, -s+a+b+i-2ab-2ai-2bi+4abi\} \tag{5}$$

同理，在逆拓扑序 $a<b<s<c$ 下，对于半加器电路 B (图 1 右侧)使用加法器重写前后的 Gröbner 基如下所示：

$$\begin{aligned} G_B &= \{-c+ab, -s+b+a-2ab, -c^2+c, -s^2+s, -b^2+b, -a^2+a\}\\ G_B' &= \{-2c-s+a+b, -s+b+a-2ab\} \end{aligned} \tag{6}$$

可以看到，优化后全加器电路的 Gröbner 基中多项式数目由 13 减少到 2，半加器电路的 Gröbner 基中多项式数目由 6 减少到 2，由此可以说明，加法器重写在很大程度上简化了 Gröbner 基。



(a) 全加器电路图　　(b) 半加器电路图

Figure 1. Full-adder (left) and half-adder circuit (right)
图 1. 全加器(左)与半加器(右)电路图

然而，在此算法中，加法器的识别过程比较复杂，还需要做出进一步优化。首先，此算法需要以拓扑序两次遍历电路中的所有变量来标记进位变量，一方面，随着乘法器电路位宽的增加，遍历的变量数目会越来越多；另一方面，两次遍历导致消耗的时间过长。此外，以拓扑序遍历乘法器电路变量，没有充分考虑到乘法器电路的结构特点，可能会导致重复遍历。

3.2. 动态数组

本文的工作首先要选择合适的数据结构来存储变量。目前能够处理大量元素的数据结构包括链表、动态数组和哈希表等。其中哈希表是用于存储键值数据的，不适用于本文研究的问题；而链表只适用于元素很少且只需要从两端添加和删除数据的情况，当处理大量元素时，使用链表无法便捷地访问到任一元素。

动态数组是一种长度可变的数组，具有占用空间少、运行速度快、易于操作大量数据等特点，它拥有链表的大部分功能[10]。动态数组最大的优势就是可以随意地在尾部添加或者删除元素，并且能够访问和遍历到任意元素，从而减少代码的运行时间和空间。但如果使用链表来访问某一元素，就需要遍历它前面所有的元素，消耗相对较长的时间。考虑到大型乘法器中存在许多变量，本文选择使用动态数组来

优化加法器重写算法。

3.3. 优化算法

优化算法是使用动态数组存储乘法器电路中所有的和位输出变量 $s_0,\cdots,s_{2n-1}$，再利用电路逻辑门中输出与输入的关系，以逆拓扑序遍历动态数组中的变量来查找全加器和半加器，并动态地增加数组中的元素，直到查找结束。这种存储和遍历方式能够缩短加法器识别所用的时间，提高验证速度。

算法 2：加法器重写优化算法

输入：AIG 格式的乘法器电路 C

输出：可供计算机代数系统计算的输出文件 file

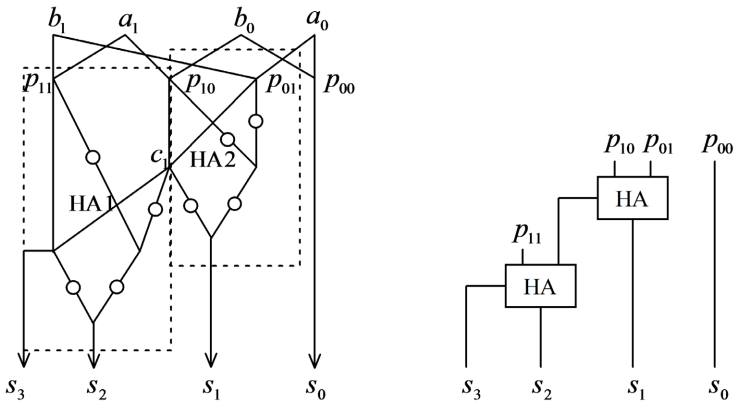
```
1. for  i ← 0  to  2n − 1  do
2.      dynamic_vars[i] ← s_{2n−1−i}
3. end for
4. for  i ← 0 to  end  do
5.      dy_adder_reduction();
6.      if  var->fhadder = 1  then
5.          dynamic_vars[end++] ← var
8.      end if
9. end for
10.  file ← print_spec ;
```

算法 2 给出了使用动态数组的加法器重写优化算法。此算法以动态数组 dynamic_vars 存储加法器重写所需的变量，主要包括以下几个步骤：1) 将和位输出变量 $s_0,\cdots,s_{2n-1}$ 以逆拓扑序的方式添加到动态数组中；2) 使用 for 循环遍历动态数组中的每一个元素以搜索全加器和半加器，并将其对应的规范多项式添加到 Gröbner 基中；3) 将搜索到的加法器对应的输入变量添加在动态数组尾部，同时数组长度随之增加，以动态地增加循环的次数，而乘法器电路门变量的有限性保证了循环过程最终得以终止；4) 将得到的 Gröbner 基写入到输出文件中，以便提供给计算机代数系统进行计算。若计算结果为零，则乘法器是正确的，否则是错误的。

相比于原本的加法器重写算法，优化算法不需要借助进位变量来识别加法器，它直接根据加法器的结构性知识进行识别。同时，优化算法使用动态数组来存储加法器识别所需的变量，减少了遍历和循环所需的时间和空间。为了识别电路中全部的加法器，原算法是以拓扑序遍历所有的进位变量，这种识别方式可能会出现重复遍历的问题，并且逻辑关系比较混乱。而优化算法选择逆拓扑序，从和位输出变量开始，自下而上遍历相应的变量，不会出现重复遍历的情况，且符合逻辑顺序，使整个识别过程更加清晰。

本文以二位 btor 乘法器为例来描述优化算法的运行过程。图 2 右侧给出了乘法器的结构图，其 PPA 阶段只由两个半加器构成。验证工具读取电路对应的 AIG 文件，识别出所有的和位输出 s_0,s_1,s_2,s_3 并将其添加在动态数组 dynamic_vars 中，以逆拓扑序依次遍历数组中的元素，首先使用 s_3 无法识别出加法器，再进一步遍历 s_2。利用 s_2 识别出半加器 HA1，将其对应加法器的规范多项式添加到 Gröbner 基中，并将 HA1 的输入变量 p_{11},c_1 添加到动态数组的尾部，同时标记 s_3 为加法器内部变量；再根据 s_1 识别出半加器 HA2，将 HA2 的输入变量 p_{10},p_{01} 添加到动态数组的尾部，同时标记 c_1 为加法器内部变量。以此方式依次

遍历数组元素，直到无法继续搜索出加法器，则搜索过程结束。



(a)AIG图　　(b)电路结构图

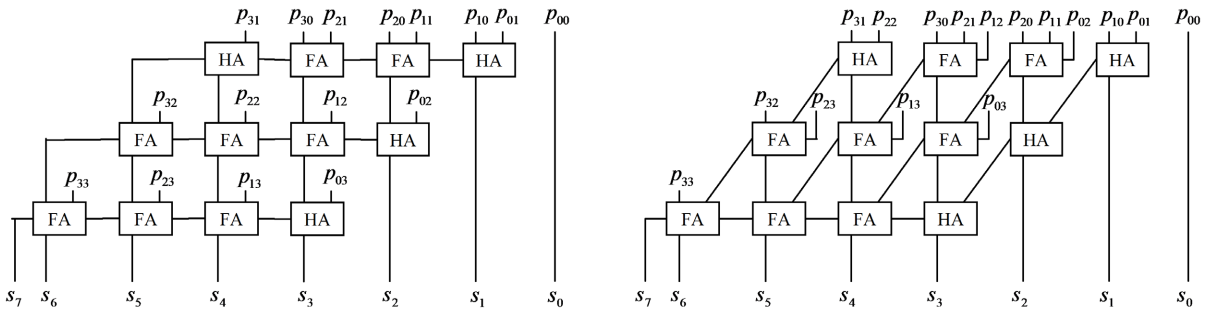
Figure 2. AIG and circuit structure of 2-bit btor multiplier with $p_{i,j} = a_i b_j$

图 2. 二位 btor 乘法器 AIG 与电路图 $p_{i,j} = a_i b_j$

优化算法自乘法器电路的和位输出变量开始，直接根据加法器的结构性知识进行识别，无需借助进位变量，从而避免了全部变量的遍历过程，减少了时间和空间的使用。从理论上来说，使用动态数组的加法器重写优化算法提高了乘法器的验证效率。

4. 对比实验

优化算法的实现以验证工具 AIGMULTOPOLY [5]为基础。本文的实验选择 btor 和 sp-ar-rc 两种类型的乘法器(如图 3)进行验证，其中 btor 乘法器基准是由 Boolector [11]生成的；sp-ar-rc 乘法器基准作为 AOKI 基准集的一部分，是由算术模块生成器 AMG [12]生成的。这两种乘法器虽然功能相同，但累加方式不同。btor 乘法器以网状结构累加，sp-ar-rc 乘法器以对角结构累加。乘法器电路对应的 AIG 文件、源代码以及实验结果都包含在实验文件 Darray_aigmultopoly 中。



(a)btor乘法器　　(b)sp-ar-rc乘法器

Figure 3. Structure of 4-bit btor and sp-ar-rc multiplier

图 3. 四位 btor 乘法器与 sp-ar-rc 乘法器结构图

实验使用的是具有 Inter i5-4210U 1.70GHz CPU 和 8GB 主内存的 Ubuntu 18.04 虚拟机。由于本次实验与 AIGMULTOPOLY 使用的计算机代数系统都为 Mathematica 和 Singular [13]，因此，我们不比较这

一过程消耗的时间。实验所用时间都以秒为单位，内存以 MB 为单位，时间限制为 1200 秒，内存限制为 4096 MB。实验结果如下：

Table 1. Comparison of adder-rewriting algorithm with optimization algorithm
表 1. 加法器重写算法与优化算法对比表

乘法器	位数	加法器重写算法		优化算法	
		时间/s	内存/MB	时间/s	内存/MB
btor	16	0.03	2.09	0.01	1.86
btor	32	0.13	3.12	0.04	2.84
btor	64	0.78	8.37	0.15	6.82
btor	128	5.68	29.18	0.70	21.99
sp-ar-rc	16	0.03	2.02	0.01	2.04
sp-ar-rc	32	0.16	3.83	0.05	3.41
sp-ar-rc	64	1.07	11.21	0.21	8.80
sp-ar-rc	128	7.68	40.78	0.92	30.28

表 1 结合两种不同类型的乘法器对加法器重写算法和使用动态数组的优化算法进行了比较。通过对比第 3 列和第 5 列、第 4 列和第 6 列的数据可以看出，随着乘法器位数的不断增加，优化算法中的计算效率提高的愈加明显，并且能够使用更少的内存空间。这表明使用动态数组来优化加法器重写算法能够在很大程度上提高乘法器的验证速度，同时能够减少内存的使用。

5. 结束语

本文的主要工作是使用动态数组来提高 Gröbner 基法中加法器重写算法的效率。该算法利用动态数组占用空间小、运行速度快等特点，以从输出到输入的逆拓扑序来存储加法器识别所需的变量；结合加法器的结构性知识，通过遍历数组元素来查找加法器，而无需借助进位变量，从而避免了因变量的重复遍历导致的时间和空间大量损耗的问题，提高了乘法器的验证速度。

实验结果表明，使用更加优化的数据结构对算法的重新设计起到了显著的作用。使用动态数组的优化算法能够在很大程度上消除冗余遍历，缩短代码的运行时间，从而提高整个乘法器的验证速度，还能减少内存的使用。但就目前而言，该算法还仅应用于验证整数乘法器电路的 Gröbner 基法中。在未来的工作中，我们希望能够将优化后的加法器重写算法应用于除法器和浮点运算器等更加复杂的算术电路验证中，也希望将这种算法应用于其它的验证方法中，例如代数重写法。

参考文献

[1] Shekhar, N., Kalla, P. and Enescu, F. (2007) Equivalence Verification of Polynomial Data Paths Using Ideal Membership Testing. *IEEE Transactions on Computer-Aided Design*, **26**, 1320-1330. https://doi.org/10.1109/TCAD.2006.888277

[2] Yu, C., Brown, W., Liu, D., Rossi, A. and Ciesielski, M. (2016) Formal Verification of Arithmetic Circuits by Function Extraction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **35**, 2131-2142. https://doi.org/10.1109/TCAD.2016.2547898

[3] Sayed-Ahmed, A., Große, D., Kühne, U., Soeken, M. and Drechsler, R. (2016) Formal Verification of Integer Multipliers by Combining Gröbner Basis with Logic Reduction. *Proceedings of the* 2016 *Conference on Design, Automation & Test in Europe*, Dresden, 14-18 March 2016, 1048-1053. https://doi.org/10.3850/9783981537079_0248

[4] Ritirc, D., Biere, A. and Kauers, M. (2017) Column-Wise Verification of Multipliers Using Computer Algebra. *Pro-*

ceedings of the 17*th Conference on Formal Methods in Computer-Aided Design*, Vienna, 2-6 October 2017, 23-30. https://doi.org/10.23919/FMCAD.2017.8102237

[5] Ritirc, D., Biere, A. and Kauers, M. (2018) Improving and Extending the Algebraic Approach for Verifying Gate-Level Multipliers. 2018 *Design, Automation & Test in Europe Conference & Exhibition*, Dresden, 19-23 March 2018, 1556-1561. https://doi.org/10.23919/DATE.2018.8342263

[6] 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

[7] 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* (*DATE*), Grenoble, 9-13 March 2020, 544-549. https://doi.org/10.23919/DATE48585.2020.9116485

[8] Cox, D., Little, J. and O'Shea, D. (2015) Ideals, Varieties, and Algorithms. 4th Edition. Springer-Verlag, New York.

[9] Biere, A. (2017) The AIGER And-Inverter Graph (AIG) Format, 20071012. Institute for Formal Models and Verification, Linz.

[10] Zed A. Shaw. 笨方法学 C 语言[M]. 王巍巍, 译. 北京: 人民邮电出版社, 2019.

[11] Niemetz, A., Preiner, M. and Biere, A. (2015) Boolector 2.0 System Description. *Journal on Satisfiability, Boolean Modeling and Computation*, **9**, 53-58 https://doi.org/10.3233/SAT190101

[12] Homma Laboratory, RIEC, Tohoku University (2019) Arithmetic Module Generator. https://www.ecsis.riec.tohoku.ac.jp/topics/amg/

[13] Decker, W., Greuel, G.M., Pfister, G. and Schönemann, H. (2016) SINGULAR. http://www.singular.uni-kl.de