CN43-1258/TP
ISSN 1007-130X

计算机工程与科学
COMPUTER ENGINEERING & SCIENCE

2009 年第 31 卷第 10 期
Vol. 31, No. 10, 2009

文章编号:1007-130X(2009)10-0085-05

基于 Isabelle 定理证明器算法程序的形式化验证*

Formal Verification of Algorithmic Programs Based on the Isabelle Theorem Prover

游 珍[1],薛锦云[1,2]

YOU Zhen[1], XUE Jin-yun[1,2]

(1. 江西师范大学省高性能计算技术重点实验室,江西 南昌 330022;2. 中国科学院软件研究所,北京 100190)

(1. Key Laboratory for the High-Performance Computing Technology, Jiangxi Normal University, Nanchang 330022;
2. Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

摘 要:形式化验证对保证软件的正确性和可靠性具有十分重要的意义。定理机械证明是形式化验证的一个重要研究领域,Isabelle 系统是一个被广泛运用的定理证明辅助工具。本文在分析 Dijkstra 最弱前置谓词理论的基础上,根据 PAR 方法开发的算法程序循环不变式,提出了一种使用 Isabelle 定理证明器对算法程序进行机械验证的方法。该方法既克服了传统手工验证过程的繁琐性和易错性等缺点,又达到"提高验证效率和保证算法程序高可信"的目标,具有很好的实用价值。

Abstract: Formal verification plays an important role in ensuring software's correctness and dependability. Theorem's mechanical proof is a significant research domain of formal verification. The Isabelle system is generically considered as a useful tool for theorem proving. In this paper, according to the analysis of Dijkstra's weakest pre-condition theory and the algorithmic program's loop invariant developed by the PAR method, we offer a method of how to mechanically verify algorithmic programs based on Isabelle. The method not only overcomes the shortcomings of traditional manual verification, but also attains the goal of enhancing the dependability and efficiency of the verification process and ensuring the algorithmic programs' trustworthiness. Therefore, it is useful and valuable in practicability.

关键词:形式化验证;定理机械证明;Dijkstra 最弱前置谓词理论;PAR 方法;算法程序;定理证明器

Key words: formal verification; theorem's mechanical proof; Dijkstra's weakest pre-condition theory; PAR method; algorithmic program; theorem prover

doi: 10.3969/j.issn.1007-130X.2009.10.024

中图分类号:TP311

文献标识码:A

1 引言

形式化方法是建立在严格的数学基础上的软件开发方法,主要包括形式化规约(Formal Specification)和形式化验证(Formal Verification)。形式化验证是在对软件系统进行规约的基础上,分析系统是否具有所期望的性质。形式化验证对保证软件正确性和可靠性具有十分重要的意义。

定理机械证明是形式化验证和人工智能的重要研究领域。目前,比较著名的定理证明系统有 ACL2、HOL、Oyster、Coq、PVS、Nuprl 和 Isabelle 等,它们在数学定理证明[1]、硬件验证[2]、协议验证[3]和软件验证[4]等方面发挥着越来越重要的作用。Isabelle[5]是由英国剑桥大学 Paulson L C 教授和德国慕尼黑技术大学 Nipkow T 教授于 1986 年联合开发的基于高阶逻辑的交互式定理证明器,并在不断地更新和发展。

算法程序是用可执行程序设计语言或抽象程序设计语言描述的算法。开发正确且高效的算法程序仍然是计算机

* 收稿日期:2009-04-13;修订日期:2009-07-10
基金项目:国家自然科学基金资助项目(60573080,60773054);科技部国际科技合作资助项目(2008DFA11940)
作者简介:游珍(1982-),女,江西高安人,硕士生,CCF 学生会员(E20-0010471G),研究方向为软件形式化及其自动化;薛锦云,教授,博士生导师,研究方向为软件形式化及其自动化。
通讯地址:330022 江西省南昌市江西师范大学(瑶湖校区)省高性能计算技术重点实验室;Tel:13970927858;E-mail:yucy0405@163.com
Address: Key Laboratory for the High-Performance Computing Technology, Jiangxi Normal University, Nanchang, Jiangxi 330022, P. R. China

科学最具有挑战性的核心部分。随着软件系统的规模越来越庞大，复杂程度越来越高，由算法程序的缺陷和错误造成的危害也日益突出。于是，如何保证和验证算法程序的正确性始终是计算机科学领域中热门的研究课题，吸引了国内外大批计算机科学工作者，包括 Floyd、Hoare、Dijkstra、Gires 等著名科学家，并出现了多种算法程序的验证技术。其中，Floyd 归纳断言法[6] 和 Hoare 公理法[7] 的局限性是在证明过程中要给出语句之间的中间断言；Dijkstra 最弱前置谓词法[8,9] 克服了前两种方法的缺点，提供了书写中间断言的有利手段和工具，即最弱前置谓词 WP( )。

使用定理证明工具对算法程序进行机械证明是一种发展趋势。本文基于 Isabelle 定理证明器，从验证算法程序的 Dijkstra 最弱前置谓词法入手，提出一种对算法程序进行机械验证的方法，既克服了传统手工验证过程的繁琐性和易错性等缺点，又达到"提高验证效率和保证算法程序高可信"的目标，具有很好的实用价值。同时，该方法依赖于循环不变式开发的准确性，这里使用一种系统的算法程序设计方法——PAR 方法[10] 来构造出正确的循环不变式。

2 Isabelle 概述

2.1 Isabelle 特点

Isabelle 是一种通用的交互式定理证明器，支持对数学公式的形式化描述，并为这些公式的逻辑演算提供了证明工具。与其它的定理证明辅助工具相比，Isabelle 定理证明器的特点突出表现在以下几个方面：

(1)既支持多种对象逻辑，又允许自定义新的逻辑。绝大多数定理证明辅助工具仅能支持单一的对象逻辑(如 PVS)，而 Isabelle 系统支持多种对象逻辑，包括直觉主义一阶逻辑(IFOL)、建设性类型理论(CTT)、高阶逻辑(HOL)、基于序列演算的一阶逻辑(LK)和模态逻辑(ML)等。Isabelle 的另一个显著特点在于它支持定义新的逻辑，可以通过定义对象逻辑具体和抽象的句法以及推理规则来实现一个新的逻辑系统。

(2)具有丰富的类型系统。Isabelle 用函数型语言 ML[11] 来编写，ML 是强类型语言，其强类型机制可以帮助用户及时纠正程序的类型错误，保持类型前后的一致性。Isabelle 具有丰富的类型系统，含有如下几种类型[11]：

①基本类型 base types：布尔型(bool)、整数(nat)。

②构造类型 type constructors：表(list)、集合(set)。

③函数类型 function types：用短箭头"⇒"表示，如 $T_1 \Rightarrow T_2 \Rightarrow T_3$ 等价于 $T_1 \Rightarrow (T_2 \Rightarrow T_3)$；在 Isabelle 中，$T_1 \Rightarrow \cdots \Rightarrow T_n \Rightarrow T$ 可以化简写成 $[\![T_1 \Rightarrow \cdots \Rightarrow T_n]\!] \Rightarrow T$。

④多态类型：使用一个先导引号，如'a、'b 来表示一个类型变量，也就是表示任何类型。

(3)具有强大的规则库和灵活高效的命令集。定理证明工具所提供的规则库中的规则越多，证明的过程就越容易。Isabelle 系统的每一个理论中都包含了相关的规则和定理；同时，用户可以根据自己的需要添加定理，并对其进行证明。如果新添加的定理通过证明是正确可行的，系统将存储定理，并且可以应用到以后的定理证明当中。用 Isabelle 进行定理证明的过程中，通常用到两种命令：一种是证明命令，用于定理的求精和验证；另一种是辅助命令，主要是用来检查证明的状态并控制其显示。

2.2 Isabelle 验证方法

Isabelle 是基于高阶逻辑的交互式定理证明器，需要在用户的帮助下来构造定理的证明。它支持两种验证方式：前推证明(Forwards Proof)和后推证明(Backwards Proof)。前推证明是从已知定理的假设条件和公理、定理、定义出发推导出新的公式或定理。更常用的是后推证明，又称为目标制导的证明(Goal-Directed Proof)，它是策略风格的证明。其证明的过程是：用户给出一条规则 $rule_1$(或策略 tac_1)作用到初始目标 g，然后机器就执行一步推理，产生 n 个子目标 $g_i (1 \leqslant i \leqslant n)$，并等待用户给出下一步规则/策略；用户根据当前状态，再指定一个规则 $rule_2$(或策略 tac_2)作用于当前子目标，机器再执行，如此反复直到证明成功(没有子目标)。

在 Isabelle 中，推理规则和证明状态的形式化描述是相同的[12]。在表达推理规则或待证明的定理时使用宽括号[[]]组织前提，用长箭头⇒把前提和结论分开。

例如，$[\![F_1; \cdots; F_n]\!] \Rightarrow F$。

(1)当表示推理规则时，宽括号[[]]中 $F_1; \cdots; F_n$ 表示前提，F 表示结论。

(2)当表示证明状态时，宽括号[[]]中 $F_1; \cdots; F_n$ 表示子目标，F 表示要证明的初始目标。

使用规则和策略的命令形式为：*apply*(*method* [*rule*/*tactic*])。其中，*method* 是证明操作(如命令 *apply*(*auto*) 是对子目标应用证明策略 *auto*，简化子目标)；通常在方法 *method* 后面要带有证明的推理规则(rule)或者策略(tactic)[11]。

2.3 Isabelle 理论

Isabelle 的工作过程即是不断创建理论的过程，其理论文件(T. thy)相当于程序设计语言中的基本模块。在创建新理论 T 时，可以使用格式"imports $S_1 \cdots S_n$"来指定相应的父理论，那么理论 T 继承了其父理论 $S_1 \cdots S_n$ 的类型、常量和规则等，并可以通过定义新的类、类型、常量和规则等对父理论进行扩充。理论 T 的基本结构[10] 如下：

```
theory T
imports S1…Sn
begin
  Declarations;//声明部分
  Definitions; //定义部分
  Proofs //证明部分
end
```

3 使用 Isabelle 形式化验证算法程序

给定一个算法程序，使用 Isabelle 定理证明器对其正确性进行验证的工作流程为：

第 1 步 创建理论文件 *.thy，并选择合适的父理论。

使用 imports 命令选择合适的父理论，其中最常用的是 Main 理论，它包含了所有预定义的基本理论(如 arithmetic、lists、sets 等)，Main 理论是所有理论文件的直接或间接父理论。

第 2 步 在声明部分(Declarations)定义相关数据类型和数据结构。

如果 Isabelle 中不存在待验证算法程序的数据类型和结构,用户可以自定义数据类型。

例如,自定义 Hanoi 塔算法程序中的塔类型:datatype peg = A | B | C;自定义二叉树类型:datatype 'a BTree = Tnull | BT "'a BTree" 'a "'a BTree"。

第 3 步 描述求解问题的形式规约(前置断言 Q 和后置断言 R)。

第 4 步 如果待验证算法程序中含有循环语句,根据 PAR 方法中循环不变式的新定义[13]和循环不变式开发的新策略[13],构造出循环不变式 ρ 和界函数 τ。

第 5 步 用户可以根据需要在定义部分(Definitions)定义相关函数。

Isabelle 定理证明器提供了定义递归函数的功能,并为每个递归函数提供了定制的归纳规则,这些规则和递归定义中的组织结构相一致。可以用两种方式定义:fun 命令和 function 命令,其格式[14]为:

```
fun f :: T
//f 为函数名
//T 给出了函数类型
where
  equations
  ...
function (sequential) f :: T
  where
    equations
    ...
  by pat_completeness auto
termination by lexicographic_order
```

两者的区别在于 Isabelle 能够自动地验证 fun 命令中等式(equations)的正确性,当验证失败时,这个函数定义就不成立,从而导致定义本身的不完善性和不灵活性;function 命令弥补了 fun 命令的不足,使等式(equations)的证明任务可以手工进行分析和解答,选项 sequential 的作用是表示可以进行预处理模式重叠,如果没有 sequential,则所有的等式必须是不相交的。function 命令定义必须满足两个主要性质:完整性和终止性。其中"by pat_completeness auto"是用于证明完整性;而"termination by lexicographic_order"表示递归函数的终止性。

第 6 步 在证明部分(Proofs)验证算法程序中语句的正确性。

(1)如果程序中有条件语句 if,为了证明{Q} if {R}的正确性,Isabelle 需要证明如下定理:$Q \Rightarrow WP(\text{"if"}, R)$。

(2)如果程序中有循环语句 Do,为了证明{Q} Do {R}的正确性,把 Dijkstra 最弱前置谓词法中证明循环语句的五个蕴含表达式分别描述成 Isabelle 能够识别的五个定理:

Theorem-WP1: $Q \Rightarrow \rho$;

Theorem-WP2: $\rho \wedge C_i \Rightarrow WP(\text{"}S_i\text{"}, \rho), 1 \leqslant i \leqslant n$;

Theorem-WP3: $\rho \wedge \neg\, Guard \Rightarrow R$;

Theorem-WP4: $\rho \wedge Guard \Rightarrow \tau > 0$;

Theorem-WP5: $\rho \wedge C_i \Rightarrow WP(\text{"}\tau_1 := \tau;\ S_i\text{"}, \tau < \tau_1), 1 \leqslant i \leqslant n$。

4 数组段最小和算法程序的形式化验证

本文以数组段最小和问题为例说明如何使用 Isabelle 定理证明器形式化验证算法程序。

问题描述:给定整型数组 $a[0:n-1]$,试计算数组 a 中相邻元素的最小和(假定 a 中空段的和为零)。下面使用 PAR 方法[10]中的抽象程序设计语言 Apla 描述出数组最小和问题的算法程序:

```
program MinSumProg;
  const n=10;
  var i,s,c:integer; a:array[0..n-1,integer];
  begin
    writeln("最小和---请输入 10 个整数:");
    foreach(i:0≤i≤n-1:read(a[i]););
    i,s,c:=1,a[0],a[0];
    do i≠n → c:=min(c+a[i],a[i]); s:=min(s,c); i:=i
+1;
    od;
    writeln("这些数的最小和是:",s);
  end.
```

对上述算法程序验证的步骤为:

第 1 步 创建理论文件 minsum. thy,使用 imports 命令选择 Main 理论作为父理论。

```
theory minsum
imports Main
```

第 2 步 Isabelle 含有存放数组的 list 类型,因此无需另外定义。

第 3 步 描述求解问题的形式规约。

前置断言 Q: 给定整型数组 $a[0:n-1], n>0$。

后置断言 R: $minsum(n-1) = (\text{MIN } i, j:\ 0 \leqslant i \leqslant j \leqslant n-1:\ sum(i,j))$。

其中,$sum(i,j) = \begin{cases} 0, & i>j \\ (\sum k:\ i \leqslant k \leqslant j:\ a[k]), & i \leqslant j \end{cases}$

第 4 步 使用 PAR 方法开发出循环不变式 ρ 和界函数 τ。

首先,构造出递推关系:

$minsum(m) = (\text{MIN } i,j:\ 0 \leqslant i \leqslant j \leqslant m:\ sum(i.j)) =$

$(\text{MIN } j:\ 0 \leqslant j \leqslant m:\ (\text{MIN } i:\ 0 \leqslant i \leqslant j:\ sum(i,j))) =$

{交叉积性质}

$(\text{MIN } j: 0 \leqslant j \leqslant m:\ ms(j)) =$

{令 $ms(j) = (MIN\ i:\ 0 \leqslant i \leqslant j:\ sum(i,j))$}

$\min((\text{MIN } j:\ 0 \leqslant j < m:\ ms(j)),\ ms(m)) =$

{范围分裂和单点范围,min 是 MIN 量词对应的二元运算符}

$\min(minsum(a[0:m-1]),\ ms(m))$

于是,我们获得第一个递推关系式:

Recurrence 1 $minsum(m) = \min(minsum(m-1),\ ms(m)), 0 \leqslant m \leqslant n-1$。

然后,找出 $ms(m)$ 和 $ms(m-1)$ 的递推关系:

$ms(m) = (\text{MIN } i:\ 0 \leqslant i \leqslant m:\ sum(i,m)) =$

$(\text{MIN } i:\ 0 \leqslant i \leqslant m:\ (\sum k:\ i \leqslant k \leqslant m:\ a[k])) =$

$(\text{MIN } i:\ 0 \leqslant i \leqslant m:\ (\sum k:\ i \leqslant k \leqslant m-1:\ a[k]) + a[m]) =$

{范围分裂和单点范围}

(MIN i: $0\leqslant i\leqslant m$: $sum(i,m-1)$) + $a[m]$=

{--般分配律和 $sum(i,j)$的定义}

min((MIN i: $0\leqslant i<m$: $sum(i,m-1)$), $sum(m,m-1)$) + $a[m]$=

{范围分裂和单点范围}

min($ms(m-1)$, 0)+$a[m]$=

{$ms(m)$的定义}

min($ms(m-1)+a[m]$, $a[m]$)

于是,我们获得第二个递推关系式:

Recurrence 2 $ms(m)=\min(ms(m-1)+a[m], a[m])$

基于循环不变式的新定义和策略,让变量 s 和 c 分别存放 $minsum(i-1)$和 $ms(i)$的值,得到下列循环不变式:ρ: $s = minsum(i-1)\wedge c=ms(i-1)\wedge 0\leqslant i\leqslant n-1$。

另外,该循环语句的界函数 $\tau= n-i+1$。

第 5 步 根据需要定义相关函数 *sum*、*ms* 和 *minsum*。

在 Isabelle 中使用 fun 命令定义 *sum* 函数 $sum(i, j)=(\sum k: i\leqslant k\leqslant j: a[k])$:

```
fun sum::"nat⇒nat⇒int list⇒int"
where
"sum i j a=(∑k∈{i..<Suc j}.(a! k))"
```

由于 Isabelle 只能对集合 set 类型的数据求最小值,即量词 MIN 只作用于 set。为了定义函数 $ms(m)=$ (MIN i: $0\leqslant i\leqslant m$: $sum(i,m)$),需要先定义集合函数 *set_sum_down*。

```
function set_sum_down::"nat⇒nat⇒int list⇒int set"
where
"set_sum_down i j a=(if i>j then {} else (⋃k∈{1..<Suc j}.{sum k j a}))"
by pat_completeness auto
termination
by (relation "measure(λ(i,j,_).j+1−i)")auto
(*−−−−−−−−min of list_sum−−−−−−−−−−*)
fun ms::"nat⇒int list⇒int"
where
"ms j a=MIN(set_sum_down 0 j a)"
```

使用类似定义函数 *ms* 的方式定义函数 *minsum*。

```
function set_sum::"nat⇒int list⇒int set"
where
"set_sum 0 a={a! 0}"!
"set_sum(Suc m)a=(set_sum m a)⋃(set_sum_down 0(Suc m)a)"
(*−−−−−−−−min of set_sum−−−−−−−−−−*)
fun minsum::"nat⇒int list⇒int"
where
"minsum n a=MIN(set_sum n a)"
```

第 6 步 证明递推关系式 Recurrence 1:$minsum(m)=\min(minsum(m-1), ms(m))$。

```
theorem recur1[simp]:"i≥0⇒(min (minsum i a)(ms(Suc i)a))=(minsum(Suc i)a)"
apply(unfold minsum. simps)
apply(unfold ms. simps)
apply(simp del:sum. simps set_sum_down. simps)
apply(rule Min_Un[symmetric])
apply simp
apply simp
apply simp
apply(rule set_sum_down_Suc_not_null)
done
```

在证明定理 recur1 的过程中需要使用 Isabelle 规则库中的规则 Min_Un:

```
Theorem Min_Un:⟦finite ? A;? A≠{};finite? B;? B≠{}⟧
⇒Min(? A⋃? B)=min(Min? A)(Min? B)
```

可见,还需要另外证明四个引理(其证明过程省略):

```
lemma finite_set_sum_down[simp]:"finite(set_sum_down i j a)";
lemma finite_set_sum[simp]:"finite(set_sum n a)";
lemma set_sum_down_not_null[simp]:"i≥0⇒(set_sum_down 0 i a)≠{}";
lenna set_sum_not_null[simp]:"i≥0⇒(set_sum i a)≠{}"
```

定理 recur1 和引理的依赖关系如图 1 所示。

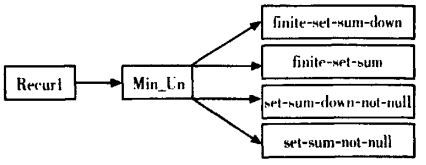

图 1 定理 recur1 的依赖图

第 7 步 证明递推关系式 Recurrence 2:$ms(m)=\min(ms(m-1)+a[m], a[m])$。

```
theorem recur2[simp]:"i≥0⇒(min((a! i)+(ms(i−(1::nat))a))(a! i))=(ms i a)"
apply(unfold ms. simps)
apply(simps only:set_sum_down_recur)
apply(simps del:set_sum_down. simps)
apply (subst Min_add2)
apply simp
apply (simp add:inf_commute)
done
```

定理 recur2 的证明需要自定义引理 Min_add2:

```
lemma Min_add2:"i>0⇒a! i+(Min(set_sum_down 0 (i−Suc 0)a))=Min((λx. (a! i)+x)'set_sum_down 0 (i−Suc 0) a)"
apply(subst eq_set)
apply(rule Min_add1)
apply simp
done
```

引理 Min_add2 又依赖于引理 eq_set 和 Min_add1(其证明过程省略):

```
lemma eq_set:"((λx. m+x)'N)={m+x|x,x∈N}"
lemma Min_add1:"i>0⇒a! i+(Min(set_sum_down 0 (i−Suc 0) a))=Min{a! i+m|m. m∈(set_sum_down 0 (i−Suc 0) a)}"
```

定理 recur2 和引理的依赖关系如图 2 所示。

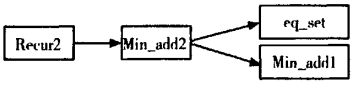

图 2 定理 recur2 的依赖图

第 8 步 证明循环语句 do。

循环语句 do 的五个定理(wp1～wp2)的证明依赖于上述两个定理 recur1 和 recur2,由于这两个定理分别添加了[simp]属性,因此在下面的证明过程中通过 applyauto 命令 Isabelle 会自动把 recur1 和 recur2 作为推理规则。首先,证明循环不变式 $s=minsum\ (i-(1::nat))\ a\wedge c=ms\ (i-(1::nat))\ a\wedge i>0\wedge i\leqslant n$ 在每次循环执行前后都为真:

```
(*−−−proof the wp−−−*)
(*−−−LI:s=minsum(i−(1::nat)) a∧c=ms(i−(1::nat)) a∧i>0∧i≤n−−−*)
theorem wp1:"n≥1∧i=1∧s=(a! 0)∧c=(a! 0)⇒s=minsum(i−(1::nat))a∧c=ms(i−(1::nat))a∧i>0∧i≤n"
apply auto
done
theorem wp2:"s=minsum(i−(1::nat))a∧c=ms(i−(1::nat))a∧i>0∧i≤n∧i≠n⇒min s(min((a! i)+c)(a! i))=minsum i a∧ms i a=(min((a! 1)+c)(a! 1))∧i+(1::nat)>0∧i+(1::nat)≤n"
```

```
apply (simp del:minsum. simps ms. simps)
done
theorem wp3:"s=minsum(i-(1::nat))a∧c=ms(i-(1::
nat))a∧i>0∧i≤n∧i=n⇒s=minsum(n-(1::nat))a"
apply auto
done
```

然后，证明界函数 $n-i+1$ 在每次循环执行前都大于0，且界函数每执行一次后却会递减：

```
(*---proof the termination of program---*)
(*---t:n-i+1---*)
theorem wp4:"s=minsum(i-(1::nat))a∧c=ms(i-(1::
nat))a∧i>0∧i≤n∧i≠n⇒(n-i+(1::nat))>0"
apply auto
done
theorem wp5:"s=minsum(i-(1::nat))a∧c=ms(i-(1::
nat))a∧i>0∧i≤n∧i=n⇒(n-i)<(n-1+(1::nat))"
apply auto
done
```

5 结束语

使用定理证明工具对算法程序进行机械证明是一种发展趋势，也是形式化验证的一个重要研究领域。本文提出了一种使用 Isabelle 定理证明器对算法程序进行机械验证的方法，其优点体现在：(1)实现了算法程序的机械验证替换手工证明过程，克服了传统手工证明的繁琐性和易错性等缺点；(2)验证过程直观严谨，提高了验证效率并保证了算法程序的高可靠性。目前，已运用该方法验证了许多算法程序(如 Hanoi 塔非递归算法、二叉树遍历、图遍历等)及 PAR 平台生成系统中的部分核心代码。同时，我们也必须意识到该方法依赖于循环不变式的准确性和 Isabelle 中定理表述的正确性。不过，根据 PAR 方法中循环不变式的新定义和循环不变式开发的新策略能够帮助我们构造出正确的循环不变式。由于软件本身所固有的复杂性，使得这一领域尚有大量的课题期待进一步研究。

参考文献：

[1] 吴文俊. 东方数学典籍<九章算术>及其刘徽注研究[M]. 西安：陕西人民出版社，1990.

[2] 韩俊刚，杜慧敏. 数子硬件的形式化验证[M]. 北京：北京大学出版社，2001.

[3] Paulson L C. The Inductive Approach to Verifying Cryptographic Protocols[J]. Journal of Computer Security, 1998,6(1-2):85-228.

[4] Schirmer N. Java Definite Assignment in Isabelle/HOL [C]//Proc of the Formal Techniques for Java-Like Programs the 2003 Conf on,2003.

[5] Isabelle [EB/OL]. [2008-06-08]. http://www. cl. cam. ac. uk/research/hvg/Isabelle/.

[6] Floyd R W. Assigning Meanings to Programs[C]//Proc of Symposia in Applied Mathematics,1967:19-32.

[7] Hoare C A R. An Axiomatic Basis for Computer Programming[J]. Communications of the ACM, 1969,12(10):576-580.

[8] Dijkstra E W. A Discipline of Programming[M]. Prentice Hall, 1976.

[9] Dijkstra E W,Scholten C S. Predicate Calculus and Program Semantics[M]. Springer-Verlag, 1990.

[10] Xue Jinyun. A Unified Approach for Developing Efficient Algorithmic Programs[J]. Journal of Computer Science and Technology, 1997,12(4):314-329.

[11] Nipkow T, Paulson L C, Wenzel M. Isabelle/HOL: A Proof Assistant for Higher-Order Logic[M]//LNCS 2283. Berlin: Springer-Verlag, 2004.

[12] Paulson L C. Isabelle: The Next 700 Theorem Provers[M]//Logic and Computer Science, Academic Press, 1990:361-386.

[13] Xue Jinyun. Two New Strategies for Developing Loop Invariants and Their Applications [J]. Journal of Computer Science and Technology, 1993,8(2):147-154.

[14] Krauss A. Defining Recursive Functions in Isabelle/HOL [EB/OL]. [2008-10-01]. http://www. cl. cam. ac. uk/research/hvg/Isabelle/dist/Isabelle/doc/functions. pdf.

(上接第 72 页)

[10] Oh S C, Lee D,Kumara S R T. A Comparative Illustration of AI Planning-Based Web Services Composition[J]. ACM SIGecom Exchange, 2006,5(5):1-10.

[11] Foster H, Uchitel S, Magee J, et al. LTSA-WS: A Tool for Model-Based Verification of Web Service Compositions and Choreography[C]//Proc of the 28th Int'l Conf on Software Engineering, 2006:771-774.

[12] Brogi A, Corfini S, Popescu R. Semantics-Based Composition-Oriented Discovery of Web Services[J]. ACM Trans on Interet Technology,2008,8(4):1-39.

[13] Lécué F, Delteil A, Léger A. Towards the Composition of Stateful and Independent Semantic Web Services[C]//Proc of the 2008 ACM Symp on Applied Computing,2008:2279-2285.

[14] Ryu S H, Casati F, Skogsrud H, et al. Supporting the Dynamic Evolution of Web Service Protocols in Service-Oriented Architectures[J]. ACM Trans on Web, 2008,2(2):1-46.

[15] Sirin E, Parsia B, Wu D, et al. HTN Planning for Web Service Composition Using SHOP2 [J]. Web Semantics: Science, Services and Agents on the World Wide Web, 2004,1(4):377-396.

[16] Gooneratne N, Tari Z, Harland J. Verification of Web Service Descriptions Using Graph-Based Traversal Algorithms [C]//Proc of the 2007 ACM Symp on Applied Computing, 2007:1385-1392.

[17] Menascé D, Casalicchio E, Dubey V. A Heuristic Approach to Optimal Service Selection in Service Oriented Architectures[C]//Proc of the 7th Int'l Workshop on Software and Performance,2008:13-24.

[18] Berardi D, Calvanese D, De Giacomo G, et al. Automatic Service Composition based on Behavioral Descriptions[J]. Int'l Journal of Cooperative Information Systems, 2005,14(4):333-376.

[19] Nezhad H, Benatallah B, Martens A, et al. Semi-Automated Adaptation of Service Interactions[C]//Proc of the 16th Int'l Conf on World Wide Web,2007:993-1002.