基于幂表的并行加法器的归纳验证

张欢欢[1,2], 邵志清[1,2], 宋国新[1]
(1. 华东理工大学计算机技术研究所, 上海 200237; 2. 中国科学院软件研究所计算机科学重点实验室, 北京 100080)

摘 要: 介绍了基于幂表和重写规则的并行加法器的功能描述, 直接使用重写归纳证明技术验证了这些描述的正确性, 为重写技术用于描述和验证更加复杂的硬件电路奠定了基础.

关键词: 重写; 归纳; 加法器; 描述; 验证
中图分类号: TP332 文献标识码: A 文章编号: 0372-2112 (2003) 06-0932-05

Inductive Verification of Powerlist-Based Carry Lookahead Adders

ZHANG Huan-huan[1,2], SHAO Zhi-qing[1,2], SONG Guo-xin[1]
(1. Institute of Computer Technology, East China University of Science and Technology, Shanghai 200237, China;
2. Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100080, China)

Abstract: Based on functional specifications of carry lookahead adders, and it directly uses rewriting induction techniques to establish the correctness of these specifications by proving that they actually implement addition on the natural numbers. A brief comparison with related work is made and the results show that this approach has advantages in specifying and verifying hardware circuits.

Key words: rewriting; induction; adder; specification; verification

1 引言

目前, 输入输出级的硬件电路的描述和验证方法主要有基于状态的模型检查方法[1, 2]、基于归纳的定理证明方法[3, 4]和基于逻辑的硬件建模方法[5]. 基于状态的模型检查方法非常适合于固定字长的硬件验证, 其验证可以通过符号处理技术自动实现. 对字长较长的电路的验证, 状态数量呈指数增长趋势, 模型检查不能高效地进行. 基于归纳法的硬件电路验证使用定理证明和证明检查系统, 如 Nqthm、RRL、PVS、HOL、Clam 和 Oyster 等, 可以同时证明多种字长规格的电路的正确性, 其不足之处是不能完全自动地进行证明, 需要用户采用启发式策略干预证明过程. 基于逻辑的方法用高阶逻辑对硬件电路建立模型, 然后用逻辑系统验证硬件性质, 其自动化程度相对较低.

本文用重写归纳技术[6]对硬件电路进行描述和验证. 重写系统是一个重要计算模型[7], 具有计算速度快的优点, 在数学和计算机的许多领域(如抽象数据类型、函数式程序设计语言、程序正确性证明和自动推理等)中都有广泛的应用. 另一方面, 归纳法作为经典的数学证明方法, 在人工智能、程序验证和理论计算机科学中有重要的应用[8, 9]. 重写归纳技术作为重写系统和归纳法的结合, 在软件的开发和验证方面已经取得了较好的进展[10, 11], 但是用于硬件的描述和验证并不多见.

本文研究经典的并行加法器的自动验证问题, 为进一步考察乘法器和除法器等复杂电路的描述和验证问题作准备.

2 基本逻辑电路和自然数的描述

限于篇幅, 有关重写系统、归纳法和重写归纳技术的基本符号、概念和理论, 可参阅有关文献[7].

本文用逻辑常元符号 0(表示真)和 1(表示假)分别表示逻辑位 0 和 1, 非、或、与等逻辑操作则分别用函数符号 non、or、and 等表示, 它们可以用下面的重写系统进行描述.

```
Bool=
  S::  bool
  E::  0: bool
       1: bool
       non: bool → bool
       or: bool, bool → bool
       and: bool, bool → bool
  E::  non(0) → 1
       non(1) → 0
       or(0, y) → y
       or(1, y) → 1
       and(0, y) → 0
       and(1, y) → y
```

收稿日期: 2001-12-30; 修回日期: 2003-03-20
基金项目: 国家自然科学基金(No. 69903004); 教育部优秀青年教师资助计划; 教育部科学技术研究重点项目(No. 01077)

事实上, 由归纳法可以证明: non、or、and 分别实现了非门、或门、与门的逻辑功能.

我们用逻辑位 0 和 1 组成的串表示二进制数, 可以用基类型为布尔类型的列表进行描述, 其构造算子为 nil 和 cons.

```
List=
  S: :   list
  E: :   nil: list
  cons:  bool, list y list
```

由于 cons(x, y) 表示在列表 y 之前插入元素 x, 所以这里的描述方法与传统的表示法相反: 列表的第 1 个元素作为最低位. 这样, 自然数 6 表示为 cons(0, cons(1, cons(1, nil))), 即 011, 而传统的二进制表示为 110.

自然数可由下面的重写系统刻划:

```
Nat=
  S: :   nat

  E: :   0: nat
         s: nat y nat
         add: nat, nat y nat
         multiply: nat, nat y nat
         exp2: nat y nat
  E: :   add( x, 0) y x
         add( x, s( y)) y s( add( x, y))
         multiply( x, 0) y 0
         multiply( x, s( y)) y add( multiply( x, y), x)
         exp2( 0) y s( 0)
         exp2( s( x)) y multiply( s( s( 0)), exp2( x))
```

其中构造算子 0 和 s 分别表示自然数 0 和后继函数, add、multiply、exp2 分别表示加法、乘法、计算 2 的幂. 为简单起见, 我们将 s(0)、s(s(0))、s(s((0))), 缩写为 1、2、3, , 并对逻辑位 0、1 和自然数 0、1 不作区分, 当然对表示逻辑位的 0、1 和表示自然数的 0、1 也不作区分, 甚至对自然数 0、1、2, 和表示它们的符号(串) 0、1(即 s(0))、2(即 s(s(0))), 也不加区别. 读者可依据上下文作出判断, 不致引起误解.

为了照顾读者的阅读习惯, 我们经常含混地将 add(x, y)、multiply(x, y)、exp2(x) 分别表示为 x+ y、x# y、2^x, 将 x+ y + z、x# y# z 分别看成是 x+ (y+ z)、x#(y# z) 的缩写. 另外, 有时也用 $\overset{*}{\longrightarrow}$ 表示多步归约关系(即 $\longrightarrow$ 的自反传递闭包).

3　并行加法器的描述

本节介绍基于幂表和重写系统的并行加法器的功能描述, 有关细节和符号含义可参考 Cyrluk、Kapur 和 Misra 等的工作[12~15].

与串行加法器的逐位进位不同, 并行加法器采用进位预测的办法, 因而具有更高的效率. 两个二进制数的相加过程中产生的进位可以预测如下: 如果一个位置上的两个位值相同, 则这个位置上产生的输出进位就是那个相同位值本身; 否则进位不能确定, 依赖于低位产生的输出进位, 所以我们把这种位置上产生的输出进位记作 P. 例如, 设有两个位向量 31 1 0 0 1 4 和 31 0 1 0 1 4, 则预测的输出进位向量是 31 P P 0 1 4.

要获得真正的进位向量, 需要对 P 实例化, P 的具体值是从其本身开始自右向左的第一个不为 P 的值. 这样, P 起到了从左向右(即从低位向高位) 传播进位的作用(注意, 关于高低位的约定与传统做法相反). 对于上面的预测输出进位向量, 最终的输出进位向量是 31 1 1 0 1 4. 这个过程可以通过下面的并行前缀计算实现: 设有向量 3 x_1, , , x_n4 和向量元素上二元可结合算子 * , 则关于 * 的并行前缀计算是指向量 3 x_1, x_1 * x_2, , , x_1 * x_2 * , * x_n4. 显然, 如果使用 O(n) 个处理器, 并行前缀计算可在 O(log(n)) 时间内完成.

对于进位计算, 定义 * : bool, bool G {P} y nat,

```
x* Py x,   x* 0y 0,   x* 1y 1
```

这样, 由预测的输出进位向量 3 c_1, , , c_n4 得到实际的输出进位向量 3 c_1, c_1 * c_2, , , c_1 * c_2 * , * c_n > .

幂表是由 2^k(k \ 0) 个标量元素组成的列表, 常用于并行计算的描述和推理[15]. 幂表的构造函数符号是 p 和 tie, p 的作用是构造由单个标量元素组成的幂表, tie 用于将两个相同长度的幂表连接成另一个幂表. 例如, p(3) = 334, tie(31, 24, 33, 44) = 31, 2, 3, 44. 标量元素为 data 类型的幂表 Plist(data) 可以描述如下:

```
Plist( data) =
  S: :   plist( data)
  E: :   p: data y plist( data)
         tie: plist( data), plist( data) y plist( data)
         len: plist( data) y nat
         lst: plist( data) y nat
  E: :   len( p( x)) y s( 0)
         len( tie( s, t)) y add( len( s), len( t))
         lst( p( x)) y x
         lst( tie( s, t)) y lst( t)
```

其中 len 和 lst 分别表示求幂表的长度和最后一个元素.

我们约定: 本文使用的标量元素仅限于 0、1 和 P, 另外 data 类型上的条件选择函数由 cond: bool, data, data y data 定义如下:

```
cond( 1, x, y) y x
cond( 0, x, y) y y
```

311　预测输出进位向量

预测输出进位向量可以用 pcout: Plist(bool), Plist(bool) y Plist(bool G {P}) 描述:

```
pcout( p( x), p( y)) y p( bcout( x, y))
pcout( tie( x, y), tie( u, v)) y tie( pcout( x, u), pcout( y, v))
```

其中 bcout: bool, bool y bool G {P} 定义为

```
bcout( x, y) y cond( x= y, x, P)
```

312　实际输入进位向量

由初始进位和预测输出进位向量产生的实际输入进位向量可以用同时表示向右移位和前缀计算的函数符号 preshf: bool, Plist(bool G {P}) y Plist(bool) 进行描述:

```
preshf( x, p( y)) y p( x)
preshf( x, tie( y, z)) y tie( preshf( x, y), preshf( lst( preshf( x, y))
* , lst( y), z))
```

3.3 并行加法器

设 r 和 s 分别是输出和输入进位向量. 对任何位置 i, 若 $r_i \propto P$, 则两个求和的输入向量在位置 i 处具有相同的位值, 这时求和后的结果向量中该位置处的位值与最终的输入进位 s_i 相同; 否则, 该位置处的位值为 s_i 的补. 因此, 带有进位信息的和向量可以用函数符号 pcin: Plist(bool), Plist(bool G {P}) → Plist(bool) 描述:

pcin(p(x), p(y)) → p(bcin(x, y))

pcin(tie(x, y), tie(u, v)) → tie(pcin(x, u), pcin(y, v))

其中 bcin: bool, bool G {P} → bool 和 cnot: bool G {P} → bool G {P} 的定义为:

bcin(x, y) → cond(y = P, cnot(x), x)

cnot(0) → 1

cnot(1) → 0

cnot(P) → P

最后, 并行加法器用 cla: bool, Plist(bool), Plist(bool) → bool @Plist(bool) 描述.

cla(x, y, z) → pairc(lst(preshf(x, z_1)) * lst(z_1)), pcin(preshf(x, z_1), z_1))),

其中 z_1 = pcout(y, z), lst(preshf(x, z_1)) * lst(z_1)) 表示最终的高位进位, pairc 是配对函数, 计算由两个元素形成的有序对偶.

4 并行加法器的归纳验证

在本节中, 直接用归纳法和重写系统性质对上述描述进行验证. 要证明并行加法器的正确性, 必须先定义幂表和自然数之间的对应关系, 为此引入函数符号 pton: Powerlist(bool) G (bool @Plist(bool)) → nat:

pton(p(x)) → x

pton(tie(u, v)) → pton(u) + pton(v) * exp2(len(u))

pton(pairc(lst(preshf(x, z_1)) * lst(z_1)), pcin(preshf(x, z_1), z_1))))

→ pton(pcin(preshf(x, z_1), z_1) + (lst(preshf(x, z_1)) * lst(z_1)) (exp2(len(preshf(x, z_1)))

在证明主要结论的过程中, 需要下列引理.

引理 1 (1) 若 x ∈ {0, 1}, 则 cnot(x) + x → 1

(2) 若 u, v ∈ {0, 1}* 且 u ∝ v, 则 u + v → 1

引理 2 (1) len(tie(x, y)) → 2len(x)

(2) len(pcout(x, y)) → len(x)

(3) len(pcout(x, y)) → len(y)

(4) len(pcin(x, y)) → len(x)

(5) len(pcin(x, y)) → len(y)

(6) len(preshf(x, y)) → len(y)

证明: 以(2)为例, 施归纳于 x.

(A) 奠基: 若 x = p(u), 则有 v 使得 y = p(v). 因此,

len(pcout(x, y)) → len(pcout(p(u), p(v))) → len(p(bcout(u, v))) → 1

len(x) = len(p(u)) → 1

(B) 归纳步: 若 x = tie(u, v), 则有 s, t 使得 y = tie(s, t). 因此,

len(pcout(x, y)) → len(tie(pcout(u, s), pcout(v, t)))

→ 2len(pcout(u, s)) (由(1))

→ 2len(u) (由归纳假设)

len(x) = len(tie(u, v)) → 2len(u) (由(1))

并行加法器的正确性由下列定理保证.

定理 若 y 和 z 是 bool 型幂表且 len(y) = len(z), 则

pton(cla(x, y, z)) = x + pton(y) + pton(z).

证明: 由 pton、cla 的定义和引理 2, 待证目标是

x + pton(y) + pton(z) = pton(pcin(preshf(x, z_1), z_1) + (lst(preshf(x, z_1)) * lst(z_1)) (exp2(len(z_1))),

其中 z_1 = pcout(y, z). 我们对 y 进行归纳:

(1) y 为 p(u): 由于 len(y) = len(z), 必有 v 使得 z = p(v). 这时,

左 → x + u + v.

z_1 = pcout(p(u), p(v)) → s(bcout(u, v)) → s(cond(u = v, u, P)). 所以,

preshf(x, z_1) → s(x),

lst(preshf(x, z_1)) → x,

lst(z_1) → cond(u = v, u, P),

pcin(preshf(x, z_1), z_1) → pcin(s(x), s(cond(u = v, u, P)))

→ s(bcin(x, cond(u = v, u, P)))

→ s(cond(cond(u = v, u, P)

= P, cnot(x), x))

→ s(cond(u = v, x, cnot(x))).

右 → pton(pairc(x * cond(u = v, u, P), p(cond(u = v, x, cnot(x))))

→ pton(p(cond(u = v, x, cnot(x)))) + (x * cond(u = v, u, P))#exp2(len(p(cond(u = v, x, cnot(x)))))

→ cond(u = v, x, cnot(x))) + 2(x * cond(u = v, u, P))

分情形讨论:

(A) u = v:

右 → x + 2(x * u) = x + 2u (因为 u ∝ P)

左 → x + u + v = x + u + u = x + 2u

(B) u ∝ v:

右 → cnot(x) + 2(x * P)

→ cnot(x) + 2x

= cnot(x) + x + x

→ x + 1 (由引理 1)

左 → x + u + v → x + 1 (由引理 1)

(2) 若 y = tie(s, t), 则由于 len(y) = len(z), 必有 u, v 使得 z = tie(u, v) 并且 len(s) = len(u) = len(v). 这时,

左 → x + pton(tie(s, t)) + pton(tie(u, v))

→ x + pton(s) + pton(t)#exp2(len(s)) + pton(u) + pton(v)#exp2(len(u))

→ x + pton(s) + pton(u) + (pton(t) + pton(v)))#exp2(len(s)),

而 z_1 → pcout(tie(s, t), tie(u, v)) = tie(pcout(s, u), pcout(t, v)), 故

preshf(x, z_1) → tie(preshf(x, pcout(s, u)),

preshf(lst(preshf(x, pcout(s, u)) * lst(pcout(s, u))), pcout(t, v))),

lst(preshf(x, z_1))

→ lst(preshf(lst(preshf(x, pcout(s, u)) * lst(pcout(s, u))), pcout(t, v)),

lst(z_1) → lst(pcout(t, v)),

lst(preshf(x, z_1)) * lst(z_1)

y lst(preshf(lst(preshf(x, pcout(s, u)) * lst(pcout(s, u))))) * lst(pcout(t, v))

len(preshf(x, z_1))

y 2(len(preshf(x, pcout(s, u))))(由引理 2)

y 2(len(pcout(s, u)))(由引理 2)

y 2(len(s))(由引理 2),

pcin(preshf(x, z_1), z_1)

y pcin(tie(preshf(x, pcout(s, u)), preshf(lst(preshf(x, pcout(s, u)) * lst(pcout(s, u))), pcout(t, v))),

tie(pcout(s, u), pcout(t, v)))

y tie(pcin(preshf(x, pcout(s, u)), pcout(s, u)),

pcin(preshf(lst(preshf(x, pcout(s, u)) * lst(pcout(s, u))), pcout(t, v)))

len(pcin(preshf(x, pcout(s, u)), pcout(s, u)))

y len(pcout(s, u))

y len(s)

右 y pton(tie(pcin(preshf(x, pcout(s, u)), pcout(s, u)),

pcin(preshf(lst(preshf(x, pcout(s, u)) * lst(pcout(s, u))), pcout(t, v))))

+ (lst(preshf(x, z_1)) * lst(z_1))#exp2(2#len(s))

y pton(pcin(preshf(x, pcout(s, u)), pcout(s, u)))

+ pton(pcin(preshf(lst(preshf(x, pcout(s, u)) * lst(pcout(s, u))), pcout(t, v))))

#exp2(len(pcin(preshf(x, pcout(s, u)), pcout(s, u))))

+ (lst(preshf(lst(preshf(x, pcout(s, u)) * lst(pcout(s, u))))) * lst(pcout(t, v)))#exp2(2#len(s))

(由 pton(tie(a, b))的定义, 对 lst(preshf(x, z_1)) * lst(z_1)进行重写)

y pton(pcin(preshf(x, pcout(s, u)), pcout(s, u)))

+ pton(pcin(preshf(lst(preshf(x, pcout(s, u)) * lst(pcout(s, u))), pcout(t, v))))#exp2(len(s))

+ (lst(preshf(lst(preshf(x, pcout(s, u)) * lst(pcout(s, u))))) * lst(pcout(t, v)))#exp2(2#len(s))

(由引理 2, len(pcin(preshf(x, pcout(s, u)), pcout(s, u))) y len(pcout(s, u)) y len(s))

= pton(pcin(preshf(x, pcout(s, u)), pcout(s, u)))

+ exp2(len(s))#(pton(pcin(preshf(lst(preshf(x, pcout(s, u)) * lst(pcout(s, u))), pcout(t, v))))

+ (lst(preshf(lst(preshf(x, pcout(s, u)) * lst(pcout(s, u))))) * lst(pcout(t, v)))#exp2(len(t)))

(因为 len(s) = len(t), 所以 exp2(2#len(s)) = exp2(len(s) + len(t)) = exp2(len(s))#exp2(len(t)))

= pton(pcin(preshf(x, pcout(s, u)), pcout(s, u)))

+ exp2(len(s))#(pton(pcin(preshf(lst(preshf(x, pcout(s, u)) * lst(pcout(s, u))), pcout(t, v))))

+ (lst(preshf(lst(preshf(x, pcout(s, u)) * lst(pcout(s, u))))) * lst(pcout(t, v)))(exp2(len(pcout(t, v)))))

(由引理 2, exp2(len(pcout(t, v))) = exp2(len(t))

y pton(pcin(preshf(x, pcout(s, u)), pcout(s, u)))

+ exp2(len(s))#(lst(preshf(x, pcout(s, u)) * lst(pcout(s, u)) + pton(t) + pton(v))

(由归纳假设, 对任何 w,

w + pton(t) + pton(v) = pton(pcin(preshf(w, pcout(t, v)), pcout(t, v)) + (lst(preshf(w, pcout(t, v))) * lst(pcout(t, v)))#exp2(len(pcout(t, v)))))

= pton(pcin(preshf(x, pcout(s, u)), pcout(s, u)))

+ (lst(preshf(x, pcout(s, u)) * lst(pcout(s, u)))#exp2(len(s)) + (pton(t) + pton(v)))#exp2(len(s))

y pton(pcin(preshf(x, pcout(s, u)), pcout(s, u)))

+ (lst(preshf(x, pcout(s, u)) * lst(pcout(s, u)))#exp2(len(pcout(s, u))) + (pton(t) + pton(v)))#exp2(len(s))

= x + pton(s) + pton(u) + (pton(t) + pton(v)))#exp2(len(s))

(由归纳假设, 对任何 w,

x + pton(s) + pton(u) = pton(pcin(preshf(x, pcout(s, u)), pcout(s, u)) + (lst(preshf(x, pcout(s, u))) * lst(pcout(s, u)))#exp2(pcout(s, u))))

综上所述, pton(cla(x, y, z)) = x + pton(y) + pton(z).

5　结束语

本文介绍了基于幂表和重写规则的并行加法器的描述, 并直接用重写系统性质和归纳法进行了验证. 研究结果表明, 用只涉及简单类型的重写系统可以描述硬件电路, 并且能够用重写归纳技术高效地证明描述的正确性.

Kapur 等学者最近一直在研究将归纳证明技术用于硬件验证[13, 14]. 他们研究了基于证明器 RRL(重写规则实验室, Rewrite Rule Laboratory)的并行和串行加法器的描述和验证问题, 分别使用幂表和线性表进行, 并直接证明了基于线性表的串行加法器的正确性. 对于并行加法器的验证, 没有能够给出直接证明, 是通过下列一系列的等价性证明(如图 1)而进行的[13], 最终由基于线性表的串行加法器的正确性确认并行加法器的正确性. 相比之下, 我们采用基于重写归纳的直接证明的方法, 证明方案非常直观, 容易理解, 也没有使用很多复杂的抽象数据类型, 只涉及布尔类型、列表、自然数等几种简单类型, 虽然需要少量的预备引理, 但基本能完成自动证明, 不需要很深入的人工干预. 我们准备用本文的方法对更加复杂的电路(例如, 乘法器和除法器等)进行描述和验证, 进一步说明这种技术的合理性和可行性以及重写系统的实用性, 从而为硬件验证提供一种新的方法和途径.

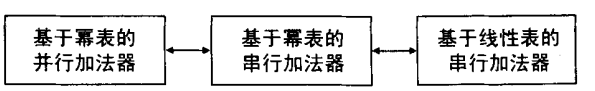

图 1　等价证明

参考文献:

[1] Burch J R, et al. Sequential circuit verification using symbolic model checking [A]. Proc of 27th ACM/IEEE Design Automation Conference [C]. New York: ACM Press, 1990. 46-51.

[2] Bryant R E. Graph-based algorithms for boolean function manipulation [J]. IEEE Transactions on Computer Science, 1986, C235(8): 677-691.

[3] Hunt W A, Brock B C. The verification of a bit2slice ALU [A]. Work2shop on Hardware Specification, Verification and Synthesis: Mathemat2cal Aspects [C]. New York: Springer2Verlag, 1989. 282-306.

[4] Hunt W A. FM8051: A verified microprocessor [D]. Austin: University

of Texas at Austin, 1985.

[5] Camilleri A J, Gordon M J, Malham T F. Hardware verification using higher2order logic [A]. HDL Descriptions to Guaranteed Correct Cur2cuit Designs [C]. Amsterdam: North2Holland Publishing Company, 1987. 43- 67.

[6] 邵志清. 重写归纳技术 [D]. 上海: 上海交通大学, 1998.

[7] Dershowitz N, Jouannaud J2P. Rewrite systems [A]. Handbook of Theo2retical Computer Science, Vol. B [M]. Amsterdam: North2Holland Pub2lishing Company, 1990. 244- 319.

[8] McCarthy J. A basis for a mathematical computation [A]. Computer Programming and Formal Systems [C]. Amsterdam: Noth2Holland Pub2lishing Company, 1963. 33- 70.

[9] Burstall R. Proving properties of programs by structural induction [J]. Computer Journal, 1969, 12(1): 41- 48.

[10] 孙永强, 陆朝俊, 邵志清. 基于重写技术的程序开发与验证 [J]. 软件学报, 2000, 11(8): 1066- 1070.

[11] Shao Zhiqing, et al. Proving inductive theorems using witnessed test sets [A]. Proc of 2nd Conf on Formal Engineering Methods [C]. Las Alamitos: IEEE Computer Society Press, 1998. 158- 164.

[12] Cyrluk D, et al. Effective theorem proving for hardware verification [A]. Proc of 2nd Conf on Theorem Provers in Circuit Design [C]. Bad Herrenalb: Springer2Verlag, 1994. 203- 222.

[13] Kapur D, Subramaniam M. Mechanical verification of adder circuits us2ing powerlists [J]. Journal of Formal Methods in System Design, 1998, 13(2): 127- 158.

[14] Kapur D, Subramaniam M. Mechanically verifying a family of multiply circuits [A]. Proc of 8th Conf on Computer Aided Verification [C]. Berlin: Springer2Verlag, 1996. 135- 146.

[15] Misra J. Powerlist: A structure for parallel recursion [A]. A Classical Mind: Essays in Honor of C. A. R. Hoare [C]. Hertfordshire: Prentice Hall, 1994. 295- 316.

作者简介:

张欢欢 女, 1968 年 1 月生于黑龙江哈尔滨, 1990 年毕业于国防科技大学, 获学士学位, 1997 年毕业于华东理工大学, 获硕士学位, 现为华东理工大学讲师, 博士研究生, 主要研究领域为形式化方法和网络信息服务技术.

邵志清 男, 1966 年 3 月生于江苏常熟, 分别于 1986、1989、1998 年在南京大学、中国科学院软件研究所、上海交通大学获得学士、硕士、博士学位, 现为华东理工大学计算机系系主任、教授、博士生导师, 主要研究领域为网络信息服务技术、软件开发技术和程序设计方法.