计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.190900197

基于 COQ 的有限域 $GF(2^n)$ 的形式化研究

范永乾　陈　钢　崔　敏
南京航空航天大学计算机科学与技术学院　南京 211100
高安全系统的软件开发与验证技术工信部重点实验室　南京 211106
(fanyongq123@163.com)

摘　要　有限域 $GF(2^n)$ 是多种安全关键性算法的基础，包括 AES 加密算法、椭圆曲线加密和感染函数掩码等。相关资料表明，有限域上的运算因为自身的复杂性而容易出错，从而导致系统问题。基于测试和基于模型检测的验证方法只能在 n 固定的特定有限域上进行验证，而且计算量往往超出计算机的能力。基于交互式定理证明器的形式化验证为有限域性质的通用验证提供了可能性，但这方面的工作难度较大。已有研究主要针对有限域的抽象性质进行形式化验证，但计算机领域更关心的是有限域的构造性定义及相关性质的验证。针对这些问题，借助定理证明器 COQ，建立了有限域 $GF(2^n)$ 并给出了其基本运算的构造性定义，同时对一组与有限域有关的基本性质进行了形式化验证，包括有限域加法基本性质的验证、多项式乘法基本性质的验证等，其中多项式乘法是有限域乘法的基础。这项工作为有限域的完整的形式化及基于有限域的算法的形式化验证奠定了基础。

关键词：有限域；COQ；形式化验证；定理证明；多项式运算
中图法分类号　TP311

Formalization of Finite Field $GF(2^n)$ Based on COQ

FAN Yong-qian, CHEN Gang and CUI Min
College of Computer Science and Technology, Nanjing University of Aeronautics and Astronauitcs, Nanjing 211000, China
Key Laboratory of Safety-critical Software, Ministry of Industry and Information Technology, Nanjing 211106, China

Abstract　The finite field $GF(2^n)$ is the basis of many security-critical algorithms, including AES encryption algorithms, elliptic curve cryptography, infection function masks, and so on. There is data that the operations on the finite field are prone to errors due to their complexity, which causes problems in the system. Verification methods based on testing and model checking can only be applied on specific finite field with fixed by n, and the computation time for the verification often exceeds the capability of the computer. Formal verification using interactive theorem provers provides the possibility for generic verification of finite field properties, but working in this direction fairly challenging. The existing researches mainly focused on the formal verification of abstract properties of finite fields, however, solving practical problems require the use of constructive definitions of finite fields and the verification of its properties. In response to this requirement, this paper uses the theorem prover COQ to develop a constructive and generic definition of finite field $GF(2^n)$, and formally verified a large amount basic properties of finite fields, including verification of the basic properties of finite field addition, verification of the basic properties of polynomial multiplication which is the buliding block of finite field multiplication, as well as verification of other related properties. This work lays a foundation for the complete formalization of finite fields, which will support formal verfications of various algorithms using finite field.

Keywords　Finite field, COQ, Formal verification, Theorem proving, Polynomial operation

1　引言

有限域 $GF(2^n)$[1] 是包含 2^n 个元素的域，由于它的元素可以用固定长度的二进制串表示，因此其被广泛应用于 AES 加密[2]、椭圆曲线加密[3]及防御故障攻击的感染传播函数中的掩码[4]等计算机领域。这些算法对安全软件有着重要作用，因此它们自身的可靠性验证在安全行业内受到高度重视。文献[5]对一组安全软件中的缺陷进行了综述性分析，从中发现了 28 个缺陷，其中 20 个来自复杂的有限域运算。为了解决这些问题，国际学者正开展对这些算法的形式化验证研究。然而，以往的研究工作显示，有限域的形式化验证是这些算法形式化验证中的一个关键性难点。

目前尚未有工作在对有限域 $GF(2^n)$ 的运算进行构造定义的基础上对所定义的运算的性质进行形式化验证。这方面工作有 3 个难点：1)不同有限域 $GF(2^n)$ 上的乘法运算不同，很难给出统一的验证；2)有限域上元素的个数随着 n 的增加

到稿日期：2019-09-29　返修日期：2020-03-09　本文已加入开放科学计划(OSID)，请扫描上方二维码获取补充信息。
通信作者：陈钢(gangchensh@qq.com)

呈指数级增长;3)有限域上乘法逆元的性质很难描述。

对于验证工作,目前主要有两种解决方式。1)通过若干公理对域上的运算进行描述性定义[6-8]。这种方式并未定义出具体运算,因此多用于对域理论进行抽象的形式化研究,并不能满足计算机领域中具体实现的有限域算法的验证需要。2)开发对有限域算法进行快速测试的技术[5,9-12]。这种技术提高了测试效率,但是不能全面覆盖所有的可能性。

对有限域 $GF(2^n)$ 上的运算进行完整测试,需要使用 $O(2^n)$ 量级的测试向量。因此,当 n 较大时,测试工作非常耗时,甚至可能超出计算机的能力;此外,这样的验证缺乏通用性。基于通用定理证明器的形式化验证技术,为有限域运算的通用描述和一般性证明提供了可能性。

定理证明器 COQ[13-14] 是基于带类型 λ 演算的一种交互式定理证明工具,在工程安全验证和数学定理证明等领域(如 JavaCard 协议的安全性验证[15] 和四色定理验证[16] 等)被广泛使用。COQ 实现了基于 Curry-Howard 原理的程序抽取,使得 COQ 中定义的数据结构和函数可以转变为 OCaml 和 Haskell 的程序。因此,研究者可以首先在 COQ 中完成算法的正确性验证,然后抽取出正确且可靠的程序。

本文将在定理证明器 COQ 中对有限域 $GF(2^n)$ 上的加法、乘法及乘法逆运算做出一般化的形式化定义;并在此基础上对有限域 $GF(2^n)$ 上加法的基本性质进行形式化验证。在乘法方面,由于有限域乘法运算包含了取模操作,因此对它进行形式化验证的难度较大。鉴于系数在有限域 $GF(2)$ 上的多项式的运算是有限域乘法运算的基础,本文对这类多项式运算的相关性质进行了形式化验证,以期为有限域运算性质的完整验证奠定基础。

本文第 2 节简述了相关工作;第 3 节介绍了背景知识;第 4 节对有限域 $GF(2^n)$ 进行了形式化定义;第 5 节对一组基本性质进行了形式化验证;第 6 节给出了两个应用实例;最后总结全文。

2 相关工作

目前对有限域的形式化研究主要集中在两方面:一方面侧重于对环和域的理论进行形式化;另一方面更关注对具体有限域上的运算进行快速计算,从而更快地进行测试验证。这些工作为算法的形式化描述和验证提供了参考依据。

有限域上的元素通常用多项式表示。COQ 的扩展库 CoLoR 库[17] 对多项式进行了形式化定义;但是该库的目的是完成向量运算,因此其定义的多项式也被视作一种向量,库中定义的多项式运算主要服务于向量运算,对多项式本身的运算和有限域上的运算并未做过多深入的探讨。

COQ 库给出了环[6] 和域[7] 的形式化定义,并在定义的基础上对环和域的理论进行了形式化验证。不过这两个库[6-7] 仅分别就抽象空间上的无限环和无限域展开形式化工作,并没有针对有限环和有限域进行定义。这些工作通过描述性定义,使用若干公理给出相关运算,并没有定义出具体的运算,不适用于计算机领域中需要求得确定值的任务。

文献[8]对有限群理论进行了形式化定义,并且根据定义对群理论做了一些形式化验证。但这项工作并未以构造性方式定义运算,而是以公理的方式定义群上的运算。这种方式的定义不支持计算,虽然能对有限群理论开展一些形式化验证,但不能用来研究运算的具体实现方案及性质验证。这项工作中进行的验证也是在给出的群公理基础上进行的。

文献[5]在 COQ 中使用列表结构构建有限域元素,并在此基础上建立了有限域中算术操作的构造性定义。这项工作以多项式为基础构造了有限域 $GF(p^n)$ 上的运算,因此使用范围更广。该研究的主要贡献是在 COQ 中设计出了有限域操作的快速计算算法,这对快速测试极为有益;但它并没有对这些操作的性质进行形式化验证。当多项式的系数中包含较多 0 时,它有较高的执行效率;但当系数中 0 很少时,计算效率就会降低。

文献[9-12]构建了能高效执行有限域乘法的电路,并支持乘法性质的测试验证。当 n 值较小时,这些工作能完成比较可靠的验证。但是,当 n 值较大时,这些工作则无法全面验证所有的可能性,同时验证工作只能针对特定的 n 进行,缺乏通用性。

AES 算法中的列混淆部分使用了有限域 $GF(2^8)$ 上的乘法,文献[18-19]通过定理证明的方法在验证 AES 算法方面做出了努力。但是,文献[18]对这一部分的验证是通过实际计算并与正确结果进行比对来实现的;文献[19]只验证了字节替代、行移位和轮秘钥加等性质,涉及到有限域 $GF(2^8)$ 上的乘法运算的列混淆操作的形式化验证尚未完成。

文献[6-8]只是通过公理化方式描述域或环或有限群操作,未给出这些操作的构造性定义。这些工作验证的性质基于给出的公理,因此若想把这些性质用在确定的运算中,则需要确保这些运算满足这些公理,须对这些公理在这些运算中是否成立进行形式化验证。

文献[5,9-12]中的某些验证可能只通过运算的性质就可以很容易地得到,例如通过乘法零元的性质容易得到 $((a \cdot b) \cdot c) \cdot 0 = 0$ 成立;但这些工作由于未对这些性质进行严密的证明,因此只能通过执行 3 次乘法运算来得到结果并进行比对来完成验证。

与上述工作不同,本文在定理证明器 COQ 中对有限域 $GF(2^n)$ 上的运算进行了构造性定义,并在此基础上通过定理证明的方式对运算的性质进行了形式化验证。本文在 COQ 中开展的验证工作大量使用了复杂的归纳证明技术;之前并没有工作能在使用构造性方法定义出适用于任意有限域 $GF(2^n)$ 的运算的基础上,完成运算性质的形式化验证。

3 背景

有限域 $GF(p^n)$(其中 p 为质数)[20] 是指包含 p^n 个元素的域,有限域 $GF(2^n)$ 是包含 2^n 个元素的域。有限域 $GF(p^n)$ 上的元素 a 可以用多项式表示为 $a=\sum_{k=1}^{n} a_n x^{k-1}$ $(0 \leqslant a_n < p)$,其中每个 a_n 都是有限域 $GF(p)$ 上的元素。因此,有限域 $GF(2^n)$ 上的元素 a 可以用多项式表示为 $a=\sum_{k=1}^{n} a_n x^{k-1}$ $(a_n=1$ 或 $a_n=0)$,每一项的系数 a_n 都是 $GF(2)$ 上的元素。有限域 $GF(p^n)$ 上有两种二元运算,一般记作加法和乘法。它在加法操作下是一个单位元为 0 的可交换群,其中的非零元素在乘法操作下

构成可交换群，同时乘法对加法保持分配律。

有限域 $GF(2)$ 是指包含两个元素的有限域，它的两个元素可以用 0 和 1 来表示。它的加法定义为异或操作，记作⊕；乘法可以定义为逻辑且操作。通过枚举，很容易验证乘法满足域的性质。

有限域 $GF(2^n)$ 上两个元素的加法可以用多项式加法实现，表示为参与运算的两个多项式各个对应项的系数分别执行有限域 $GF(2)$ 上的加法，即：

$$a+b=\sum_{k=1}^{n}(a_k \oplus b_k)x^{k-1} \tag{1}$$

它的乘法操作定义为两个多项式相乘后对既约多项式 $p(x)$ 取模，即：

$$a \cdot b=(a(x) \cdot b(x)) \bmod p(x) \tag{2}$$

其中，$a(x) \cdot b(x)$ 执行的是多项式乘法，取模执行的是多项式的取模。以有限域 $GF(2^8)$ 为例，当 $a(x)=x^5+x^3+1$，$b(x)=x^4+x^2+1$ 时，取 $p(x)=x^8+x^4+x^3+x+1$，则：

$$\begin{aligned} a \cdot b &=(a(x) \cdot b(x)) \bmod p(x) \\ &=((x^5+x^3+1) \cdot (x^4+x^2+1)) \bmod p(x) \\ &=((x^9+x^7+x^4)+(x^7+x^5+x^2)+(x^5+x^3+1)) \\ &\quad \bmod p(x) \\ &=(x^9+x^4+x^3+x^2+1) \bmod p(x) \\ &=x^5+x^3+x+1 \end{aligned}$$

每个有限域都有一个特定的素因子（即既约多项式）$p(x)$。对于两个同样包含 2^n 个元素的域，若选取的 $p(x)$ 不同，则其被视为两个不同的有限域。

通过上文可以看到，在 $GF(2^n)$ 上的运算中，参与运算的主要是各项的系数，因此 $GF(2^n)$ 上的元素也能表示为一个各项系数组成的二进制串，如 $a(x)=x^5+x^3+1$ 可以表示为 101001。

有限域中求逆元的方法主要有两种[21]：一种方法通过有限域生成元的性质，利用 $a^{p^n-1}=1$ 的性质进行求解，但这种方式执行的次数较多；另一种方法通过扩展欧几里得算法[22]求逆元。本文选用第二种方法求逆元。

4 有限域 $GF(2^n)$ 的形式化定义

4.1 有限域上元素的定义

根据第 3 节，有限域上的元素可以表示成多项式，有限域中的元素可以在 COQ 中表示成一个由各项系数构成的列表。而有限域 $GF(2^n)$ 中的元素的每一项系数均为有限域 $GF(2)$ 上的元素，因此有限域 $GF(2^n)$ 上的元素可以定义为一个由布尔值构成的列表，0 表示 false，1 表示 true（为了方便，下文中 0 都表示 false，1 都表示 true）。

多项式的列表表示可以有两种形式，一种把左边设置为低位，右边设置为高位；另一种把左边设为高位，右边设为低位。考虑到有限域上的运算主要是加法和乘法，这两者在运算时都从低位开始对齐。为了便于归纳定义以及后面形式化验证工作的开展，本文采用左边低位、右边高位的表示方法，如将 $a(x)=x^5+x^4+x^3+1$ 表示为 100111。

用 poly 表示有限域 $GF(2^n)$ 上的元素：

```
Definition poly:=list bool
```

其中，list 表示声明的是列表类型。

4.2 等价关系的引入

在采用列表表示有限域 $GF(2^n)$ 上的元素后，会产生一个问题：运算之后可能会在高位之后出现若干个 0 项。如对于有限域 $GF(2^3)$，111+011=100=0 $x^2+0x+1=1$，对应的列表为 100。但在计算机的判断中，100 和 1 不相等，即 0 x^2+0x+1 和 1 不相等。高位上的 0 是冗余的，且会影响相等性的判断，因此需要对这个问题进行特殊处理。一种处理方式是建立一个评估函数[5]，将每个元素映射为一个十进制值，再根据十进制值判断两者是否相等，但是这样每次都要遍历整个列表来计算出一个值，增加了算法的复杂性。

考虑到 $GF(2^n)$ 是一类特殊的域，域中的每个元素都是一个二进制串，本文采用另外一种方法来判断两个元素是否相等：通过定义一个等价关系，判断两个元素去除高位 0 后是否相等，从而判断两个元素是否相等。

去除高位零的函数记作 clear_f，定义为：

```
Fixpoint clear_f'(a:poly):poly:=
  match a with
  |nil=>nil
  |true::tl=>(true::tl)
  |false::tl=>clear_f' tl
  end.
Definition clear_f(a:poly):poly:=rev (clear_f'(rev a)).
```

其中，*Fixpoint* 是 COQ 中进行递归定义的关键字，*match* 将函数接收的元素 a 与下面的 3 种情况进行匹配得到相应处理，*nil* 表示空列表。*rev* 是对列表进行倒转的函数。这里借助一个去除低位 0 的函数 *clear_f'*，通过倒转去除高位 0。

通过 *clear_f* 可以定义出等价关系 *poly_eq*：

```
Definition poly_eq(a:poly)(b:poly):Prop:=
  clear_f a=clear_f b.
```

该函数接收两个参数 a 和 b，输出命题真或假。如果两个参数去除高位 0 后相等，则认为它们相等。后面验证的性质都是在 poly_eq 这个等价关系下成立的。

4.3 加法定义

加法通过递归来定义，原理是通过递归逐位执行 $GF(2)$ 上的加法。$GF(2)$ 域上的加法为异或操作，COQ 本身有异或操作，但它对参与运算的两个元素进行两次判断才能得出结果，需要输入两个值。在后面的验证中，可能只能确定一个值，这时需要将异或化为第二个值的函数，因此本文将异或操作重写成输出第二个元素函数的操作。

```
Definition bit_add(a:bit)(b:bit):bit:=
  match a with
  true=>negb b
  |false=>b
end
```

通过比较真值表，很容易得到这种加法与 COQ 的异或操作等价。

有限域上两个元素间的加法定义为：

```
Fixpoint poly_add(a:poly)(b:poly):poly:=
  match a,b with
```

```
ha::ta,hb::tb=>(bit_add ha hb)::(poly_add ta tb)
|ha,nil=>ha
|nil,hb=>hb
end
```

该函数接收两个参数，通过递归逐位执行 bit_add 操作。当参与运算的两个元素长度不等时，结果补上较长元素未参与运算的部分。

因为每个元素的加法逆元都是它本身，所以有限域 $GF(2^n)$ 上的减法和加法一样，这一点在下一节也有验证。

在 COQ 中，整数通过归纳定义，没有大小的限制；列表的长度可以为任意大的整数，通过递归定义的加法会遍历整个列表，因此这样定义的加法适用于任意有限域 $GF(2^n)$。后文中为了便于表示，将用“+”表示 *poly_add*。

4.4 乘法定义

有限域上的乘法是模乘法，根据第 3 节中的说明，它可以看作两个元素做多项式乘法后对既约多项式取模，因此需要先对两个多项式的乘法进行形式化定义。

```
Fixpoint poly_mult(a:poly)(b:poly):poly:=
  match b with
  nil=>nil
  |hd::tl=>poly_add ((map (andb hd)) a) (poly_mult
  (poly_rshift a) tl)
end
```

其中，map 将函数(andb hd)依次作用在列表 a 上，如 map (andb 0) 1111=0000。poly_rshift 是低位补 0 的右移操作。这个函数将 a 不断右移，按当前的b_n判断是否加在结果上，如 $1\cdot 101=1*1+(01)*0+(001)*1=101$。后文为了书写方便，用“**”表示 poly_mult。

接着形式化定义两个多项式的取模运算。在取模中，从高位开始执行计算，需要进行高位对齐，因此需要先去掉高位的 0。鉴于在之前的定义中认为二进制列表左边为低位，右边为高位，为了便于运算，可以先递归定义出左边为高位的列表的取模运算，再通过倒转实现完整的运算。

```
Fixpoint poly_mod'(a b:poly)(n:nat):poly:=
  match a,n with
  nil,S n'=>nil
  |nil,O=>nil
  |ha::ta,S n'=>let b':=map (andb ha) b in
    let temp:=poly_add ta (poly_lshift b') in poly_mod'
              temp b n'
  |ha::ta,O=>let b':=map (andb ha) b in poly_add ta
              (poly_lshift b')
end
Definition poly_mod(a:poly)(b:poly):poly:=
let a':=clear_f' (rev a) in let b':=clear_f' (rev b) in rev(po-
        ly_mod' a' b' (getl a' b'))
```

其中，*poly_lshift* 是不补 0 的左移，*getl* 是获得两个元素去除高位 0 后真实差的函数。*poly_mod'* 是一个辅助函数，它是左边为高位的二进制列表的取模运算，具体方法是将 a 不断左移，根据当前的a_n判断是否与 b 的值相减(即相加)，当执行次数等于两个元素真实长度差时，a 的真实长度小于或等于 b，再进行一次运算后，剩下的值即为结果。通过对 *poly_mod'* 进行两次倒转，即可得到 *poly_mod*。

由此可定义出有限域 $GF(2^n)$ 上的乘法 fmult 为：

```
Definition fmult(a b p:poly):poly:=poly_mod (poly_mult a
b) p
```

其中，p 是相应有限域的既约多项式，a 和 b 是参与运算的两个元素。

这里的 p 不是某个定值，它可以表示任意有限域 $GF(2^n)$ 上的既约多项式，同时前文中已说明列表的长度不受限制，因此以上定义的乘法适用于任意有限域 $GF(2^n)$。

4.5 求逆元算法的定义

求逆元通过扩展欧几里得算法[22]实现。

```
Fixpoint EEA(p a s0 s1 d:poly) (n:nat):poly:=
  match n with
  O=>s0
  |S n'=>let q:=poly_div p a in
     if poly_eq' p d then s0 else
     let r:=poly_add p (poly_mult q a) in
     let s:=poly_add s0 (poly_mult q s1) in
     compute_co a r s1 s n' d
end
Definition inv(a:poly)(p:poly):=compute_co p a (false::
nil) (true::nil) (true::nil) (length p)
```

其中，*poly_eq'* 是判断两个元素是否相等的函数，它输出一个布尔值用于判断算法是否终止。整个算法最多执行 n 次，在执行 n 次或 *poly_eq'* 输出 true 后函数终止，n 是有限域 $GF(2^n)$ 中的 n。

这个函数接收两个参数，需要求得逆元的元素 a 和当前有限域 $GF(2^n)$ 的既约多项式 p，输出的结果即为 a 在该有限域内的乘法逆元。

5 相关性质的形式化验证

本节对有限域 $GF(2^n)$ 的相关性质进行了形式化验证，验证后的定理可以成为有限域定理库的一部分，支持未来的工作。

首先，由于本文是通过等价关系 *poly_eq* 来判断两个元素是否相等，完成的验证也都是在等价关系下成立的，因此需在 COQ 中对 *poly_eq* 的相关性质进行验证。接着，本文在 COQ 中验证了有限域 $GF(2^n)$ 上加法的相关性质，包括加法交换律、结合律、单位元和逆元等性质。最后介绍了本文在有限域乘法验证方面取得的一组阶段性成果。由于直接对有限域乘法的性质进行验证很困难，本文目前在 COQ 中验证了系数在 $GF(2)$ 域上的多项式乘法的相关性质，包括交换律、结合律、对有限域加法的分配律、零元和单位元等性质。这是有限域乘法验证中一个必不可少的环节。

从第 4 节中可以看出，域中的元素被定义为 bool 列表，而列表是一种递归结构，因此本文以归纳为主要验证手段，这样得到的相应性质的形式化验证对任意有限域 $GF(2^n)$ 均成立。

5.1 poly_eq 相关性质的验证

首先需要验证 *poly_eq* 的确是一个等价关系。

定理 1 *poly_eq* 是一个等价关系,且当 $a=b$ 时,*poly_eq a b*。

```
Theorem poly_eq_er: ∀ a b c,
  (poly_eq a a) ∧ (poly_eq a b→poly_eq b a) ∧ (poly_eq a b→
  poly_eq b c→poly_eq a c) ∧ (a=b→poly_eq a b).
```

由于 *poly_eq* 是通过等式定义的,因此通过等式的性质进行重写即可完成证明。

在后文对加法单位元和多项式乘法零元的相关验证中,需要对 *false* 列表进行处理。下面提供了两个定理来处理这类问题。通过定理 2,可以在 *poly_eq* 下将任意长度的 *false* 列表转换为 *nil*;定理 3 则提供了一种重写方式,通过它可以将一个在 *poly_eq* 下等价于 *nil* 的列表转化为一个 *false* 列表。

定理 2 任意长度的 *false* 列表在 *poly_eq* 下均等价于 *nil*,即:

```
Theorem nil_poly_eq:
  ∀ n, poly_eq (mk_n_false n) nil.
```

其中,*mk_n_false* 接收一个参数 n,产生一个长度为 n 的 *false* 列表。

通过对 false 列表的长度 n 进行归纳很容易完成证明。

定理 3 如果 a 在 *poly_eq* 下与 *nil* 等价,那么 a 可以表示成长度为它本身的 *false* 列表,即:

```
Theorem poly_eq_nil:
  ∀ a, poly_eq a nil→a=mk_n_false(length a).
```

其中,*length* 是求得列表长度的函数。

通过对 a 进行归纳,并利用定理 2 可以完成定理 3 的证明。

由于 *poly_eq* 由 *clear_f* 构成,在后续的验证中,通常会将 *poly_eq* 展开成 *clear_f* 进行验证,因此下面将介绍几个用来处理包含 *clear_f* 的等式的定理。

在验证中,可能出现多个 *clear_f* 套在一起的情况,这里可以通过验证 *clear_f* 的不动点性质来提供化简方式。

定理 4 构成等价关系 *poly_eq* 的函数 *clear_f* 具有不动点性质,即:

```
Theorem cf_fix: ∀ a, clear_f a=clear_f (clear_f a).
```

这一点可以通过对 a 进行归纳来证明。

在后面的验证中,在归纳进行到 $a=ha::ta$ 这一步时,可以利用的定理可能只能作用于 clear_f ta。下面的定理提供了一个更有效的方式来对包括这类形式的带 *clear_f* $(a++b)$ 的问题进行处理,其中++是将两个列表拼接的操作。

定理 5 对于两个列表 a 和 b,$a++b$ 和 $a++(clear\_f\ b)$ 在 *poly_eq* 下等价,即:

```
Theorem poly_eq_app: ∀ a b,
  poly_eq (a++b) (a++clear_f b).
```

这个定理可以通过对 a 进行归纳,并利用定理 4 进行证明。

在一般性的零元和单位元的验证中,需要对 $true::(mk\_n\_false\ n)$ 这种情况进行处理;在更一般的情况下,可能需要对 $a++(mk\_n\_false\ n)$ 这种类型的元素进行处理。针对以上情况,定理 6 提供了一种处理方式。

定理 6 a 在末尾加上任意长度的 *false* 列表后,仍与它本身在 *poly_eq* 下等价,即:

```
Theorem poly_eq_app_f:
  ∀ a n, poly_eq a (a++mk_n_false n).
```

这个定理可以通过对 a 进行归纳来证明。

在证明中通常是对 clear_f a 中的 a 进行归纳,当 $a=ha::ta$ 时,就会出现 clear_f ha::ta。为了能使用归纳条件,需要对它进行处理。定理 7 和定理 8 给出了处理这个问题的两种更一般的方式,它们不仅能处理 ha::ta 的情形,还能处理 *clear_f* $(a++b)$ 的情形。

定理 7 如果 b 与 c 在 *poly_eq* 下等价,那么 $a++b$ 和 $a++c$ 在 *poly_eq* 下也等价,即:

```
Theorem poly_eq_app_l:
  ∀ a b c, poly_eq b c→poly_eq (a++b) (a++c).
```

通过对 a 进行归纳,并利用定理 6 可以完成定理 7 的证明。

定理 8 如果 $a'++a::nil$ 和 $b'++b::nil$ 在 *poly_eq* 下等价,那么 $a=b$,a' 和 b' 在 *poly_eq* 下等价。其中,a 和 b 是两个 bool 变量,$a::nil$ 表示只含一个元素 a 的列表。这个定理有两部分,可以用两个定理表示:

```
Theorem poly_eq_app_l: ∀ (a' b':poly)(a b:bit),
  poly_eq (a'++a::nil) (b'++b::nil), poly_eq a' b'
Theorem poly_eq_app_b, ∀ (a' b':poly)(a b:bit),
  poly_eq (a'++a::nil) (b'++b::nil), a=b.
```

通过分别对 a 和 b 进行归纳,对 a' 和 b' 的取值进行讨论,并利用定理 3 可以完成定理 8 的证明。

之后会验证运算在等价关系下的与代表元无关性,这需要对在 poly_eq 下等价的元素进行转化。定理 9 给出了一种转化方式,这种方式同样适用于其他需要类似转换的场合。

定理 9 当 a 和 b 在 *poly_eq* 下等价时,设 a 的长度较小,则可以在 a 后添加长度为两者长度差的 *false* 列表,从而得到 b。

```
Theorem poly_eq_app_nf:
  ∀ a b, length a<=length b→poly_eq a b→
  b=a++mk_n_false(length b-length a).
```

分别对 a 和 b 进行归纳,再利用定理 8,即可完成定理 9 的证明。

5.2 加法性质的验证

有限域 $GF(2^n)$ 上的加法基于 $GF(2)$ 域上的加法实现,而 $GF(2)$ 域上的加法是异或操作,因此很容易得到它的交换律和结合律,通过这一点就能得到下面两个性质的形式化验证。

定理 10 poly_add 具有交换律:

```
Theorem poly_add_comm: ∀ a b, a^ +b=b^ +a.
```

对 a 和 b 分别进行归纳即可完成定理 10 的证明。

定理 11 poly_add 具有结合律:

```
Theorem poly_add_assoc:
  ∀ a b c, a^ +(b^ +c)=(a^ +b)^ +c.
```

通过分别对 a,b,c 进行归纳,即可完成定理 11 的证明。

利用定理1,可以证明在等价关系 *poly_eq* 下加法交换律和结合律也成立。

下面对加法单位元的性质进行形式化验证。

定理 12 任意在 *poly_eq* 下等价为 *nil* 的元素都是加法单位元:

```
Theorem poly_add_0: ∀ a n,
  poly_eq (a^ +(mk_n_false n)) a.
```

为了使用方便,根据定理2和定理3,将在 *poly_eq* 下等价为 *nil* 的元素表示为 *mk_n_false n*。

通过分别对 *a* 和 *n* 进行归纳,即可完成定理12的证明。

将定理12具体化,可以得到下面的引理1。

引理 1 如果 *b* 是一个长度不大于 *a* 的 false 列表,那么 $a^\wedge + b = a$:

```
Lemma poly_add_0L: ∀ a n,
  n<=length a→a^ +(mk_n_false n)=a.
```

通过对 *a* 和 *n* 进行归纳,即可完成引理1的证明。

下面对加法逆元性质进行形式化验证。

定理 13 任意一个元素 *a* 的加法逆元都是它本身:

```
Theorem poly_add_inv: ∀ a,poly_eq (a^ +a) nil.
```

对 *a* 进行归纳,再利用定理2即可完成定理13的证明。

下面对加法在等价关系下的与代表元的无关性进行形式化验证。

定理 14 如果 *a* 和 *b* 在 *poly_eq* 下等价,*c* 和 *d* 在 *poly_eq* 下等价,那么 $(a^\wedge + c)$ 和 $(b^\wedge + d)$ 在 *poly_eq* 下等价,即 *poly_add* 在 *poly_eq* 下具有与代表元无关性。

```
Theorem poly_add_uni: ∀ a b c d,
  poly_eq a b→polyeq c d→poly_eq (a^ +c) (b^ +d).
```

直接证明这个定理有一定难度,可以先证明一半的无关性的引理,即引理2。

引理 2 如果 *b* 和 *c* 在 poly_eq 下等价,那么 $(a^\wedge + b)$ 和 $(a^\wedge + c)$ 在 poly_eq 下等价。

```
Lemma poly_add_half_uni: ∀ a b c,
  poly_eq b c→poly_eq (a^ +b) (a^ +c).
```

要证明这个引理,首先需要另外一个引理,相当于引理2的特殊形式。

引理 3 $a^\wedge + b$ 和 $a^\wedge + (b{+}{+}(mk\_n\_false\ n))$ 在 poly_eq 下等价:

```
Lemma poly_add_eq_nf: ∀ a b c,
  poly_eq (a^ +b) (a^ +(b++(mk_n_false n))).
```

证明:这个引理需要分为 $a^\wedge + b$ 的长度大于等于或小于 $a^\wedge + (b{+}{+}(mk\_n\_falsen))$ 的长度这两种情况进行证明,之后分别对 *a* 和 *b* 进行归纳,并通过引理1和定理10完成证明。

有了引理3,再利用定理8和引理1即可完成引理2的证明。

通过引理2和定理10,即可完成定理14的证明。

下面将证明一个加法在 *clear_f* 下表现出的性质,用于将 *clear_f* 套在能使用相应定理的元素上。如对于 *clear_f* $(a^\wedge + b)$,可能只能单独对 *clear_f a* 使用相关定理,这时就要把外层的 *clear_f* 放入内层。

定理 15 *poly_add* 对于 *clear_f* 具有分配律,即:

```
Theorem poly_add_cf_distr: ∀ a b,
  clear_f (a^ +b)=clear_f((clear_f a)^ +(clear_f b)).
```

根据定理4和定理14进行重写完成证明。

最后给出关于加法封闭性的验证。

定理 16 poly_add 具有封闭性,即:

```
Theorem pa_len: ∀ a b,length (a^ +b)=length a ∨ length (a^ +b)=length b.
```

对 *a* 和 *b* 的长度进行讨论,然后分别对 *a* 和 *b* 进行归纳,即得到定理16的证明。

5.3 多项式乘法性质的验证

首先对多项式乘法零元和单位元性质进行形式化验证。

定理 17 任意长的 *false* 列表是乘法零元:

```
Theorem poly_mult_0:
∀ a n,poly_eq (a* *(mk_n_false n)) nil.
```

分别对 *a* 和 *n* 进行归纳,即可完成定理17的证明。

定理 18 true::nil 是乘法单位元:

```
Theorem poly_mult_unit:
  ∀ a,poly_mult a (true::nil)=a.
```

将乘法运算展开,并根据定理10即可完成定理18的证明。

在前文定理的基础上,可以完成多项式乘法交换律的形式化验证。

定理 19 *poly_mult* 在 *poly_eq* 下具有交换律:Theorem pe_poly_mult_comm:

```
  ∀ a b,poly_eq (a** b) (b** a).
```

对 *b* 进行归纳,当 $b = nil$ 时,利用定理16进行证明;当 $b = hb::tb$ 时,可以通过对 *hb* 进行讨论来将乘法化简,之后利用定理10和定理15完成定理19的证明。

有了交换律,可以完成多项式乘法在 poly_eq 下与代表元无关性的形式化验证。

定理 20 *poly_mult* 在 *poly_eq* 下具有与代表元无关性,即:

```
Theorem poly_mult_uni: ∀ a b c d,
  poly_eq a b→poly_eq c d→poly_eq (a** c) (b** d).
```

证明方法与定理14的证明方法类似,即先证明一半的与代表元无关性。一半的与代表元无关性可以通过定理19进行证明,之后就可以完成定理20的完整的证明。

下面完成多项式乘法对有限域上加法分配律的形式化验证。

定理 21 *poly_mult* 在 *poly_eq* 下具有对 *poly_add* 的分配律:

```
Theorem pe_poly_mult_distr: ∀ a b c,
  poly_eq (a** (b^ +c)) ((a** b)^ +(a** c)).
```

通过分别对 a,b,c 进行归纳,之后将乘法分解成带加法的形式,中间可以补充一些引理用于展开乘法,最后利用定理10、定理11、定理13、定理17和定理19完成定理21的证明。

与定理15类似,poly_mult 对 clear_f 也具有分配律。

定理 22 poly_mult 对 clear_f 具有分配律,即:

```
Theorem poly_mult_cf_distr: ∀ a b,
```

clear_f (a * * b) = clear_f ((clear_f a) * * (clear_f b)).

通过定理 20 和定理 4 重写，即可完成定理 22 的证明。

利用上文中的定理，可以完成多项式乘法结合律的形式化验证。

定理 23 poly_mult 在 poly_eq 下具有结合律：

```
Theorem pe_poly_mult_assoc: ∀ a b c,
  poly_eq (a * * (b * * c)) ((a * * b) * * c).
```

通过分别对 a，b，c 进行归纳，并利用定理 11、定理 14、定理 17、定理 19 和定理 21 完成定理 23 的证明。

6 定义运算和验证的使用

6.1 有限域乘法在 AES 算法中的应用

高级加密标准（Advanced Encryption Standard，AES）算法在加密过程中的列混淆[23]部分涉及到了有限域 $GF(2^8)$ 上的乘法，其中选用的既约多项式 $p(x)=x^8+x^4+x^3+x+1$。本节将通过上文的定义，对多 AES 算法加密过程中的列混淆部分[23]进行形式化。

通过 4.4 节中给出的有限域 $GF(2^n)$ 上的一般形式的乘法 fmult 可以定义出 AES 算法中列混淆部分使用的有限域乘法。

```
Definition px: =
true::true::false::true::true::false::false::false::nil.
Definition aes_cmult(a:poly)(b:poly): =
fmult a b px.
```

其中，px 是 AES 算法中选定的既约多项式。

定义 AES 算法中的每一列为：

```
Definition col: =list poly.
```

定义列乘为：

```
Fixpoint col_mult(a:col)(b:col):poly: =
  match a,b with
  |a',nil=>nil
  |nil,b'=>nil
  |hd::ta,hb::tb=>
(aes_cmult a b)^ + (col_mult ta tb)
end.
```

在此基础上可以实现列混淆操作的形式化：

```
Definition mat: =list col.
Fixpoint mixcol'(a:col)(b:mat):col: =
  match b with
  |nil=>nil
  |hb::tb=>(col_mult hb a)::(mixcol' a tb)
end.
Definition mixcol(a:col): =mixcol' a mix_col.
```

其中，*mat* 是 *col* 列表，相当于 *poly* 的矩阵。***mix_col*** 是列混淆过程中使用的混淆矩阵。输入一个待混淆的列，通过 *mixcol* 即可完成列混淆操作。

6.2 乘法零元在验证中的使用

某些加密算法会涉及复杂的有限域运算。验证运算是否正确是很重要的一步，这里将使用本文方法对运算过程中出现零的有限域乘法进行验证。

根据第 5 节中的定理和对模取操作的推论，可以得出对有限域乘法零元的验证。

```
Theorem fmult_0: ∀ a n p,poly_eq (fmult a (mk_n_false n) p) nil.
Theorem fmult_0l: ∀ a n p,poly_eq (fmult (mk_n_false n) a p) nil.
```

这两个引理分别处理乘法中零在左边和零在右边两种情况。

借助上面的定理，可以不经过计算，仅通过相关定理和引理就得到 $a\cdot b\cdot c\cdot d\cdot 0\cdot e=0$ 的验证。

```
Lemma f0_exp: ∀ a b c d e p o,
  o=mk_n_false(length o)→
poly_eq (fmult(fmult(fmult(fmult (fmult a b p) c p) d p) o p) e p) nil.
Proof
  intros a b c d e p o eqo;
    unfold poly_eq;rewrite eqo.
    rewrite pe_fm_split;rewrite fmult_0.
    replace (clear_f nil) with (mk_n_false 0);auto. rewrite fmult_0l;auto.
Qed.
```

其中，pe_fm_split 的作用与定理 15 和定理 22 一样，它将 fmult 外面的 clear_f 套进 fmult 中。

可以看到，验证的定理对任意有限域中的 a，b，c，d，e 均成立，即对有限域 $GF(2^{10000})$ 中的任意 5 个元素同样成立；而目前已有的方法并不能对这种规模的有限域中的运算进行验证。

结束语 本文采用列表的形式，在定理证明器 COQ 中对有限域 $GF(2^n)$ 进行了形式化定义，并对有限域 $GF(2^n)$ 上的元素采用基于布尔列表的低位在左的表示，在此基础上形式化定义了有限域 $GF(2^n)$ 上的加法、乘法及扩展欧几里得算法求逆元运算；在加法方面，完成了对有限域 $GF(2^n)$ 上加法封闭性、结合律、交换律、单位元和逆元等性质的形式化验证；在乘法方面，对构成有限域 $GF(2^n)$ 乘法的多项式乘法的相关性质进行了形式化验证工作，完成了对多项式乘法零元、逆元、交换律、结合律和对有限域 $GF(2^n)$ 上加法分配律的形式化验证。本文的结果适用于任意长度的有限域 $GF(2^n)$，一定程度上弥补了有限域验证方面的缺失。

下一步我们将进行多项式取模方面的形式化验证，以将多项式乘法的性质扩展到有限域乘法上，为未来进行有限域 $GF(2^n)$ 上乘法性质的形式化验证奠定基础。

参 考 文 献

[1] MCELIECE R J. Finite fields for computer scientists and engineers[M]. Springer Science & Business Media，2012.

[2] DAEMEN J，RIJMEN V. The design of Rijndael：AES—the advanced encryption standard[M]. Springer Science & Business Media，2013.

[3] HANKERSON D，MENEZES A J，VANSTONE S. Guide to el-

liptic curve cryptography[M]. Springer Science & Business Media, 2006.
[4] OSHIDA H, UENO R, HOMMA N, et al. On Masked Galois-Field Multiplication for Authenticated Encryption Resistant to Side Channel Analysis[C]// International Workshop on Constructive Side-Channel Analysis and Secure Design. Cham: Springer, 2018: 44-57.
[5] PHILIPOOM J. Correct-by-construction finite field arithmetic in Coq[D]. Cambridge: Massachusetts Institute of Technology, 2018.
[6] INRIA. Coq Ring Theory[DB/OL]. [2019-09-02]. https://coq.inria.fr/distrib/current/stdlib/Coq.setoid_ring.Ring_theory.html.
[7] INRIA. Coq Field Theory[DB/OL]. [2019-09-02]. https://coq.inria.fr/distrib/current/stdlib/Coq.setoid_ring.Field_theory.html.
[8] GONTHIER G, MAHBOUBI A, RIDEAU L, et al. A modular formalisation of finite group theory[C]// International Conference on Theorem Proving in Higher Order Logics. Berlin: Springer, 2007: 86-101.
[9] YU C, CIESIELSKI M. Formal Analysis of Galois Field Arithmetic Circuits-Parallel Verification and Reverse Engineering[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018, 38(2): 354-365.
[10] YU C, CIESIELSKI M. Efficient parallel verification of galois field multipliers[C]// 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 2017: 238-243.
[11] PAAR C, FLEISCHMANN P, ROELSE P. Efficient multiplier architectures for Galois fields GF (2/sup 4n/)[J]. IEEE Transactions on Computers, 1998, 47(2): 162-170.
[12] LV J, KALLA P, ENESCU F. Verification of composite Galois field multipliers over GF ((2^m)n) using computer algebra techniques[C]// 2011 IEEE International High Level Design Validation and Test Workshop. IEEE, 2011: 136-143.
[13] TEAM T C D. The Coq Proof Assistant Reference Manual-Version V8.7.1[C]// INRIA. 2016.
[14] BERTOT Y, CASTÉRAN P. Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions[M]. Springer Science & Business Media, 2013.
[15] ANDRONICK J, CHETALI B, LY O. Using coq to verify java card tm applet isolation properties[C]// International Conference on Theorem Proving in Higher Order Logics. Berlin: Springer, 2003: 335-351.
[16] GONTHIER G. Formal proof-the four-color theorem[J]. Notices of the AMS, 2008, 55(11): 1382-1393.
[17] BLANQUI F, COUPET-GRIMAL S, DELOBEL W, et al. CoLoR: a Coq Library on Rewriting and termination[C]// Eighth International Workshop on Termination (WST2006). Seattle, United States, 2006.
[18] OKAZAKI H, ARAI K, SHIDAMA Y. Formal Verification of AES Using the Mizar Proof Checker[C]// Proceedings of the 2012 International Conference on Foundations of Computer Science (FCS'12). 2012: 78-84.
[19] ARAI K, OKAZAKI H. Formalization of the Advanced Encryption Standard. Part I[J]. Formalized Mathematics, 2013, 21(3): 171-184.
[20] LIDL R, NIEDERREITER H. Finite fields[M]. Cambridge: Cambridge University Press, 1997.
[21] COLLINS G E. Computing multiplicative inverses in $GF(p)$[J]. Mathematics of Computation, 1969, 23(105): 197-200.
[22] ILIEV A, KYURKCHIEV N, RAHNEV A. A Note on Adaptation of the Knuth's Extended Euclidean Algorithm for Computing Multiplicative Inverse[J]. International Journal of Pure and Applied Mathematics, 2018, 118(2): 281-290.
[23] WIKIPEDIA. Rijndael MixColumns [DB/OL]. [2019-04-17]. https://en.wikipedia.org/wiki/Rijndael_MixColumns.

FAN Yong-qian, born in 1993, postgraduate, is a student member of China Computer Federation. His main research interests include formal methods and so on.

CHEN Gang, born in 1958, Ph.D, professor, is a senior member of China Computer Federation. His main research interests include formal methods and so on.