Computer Science and Application 计算机科学与应用, 2023, 13(10), 1980-1987
Published Online October 2023 in Hans. https://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2023.1310196

Hans汉斯

基于变量排序的乘法器电路验证结果的认证器

史美琦，齐　爽，冯天烁，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22日；录用日期：2023年10月20日；发布日期：2023年10月27日

摘　要

验证算术电路特别是门级乘法器电路的正确性是一项重要的研究，目前最有效的验证方法是结合计算机代数和SAT求解来验证门级整数乘法器。为了增加验证结果的可信度，进一步生成证明证书，使用认证器检查以实用代数演算(PAC)证明格式生成单个证明的正确性。在本文中，我们提出了一种基于变量输入顺序的排序方法，使项充分在内部共享以减少冗余项的分配，从而减少认证器所占内存大小。此外，本文用C++语言重新实现了认证器，将函数封装为类，隐藏内部实现细节，提高代码的可读性和复用性，增强了数据安全性。

关键词

乘法器电路，计算机代数，实用代数演算，认证器

Authenticator for Verification Results of Multiplier Circuits Based on Variable Ordering

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

School of Mathematics, Liaoning Normal University, Dalian Liaoning

Received: Sep. 22nd, 2023; accepted: Oct. 20th, 2023; published: Oct. 27th, 2023

Abstract

Verifying the correctness of arithmetic circuits, especially gate-level multiplier circuits, is an important study, and currently the most effective verification method is to combine computer algebra and SAT solving to verify gate-level integer multipliers. To increase the confidence of the verification results, proof certificates are further generated using a certifier to check the correctness

*通讯作者。

文章引用：史美琦，齐爽，冯天烁，江建国．基于变量排序的乘法器电路验证结果的认证器[J]．计算机科学与应用，2023, 13(10): 1980-1987. DOI: 10.12677/csa.2023.1310196

of generating individual proofs in Practical Algebraic Computation (PAC) proof format. In this paper, we propose a sorting method based on the order of variable inputs so that items are fully shared internally to reduce the allocation of redundant items, thus reducing the memory size occupied by the authenticator. Inaddition, this paper re-implemented the authenticator in C++, encapsulating the function as a class, hiding the internal implementation details, improving the readability and reusability of the code, and enhancing the data security.

Keywords

Multiplier Circuits, Computerized Algebra, Practical Algebraic Algorithms, Authenticator

Copyright © 2023 by author(s) and Hans Publishers Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/
Open Access

1. 引言

门级整数乘法器是数字电路系统中重要组成部分，也是如数学信号处理、密码学和机器学习中不可或缺的运算单元。复杂多样的乘法算法和大量的构建块使其成为许多设计中最复杂的部分之一，所以在今天，整数乘法器的全自动验证仍然是一项困难的工作。

形式验证可以用来证明或反驳给定系统相对于电路规范的正确性。为给定系统建立数学模型，并应用自动决策过程来导出所需的正确性属性。到目前为止，已经研究出多种用于验证乘法器电路正确性的求解技术。如 1994 年著名 Pentium FDIV 错误[1]使用的其中一种技术是基于二进制决策图[2]，然而，这种方法严重依赖于乘法器的架构，容易发生指数爆炸。基于可满足性检查(SAT)的方法[3]不具可扩展性，无法验证大规模的乘法器电路。基于定理证明器和 SAT 相结合的方法可以证明工业乘法器[4]，但定理证明器不是完全自动化的。最近，有学者优化了定理证明器的方法[5]。然而，乘法器必须以分层 SVL 网表的形式给出，这依赖于电路分层信息的保存，对我们研究的乘法器仍然是不适用的。对于扁平门级乘法器，目前最成功的技术是基于代数推理[6] [7] [8] [9] [10]。在这项工作中，电路被建模为一组生成 Gröbner 基的多项式，然后使用多项式归约算法检查规范是否由电路多项式隐含。

即使代数推理工作可以有效地验证乘法器电路，但验证过程可能不是没有错误的。为了保证验证结果的正确性，必须正式地检查验证结果。例如，使用定理证明器，需要繁琐的工作步骤，同时在结构复杂的工具上是不可行的。因此，增加对验证结果的信任的一种更常见的技术是生成证明证书，它监视验证过程的步骤并可以复制证明。现有一种涵盖代数推理的证明格式 PAC。它基于多项式演算(PC) [11]，通过捕捉门多项式，使用代数理想理论从给定的多项式集导出。与 PC 相比，PAC 证明是它扩展后的实例化，还允许对多项式进行索引，添加删除规则和扩展规则，能够更有效地被认证器检查。

本文通过改进认证器中变量的排序方法，利用级别值对变量进行排序，大大减少了不能进行内部共享的冗余项的数量，进而减少认证器的内存使用。其次本文用 C++语言实现了基于 PAC 证明格式的认证器。将不同操作的函数封装为各个类，避免了程序元素之间的高耦合性，为认证器之后的使用提供了更好的可维护性和可扩展性。

2. 基本概念

2.1. 乘法器电路

非循环的门级整数乘法器电路，电路规范表示其输入与输出之间的期望关系。如果电路对所有输

入产生的输出都与这个期望的关系相匹配，那么就说它就满足电路规范。形式验证就是证明电路符合其规范。

对于具有 $2n$ 个输入位 $a_0,\cdots,a_{n-1},b_0,\cdots,b_{n-1}\in\{0,1\}$ 和 $2n$ 个输出位 $s_0,\cdots,s_{2n-1}\in\{0,1\}$ 的乘法器电路 C。当表示无符号整数的乘法器电路时，其电路规范为 $U_n=-\sum_{i=0}^{2n-1}2^i s_i+\left(\sum_{i=0}^{n-1}2^i a_i\right)\left(\sum_{i=0}^{n-1}2^i b_i\right)$，乘法器是正确的当且仅当对于所有输入，有 $U_n=0$ 成立。当表示有符号整数的乘法器电路时，最高有效位 S_{2n-1}、a_{n-1} 和 b_{n-1} 具有决定符号作用，权重需要进行取反，此时电路规范为 $S_n=-2^{2n-1}s_{2n-1}+\sum_{i=0}^{2n-2}2^i s_i-\left(-2^{n-1}a_{n-1}+\sum_{i=0}^{n-2}2^i a_i\right)\left(-2^{n-1}b_{n-1}+\sum_{i=0}^{n-2}2^i b_i\right)$，乘法器是正确的当且仅当对于所有输入，有 $S_n=0$ 成立。

2.2. 代数模型

设 X 表示变量 $\{x_1,\cdots,x_l\}$ 的集合，我们用 $Z[X]$ 表示变量 X 中的多项式环，系数在 Z 中。如上节讨论的具有 $2n$ 个输入和输出的整数乘法器电路，$l_0,\cdots,l_k\in\{0,1\}$ 表示其内部 AIG 节点，变量集 X 在字典序下定义为 $X=\{a_0,\cdots,a_{n-1},b_0,\cdots,b_{n-1},l_0,\cdots,l_k,s_0,\cdots,s_{2n-1}\}$。

电路中的每个逻辑门被编码为多项式，常见的多项式如下：

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

设 $G(C)\subseteq Z[X]$ 是多项式集，根据反向拓扑序排序，包含每个 AIG 节点的对应多项式关系。所有变量 $x\in X$ 都是布尔的，通过布尔值约束集 $B(x)=\{x(1-x)\mid x\in X\}\subseteq Z[X]$ 来约束。

定义 2.1. 如果 $\forall p,q\in I: p+q\in I$ 和 $\forall p\in Z[X],\forall q\in I: pq\in I$，则非空子集 $I\subseteq Z[X]$ 被称为理想。如果集合 $I=\{p_1q_1+\cdots+p_sq_s\mid q_1,\cdots,q_s\in Z[X]\}$，则集合 $P=\{p_1,\cdots,p_s\}\subseteq Z[X]$ 称为 I 的基，称 I 是由 P 生成的，写作 $I=\langle P\rangle$。理想 I 和 J 的和定义为 $I+J=\{p+q\mid p\in I,q\in J\}$。

定义 2.2. 设 $P\subseteq Z[X]$，如果对于某个项阶，P 的所有前导项只由指数为 1 的唯一单个变量组成，并且对所有 $p\in P$，$lc(p)\in\{-1,1\}$，则称 P 具有唯一的单前导项(*UMLT*)。设 $X_0(P)\subseteq X$ 不作为 P 中的前导项出现的所有变量的集合。进一步定义 $B_0(P)=B(X_0(P))$。

此外，由于可以进行模块化推理，只有在 $Z[X]$ 中才能添加常量，我们将常数 $2n$ 添加到 $J(C)$ 的理想生成器中，从多项式中消除系数过大的单项式[8]。

定义 2.3. 设 C 是一个电路，$J(C)=\langle G(C)\cup B_0(X)\rangle\subseteq Z[X]$，是由 $G(C)\cup B_0(X)$ 生成的理想。

定义 2.4. 电路 C 满足规范 L 当且仅当 $L\in J(C)$。

定理 2.1. 设 $J(C)=\langle G(C)\cup B_0(C)\cup\{2^n\}\rangle\subseteq Z[X]$，那么 $G(C)\cup B_0(C)\cup\{2^n\}$ 是 $J(C)$ 关于固定逆拓扑项序的一个 D-Gröbner 基。

在约简过程中，提前消除 $G(C)$ 中的变量，以避免在过程中出现指数级的中间结果，形成更紧凑的 D-Gröbner 基。利用多项式 $G(C)\cup B_0(C)\cup\{2^n\}$，对规范 L 进行 D-Gröbner 基约简，通过检验结果是否为 0 来判断电路正确性。

2.3. 实用代数演算

实用代数演算(PAC)是将多项式演算实例化为更具体的证明格式，其证明格式是包括加法和乘法运算

的一系列证明规则。不仅隐式地处理布尔值约束上的操作，通过归一化指数来减少证明步骤的数量。还对多项式进行索引，为每个给定的多项式和证明步骤使用一一对应的正数标记。例如，第一个步骤产生的结论多项式在第三个证明步骤中被再次作为前提使用，这时我们只需使用索引标记第一个证明步骤，在第三个步骤使用该结论多项式的位置直接使用索引替代表示。并添加了删除规则，在每个证明步骤完成时，添加到约束集中的结论多项式若不再被需要，则使用删除规则来删除不需要的多项式。能够有效地生成更短和更简洁的 PAC 证明。

设 P 表示可以通过索引访问的多项式序列，$P(i)=\perp$ 表示在索引 i 处序列 P 不包含多项式，$P(i\mapsto p)$ 表示将索引 i 处设置为序列 P。初始状态为 $\left(X=Var\left(G\cup\{f\}\right),P\right)$ 其中 P 包含 G 的所有多项式。证明规则如下：

$$ADD(i,j,k,p)\ (X,P)\Rightarrow\left(X,P(i\mapsto p)\right)$$

假设 $P(j)\neq\perp,P(k)\neq\perp,P(i)\neq\perp$，且 $p=P(j)+P(k)\in Z[X]$

$$MULT(i,j,q,p)\ (X,P)\Rightarrow\left(X,P(i\mapsto p)\right) \tag{2}$$

假设 $P(j)\neq\perp,P(k)\neq\perp,q\in Z[X]$，且 $p=q\cdot P(j)\in Z[X]$

$$DELETE(i)\ (X,P)\Rightarrow\left(X,P(i\mapsto p)\right)$$

此外，PAC 添加了扩展规则，它允许将任意新的多项式作为初始约束添加到初始集合 G 中进行扩展。并通过否定对任意 $x\in X$ 引入额外的新变量 $\bar{x}$，添加形式为 $-f_x+1-x$ 的多项式，同时保留原始变量集 X 上的原始模型。

$$EXT(i,v,p)\ (X,P)\Rightarrow\left(X\cup\{v\},P(i\mapsto -v+p)\right) \tag{3}$$

假设 $P(i)=\perp$，且 $v\notin X,p\in Z[X],p^2-p=0$

例如，目标多项式 $-c+1\in\langle G\cup B(X)\rangle\subseteq Q[X]$ 的 PAC 证明如下，其中 $c=a\oplus b=1$。

⟨Constraints⟩	⟨Proof⟩
1. $-b+1-a$	$3+2,1-c+1-2ab$
2. $-c+a+b-2ab$	$2\ d$
⟨Target⟩	$4*1,-2a\ 2ab$
$-c+1$	$1\ d$
	$5+3,4-c+1$

3. 认证器的优化与实现

3.1. 类封装函数

封装性是 C++面向对象语言[12]的三大特性之一，还包括继承性与多态性。我们可以将一切事物都视为对象，对象有其特定属性和行为，具有相同性质的对象，我们将其抽象为类。例如人属于人类，车属于车类。成员变量和成员函数是类函数的两个基本组成部分。成员变量(也称为属性或数据成员)是类的数据存储单元，用于存储对象的状态信息。成员函数(也称为方法或操作)是类的行为或功能，用于操作和处理成员变量。在本文的认证器中，我们封装了如表 1 所示的六个类：

Table 1. Proof the checker's class function implementation
表 1. 证明检查器的类函数实现

类函数	函数描述
Class Variable{}	使用符号表的全局哈希表使变量的保持规范化的函数
Class Power{}	由变量和其指数构成，表示变量幂积的函数
Class Term{}	利用 rest 指针存储的有序的变量幂积的乘积构成的项函数
Class Monomial{}	由一个系数和一个项构成的单项式函数
Class Polynomial{}	利用 rest 指针存储，表示为单项式的排序链表
Class Inference{}	具有多集语义的推论哈希表

接下来我们详细阐述表示变量幂积的类 Class Power{}，如图 1 所示。项 $\tau = x_1^{d_1} \cdots\cdots x_r^{d_r}$ 是关于 $d_i \in N$ 的变量幂的乘积，单项式是项 $c\tau$ 的倍数，多项式是带有两两不同的项的单项式的有限和。成员变量 variable 用来存储变量 $x_1, x_2, \cdots, x_r$，exponent 用来存储指数 $d_1, d_2, \cdots, d_r$。其成员函数包括 print_power()，enlarge_powers()，sort_powers()分别用来表示为输出幂的乘积到文件中的函数，扩容申请原来所占内存二倍的空间进行存储的函数和根据变量顺序对幂积进行排序的函数。

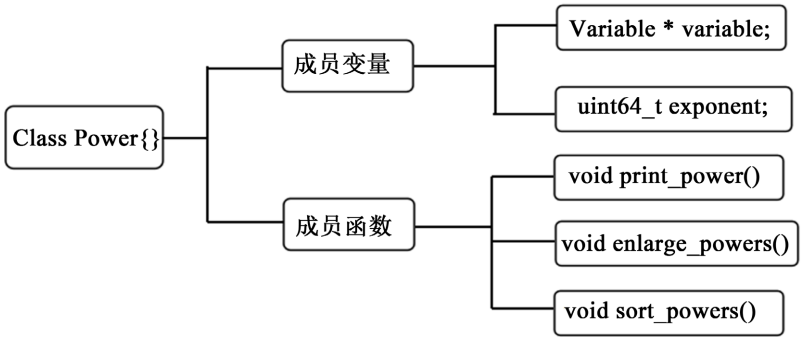

Figure 1. Diagram of member variables and member functions of Class Power
图 1. 类 Power 的成员变量与成员函数示意图

将变量和功能封装在一个类中，隐藏内部实现细节，避免外部直接访问内部的变量和方法，增强了数据的安全性。可以方便地在后面需要的地方重复使用，减少代码冗余，提高代码的可读性。此外类函数可以通过继承和多态的方式进行扩展。通过继承可以创建新的类，例如，多项式类 Polynomial 是单项式类 Monomial 的扩展，单项式类 Monomial 是项类 Term 的扩展，它们都继承原有类的属性和方法，然后在新类中添加新的功能。多态可以在运行时动态地选择调用不同的方法，从而实现不同的行为。因此，面向对象语言，各对象是一个独立体，各自完成不同的功能，耦合度低，扩展力强，复用性强。

3.2. 优化变量排序

验证工具 Amulet2.0 验证门级整数乘法器生成三个文件<constraints>、<proof>和<target>。文件<constraints>为诱导约束集，验证生成的证明步骤在文件<proof>中给出，文件<target>包含一个符合电路规格的规范多项式。认证器 Pacheck [13]将上述三个文件作为输入，目的在于验证目标规范多项式是否包含在约束集中的多项式生成的理想中。

多项式类 Polynomial，表示为单项式的有序链表，用于完成所需计算的所有多项式的算数运算。类 Monomial 用于表示多项式中的单项式，每个单项式由一个系数和一个项组成。类 Term 用于表示多项式中的项，将项表示为变量的有序链表，这些变量使用哈希表在内部共享。不同的变量排序大大影响了产生项的数量，从而影响了认证器的内存使用。如图 2 所示，选择不同的变量排序方式来表示两个单项式 uxy 和 vxy 。关于 $v>u>x>y$ 的变量顺序，项 x 和项 y 可以进行内部共享，只需分配 4 个项。而关于 $x>u>y>v$ 的变量顺序，项不能进行共享，需要分配 6 个项。

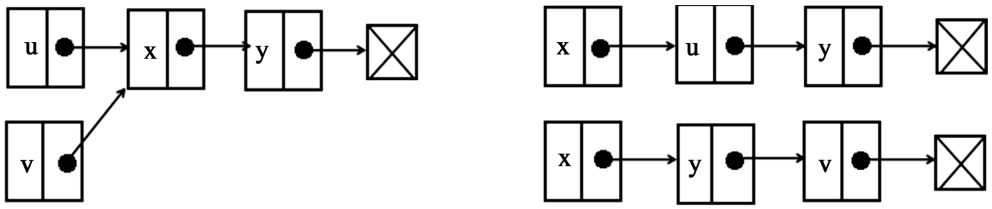

Figure 2. Term internal representation of different variables w.r.t. $v>u>x>y$ (left) and $x>u>y>v$ (right)
图 2. 关于不同变量顺序 $v>u>x>y$ (左)和 $x>u>y>v$ (右)项的内部表示

如表 2 所示，我们提出了新的基于级别值对变量排序的函数 cmp_variable_level()。替换掉原来的 cmp_variable_name()函数，即通过比较变量存储的字符串进行排序的方法。选择与给定证明文件中变量输入顺序相同的排序方法，在读取给定证明文件时为新变量分配递增的级别值 level，并根据该值进行排序。多项式中的项使用与变量相同的顺序进行排序。从而可以充分地内部共享项，减少没有必要的项的分配，从而减小证明检查器的内存使用。例如检查 128 位 sp-ar-rc 乘法器所生成的证明证书，按照原始比较字符串的方法排序产生了 340,354 个项占总项数 64%，而选择变量级别值排序，产生项数占总项数 57%，大大减少了生成的项的数量。

Table 2. Partial function implementation of the variable sort
表 2. 变量排序部分函数实现

类函数	函数描述
is_valid_variable_name()	获取变量有效的变量名称
new_variable()	为新变量分配递增的级别值 level
print_variable()	输出变量到文件中
cmp_variable_level()	通过比较变量 level 值，对变量排序的函数

4. 实验

本文实验使用了一台带有 Ubuntu18.04 虚拟机的电脑，配备 Intel(R) Pentium(R) CPU G4560 3.50 GHz 和限制为 4 GB 的主内存。实验时间以秒为单位，最高限制为 300 秒。首先使用工具 Amulet2.0 生成统一计算机代数和 SAT 证明的 PAC 证明格式的证书，验证输入位宽度为 n 的乘法器的正确性。实验上半部分选择架构为 btor 和 sp-ar-rc 类型的乘法器，以验证大型简单乘法器电路的正确性。下半部分选择 aoki 基准，其中乘法器的末级加法器都是 GP 加法器，以验证含有复杂末级加法器的复杂乘法器电路的正确性，实验结果如表 3 所示。

本文的实验中，我们首先对不同架构下的乘法器进行验证，生成统一 PAC 格式的证明证书。再比较由认证器检查证明证书所用时间和占用内存大小。通过表 3 的数据结果可以看出，优化后的认证器不仅

能够检查带有复杂末级加法器的乘法器生成的证明证书，而且检查各个类型乘法器证明证书在时间上和内存上都得到了优化。

Table 3. Optimize before and after the proof checker occupied time vs. memory comparison data
表 3. 优化前后证明检查器的时间与内存对比数据

乘法器	位宽	Pacheck		our	
		时间/s	内存/MB	时间/s	内存/MB
btor	128	6.62	91.71	3.87	91.30
btor	256	26.63	369.79	24.70	363.62
sp-ar-rc	128	5.23	138.76	4.41	135.49
sp-ar-rc	256	29.08	554.01	20.98	540.51
sp-ar-cl	64	1.23	34.75	0.94	34.82
sp-dt-lf	64	1.54	35.21	0.99	34.71
bp-wt-cl	64	1.24	29.91	1.05	29.00
bp-ct-bk	64	1.28	25.90	0.97	24.81

5. 结束语

本文优化了变量排序方法，根据证明文件中变量输入顺序排序的方法，能够充分地在内部共享项，减少了不必要项的分配。实验结果表明，优化后的认证器减少了认证过程中的内存使用。此外我们基于C++语言重新实现了检查 PAC 证明格式的认证器，引用类函数，将对象的属性和操作封装为一个独立的整体，大大提高了程序的安全性和复用性，同时降低了函数元素间的高耦合性，使我们的认证器更具扩展性和可维护性。在未来的工作中，我们希望对认证器进行扩展以检查更多不同格式的证明证书，并占用更小内存。

参考文献

[1] Sharangpani, H. and Barton, M.L. (1994) Statistical Analysis of Floating Point Flaw in the Pentium Processor.

[2] Bryant, R.E. (1986) Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, **35**, 677-691. https://doi.org/10.1109/TC.1986.1676819

[3] Biere, A. (2016) Collection of Combinational Arithmetic Miters Submitted to the SAT Competition 2016. University of Helsinki, 65-66.

[4] Hunt, W.A., Kaufmann, M., Moore, S.J., *et al*. (2017) Industrial Hardware and Software Verification with ACL2. *Philosophical Transactions of the Royal Series A*, **375**. https://doi.org/10.1098/rsta.2015.0399

[5] Temel, M., Slobodová, A. and Hunt, W.A. (2020) Automated and Scalable Verification of Integer Multipliers. In: Lahiri, S., Wang, C., Eds., *Computer Aided Verification*, Vol. 12224, Springer, Cham, 485-507. https://doi.org/10.1007/978-3-030-53288-8_23

[6] Maciej, C., Tiankai, S., Atif, Y., *et al*. (2019) Understanding Algebraic Rewriting for Arithmetic Circuit Verification: A Bit-Flow Model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **39**, 1346-1357. https://doi.org/10.1109/TCAD.2019.2912944

[7] Daniela, K. (2022) Formal Verification of Multiplier Circuits Using Computer Algebra. *it-Information Technology*, **64**, 285-291. https://doi.org/10.1515/itit-2022-0039

[8] 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

[9] Mahzoon, A., Große, D. and Drechsler, R. (2019) RevSCA: Using Reverse Engineering to Bring Light into Backward Rewriting for Big and Dirty Multipliers. *Proceedings of the* 56*th Annual Design Automation Conference* 2019, Las Vegas, 2-6 June 2019, 1-6. https://doi.org/10.1145/3316781.3317898

[10] Mahzoon, A., Große, D., *et al.* (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

[11] Clegg, M., Edmonds, J. and Impagliazzo, R. (1996) Using the Groebner Basis Algorithm to Find Proofs of Unsatisfiability. *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, Philadelphia, 22-24 May 1996, 174-183. https://doi.org/10.1145/237814.237860

[12] 张鸿, 冯文新, 主编. C++面向对象程序设计教程[M]. 武汉: 武汉大学出版社, 2008.

[13] Kaufmann, D., Biere, A. and Kauers, M. (2020) From DRUP to PAC and Back. 2020 *Design, Automation* & *Test in Europe Conference & Exhibition* (*DATE*), Grenoble, 9-13 March 2020, 654-657. https://doi.org/10.23919/DATE48585.2020.9116276