软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(9):2248−2260 [doi: 10.3724/SP.J.1001.2012.04164]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于 PAR 的排序算法自动生成研究*

石海鹤[1,2,3+], 薛锦云[1,2]

[1](江西省高性能计算重点实验室(江西师范大学),江西 南昌 330022)

[2](中国科学院 软件研究所 计算机科学国家重点实验室,北京 100190)

[3](中国科学院 研究生院,北京 100049)

Research on Automated Sorting Algorithms Generation Based on PAR

SHI Hai-He[1,2,3+], XUE Jin-Yun[1,2]

[1](Provincial Key Laboratory of High Performance Computing (Jiangxi Normal University), Nanchang 330022, China)

[2](National Key Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[3](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: E-mail: haiheshi@163.com

Shi HH, Xue JY. Research on automated sorting algorithms generation based on PAR. *Journal of Software*, 2012,23(9):2248−2260 (in Chinese). http://www.jos.org.cn/1000-9825/4164.htm

Abstract: Sorting is a kind of special problem in computer science. The flexibility of whose algorithm design tactics leads to the diversity of sorting algorithms. Based on the formal method PAR (partition-and-recur), an automated sorting algorithm generation is studied. The algebraic property of sorting problem is described, generic type components and algorithm components are formally developed, and domain specific language and a formal algorithm generative model are designed. Through replacing the generic identifiers with a few concrete operations a series of known and unknown sorting algorithms, such as quick sort, heap sort, shell sort, and increment select sort, etc., are automatically generated, which is supported by the enhanced program generation system. Through the super framework and underlying components, the reliability and productivity of domain specific algorithm have dramatically improved.

Key words: sorting algorithm; automated generation; domain specific language; formal model; PAR method

摘 要: 排序是计算机学科中的一类特殊问题,其算法设计策略的灵活性使得求解算法更具多样性.基于形式化方法 PAR(partition-and-recur),研究了排序算法的自动生成问题.刻画了排序问题的代数性质,形式化构建了排序算法领域的泛型类型构件和算法构件,建立了排序领域特定语言和算法生成形式化模型,以参数替换的方式自动生成了一组排序算法,包括快速排序、堆排序、Shell 排序等典型的已知算法以及增量选择排序等若干未见于现有文献的算法,并在程序生成系统中予以了实现.通过上层框架研究和底层构件支持,显著提高了特定领域算法的开发效率和可靠性.

关键词: 排序算法;自动生成;领域特定语言;形式化模型;PAR 方法

* 基金项目: 国家自然科学基金(61020106009); 科技部国际科技合作项目(2008DFA11940); 江西省自然科学基金(2010GQS0100); 江西省教育厅科技项目(GJJ12199)

收稿时间: 2011-05-10; 修改时间: 2011-07-21, 2011-09-22; 定稿时间: 2011-11-17

中图法分类号: TP301 文献标识码: A

如何快速、高效地开发出具有高可靠性的软件系统,已成为计算机软件领域的紧迫课题.软件自动化是解决该问题的重要途径[1−3],图灵奖获得者 Jim Gray[4]将它列为 21 世纪信息技术领域的 12 个重要研究目标之一.文献[5]在 ACM FSE 会议上指出,软件自动化在未来软件工程中占据着重要的位置.软件自动化的核心是算法的自动生成问题.算法设计,特别是精妙算法的设计,是软件开发中知识高度密集的创造性劳动.目前,人工智能技术难以处理这类创造性劳动,现有的各种算法程序的优化技术也难以产生精妙算法.该领域的研究进展缓慢.

由于通用的算法自动生成是很难的问题,人们往往研究某些特殊领域的算法自动生成.排序是计算机学科中的一类特殊问题,可应用的算法设计策略的灵活性使得其求解算法更具多样性.从同一问题规约出发,不同的问题分划和规约变换所导致结果算法的形式和性能差异尤为明显.这使得排序类算法的共性难以刻画,算法生成的自动化程度和效果不理想.从一个更广泛的方面着眼,它是研究一般情况下如何着手解决计算机程序设计问题的有价值的实例研究[6].

本文以前期研究形成的形式化方法 PAR(partition-and-recur)[7,8]为基础,对排序问题求解的规律进行了探讨,刻画了排序问题的代数性质,在抽象层次建立了排序领域特定语言和算法生成的形式化模型,并使用 PAR 给出了它们的实现;在进一步扩充的程序生成系统中,以十几种实际参数的替换自动生成了 20 余个已知和未知的排序类算法程序,从而显著提高了算法的开发效率和可靠性.

本文首先介绍相关工作.第 2 节刻画排序问题的代数性质.第 3 节对排序算法领域进行分析和设计,确定并实现泛型类型构件和算法构件,建立领域特定语言和算法生成模型,并通过一个实例展示具体算法的生成过程.第 4 节为本文结果提供系统支持.第 5 节给出总结和讨论.

1 相关工作

Backhouse[9]在程序正确性证明理论的指导下,分析冒泡/堆排序算法思想,构造出冒泡/堆排序的循环不变式,进而非形式化推导出相应的 Pascal 子集描述的程序.类似地,在最弱前置谓词理论指导下,Dromey 等人[10,11]归纳出适合程序推导的问题规约的标准,根据该标准来对后置断言加强约束,得到循环不变式和循环终止条件,从而推导出使用卫式命令语言描述的直接选择排序、冒泡排序、插入排序及快速排序.基于 fold/unfold 变换规则,Clark 等人[12]使用非形式化的一阶谓词逻辑表示规约和程序,开发了归并排序、插入排序、快速排序和选择排序;Merritt 等人[13]通过对用户输入的表达算法设计策略的卷叠问题求解,得到用谓词子句集合表示的递归逻辑程序;Broy[14]在直觉指导下,用 unfold、重排、fold 等规则开发了插入类、选择类、归并类等递归排序程序,并消除递归得到迭代程序.Borges 等人[15]主要基于 Bird-Meertens 演算[16],从排序问题高阶规约出发,通过应用融合和提升等一系列定理逐步推导出函数式的插入排序递归程序.Almeida[17]将 Haskell 语言描述的插入排序算法作为排序问题的规约,通过系列变换开发了归并排序、堆排序和快速排序的函数式程序.Guttmann[18]使用 Haskell 语言作为规约语言和目标语言,从非确定性的规约出发,在半自动程序变换系统 Ultra[19]的支持下推导了堆排序.这些程序变换工作均先得到函数式或逻辑式语言程序,如有需要再将其变换到命令式语言程序.Ward[20]使用最弱前置断言来证明程序求精,开发了用于证明递归和迭代程序终止性的定理,通过逐步求精和手工验证得到一个快速排序算法.Smith[21,22]使用代数方法描述形式化规约,用 PVS 证明求精结果,开发了归并、快速、插入和选择排序这 4 种算法.目前,这些工作都只是产生了单个或几个已知排序算法.虽然 Clark 和 Darlington[12]早就指出,通过推导已知算法来积累有关推导路径方面的经验,可望更改推导路径而发现新的算法,但目前的研究仍只是导致了已知算法的产生.

文献[23]提出,将生成式程序设计[24]作为实现自动程序设计的有效途径.目前,在使用生成式程序设计相关技术开发算法程序方面开展了一些有限的工作.Idate 等人[25]提出了一个称为表达式代码生成器(expression code generator)的系统,用于为数学公式自动产生计算代码;Nedunuri 等人[26]将问题规约和高阶算法理论相结合,为最大子段和问题的 3 个变体分别推导了有效的 Haskell 算法.Rayside 等人[27]从程序员给出的定义了对象

表示抽象视图的函数出发,利用反射机制,为 Java 语言的类中两个特定方法自动生成实现体,即对象等价性判断方法 equals 和计算对象散列值的方法 hashCode.Batory 等人[28]提出一种面向特征的软件开发方法,将程序的功能性定义为特征,以通过组合特征来自动化构建程序.Novak 等人[29]主要基于对泛型软件构件的重用来实现自动程序设计,创建了一个泛型几何程序构件库,为几何计算自动生成程序.此外,Fu 等人[30]提出一种迭代的人工智能规划途径,用以生成类参数化过程的泛型可重用规划,并与构件技术结合生成复杂代码.

本文以 PAR 方法为基础,同时借鉴生成式程序设计相关技术,研究排序算法自动生成的问题.PAR 方法涵盖了多种已知的算法设计技术,将它作为算法生成的统一方法,可以避免在现有各种算法设计方法间作出选择的困难.它由自定义泛型算法设计语言 Radl、泛型抽象程序设计语言 Apla、系统的算法和程序设计方法学及顺序软件开发平台(Apla 到 Java、C++等系列程序生成系统)组成.其中,Radl 语言的主要功能是描述算法规约、规约变换规则以及算法,Apla 描述抽象程序.基于所提出的问题求解序列递推关系的概念,使用 PAR 开发算法程序的全过程可分成功能规约构造、问题分划、规约变换、循环不变式开发和算法程序生成这 5 步,详见文献[7].

PAR 方法已形式化开发了很多具有说服力的算法程序,包括 Knuth 提出的二到十进制的转换算法[31]、Hopcroft 和 Tarjan 发明的图平面性测试和生成算法[32]、循环置换乘方的线性算法[33]等等.此外,PAR 还应用于装备保障领域,通过提出一类离散最优化问题的结构模型和算法推演技术,形式化求解了一系列典型的装备保障算法[34].

2 排序问题的代数性质

对于排序问题,可应用的算法设计策略非常灵活,采用不同的问题分划和规约变换,会得到形式和性能不同的问题求解算法.但这些算法程序间存在的共性,使得我们可以对其进行数据抽象和功能抽象,分析算法类的共性和可变性并建立其形式化模型.下面,我们从排序问题的代数性质上阐述这一点.

首先,我们参考文献[35]分别给出了置换、复合的定义及定理 1.

定义 1(置换). 设 S 是一个非空序列,从序列 S 到 S 的一个双射称为 S 的一个置换.

对于一个具有 n 个元素的序列 S,将 S 上所有 $n!$个不同置换所组成的集合记作 S_n.

定义 2(复合). 设$\pi_1,\pi_2\in S_n$,S_n 上的二元运算$\circ$使得$\pi_1\circ\pi_2$表示对 S 的元素先应用置换π_2,再应用置换π_1所得到的置换,二元运算$\circ$称为复合.

定理 1. $\langle S_n,\circ\rangle$是一个群,$\circ$是 S_n 上的复合运算.

排序问题是一个特殊的置换问题,它将一个初始无序的序列置换为有序序列.为确定起见,我们在下面只对非降序排序问题进行讨论.

定义 3(序抽取). 设 $S=\{a_1,a_2,a_3,\ldots,a_n\}$,$T$ 为 S 的排序结果,$O(a_i)$表示 a_i 元素在 T 中的位置序号,$1\leqslant i\leqslant n$,定义运算∇,有$\nabla(S)=\{O(a_1),O(a_2),O(a_3),\ldots,O(a_n)\}$,$\nabla$称为序抽取运算.

对于一个具有 n 个元素的序列$\nabla(S)$,将$\nabla(S)$上所有 $n!$个不同置换所组成的集合记作$\nabla(S)_n$.

定义 4(序复合). 在$\nabla(S)_n$ 上定义二元运算$\square$,对任意$\alpha=\{x_1,x_2,\ldots,x_n\}$,$\beta=\{y_1,y_2,\ldots,y_n\}\in\nabla(S)_n$,$\alpha\square\beta=\{\alpha[\beta[1]],\alpha[\beta[2]],\ldots,\alpha[\beta[n]]\}$,$\square$称为序复合运算.

例如,设 $S=\{30,5,10,26,9\}$,则有$\nabla(S)=\{5,1,3,4,2\}$;又设$\alpha=\{3,2,4,1,5\}$,$\beta=\{2,5,1,4,3\}\in\nabla(S)_n$,则$\alpha\square\beta=\{2,5,3,1,4\}$.

定理 2. $\langle\nabla(S)_n,\square\rangle$是一个群,$\square$是$\nabla(S)_n$上的序复合运算.

证明:

(1) 由定理 1,二元运算$\square$在$\nabla(S)_n$上是封闭的;

(2) $\forall\alpha,\beta,\gamma\in\nabla(S)_n$,$\forall t\in\nabla(S)$,设$\gamma(t)=x$,$\beta(x)=y$,$\alpha(y)=z$,由于

$$\alpha\square\beta[x]=\alpha[\beta[x]]=\alpha[y]=z.$$

所以有,

$$(\alpha\square\beta)\square\gamma(t)=(\alpha\square\beta)[\gamma(t)]=(\alpha\square\beta)[x]=z.$$

同样地,由于

$$\beta\square\gamma[t]=\beta[\gamma[t]]=\beta[x]=y,$$

所以,

$$\alpha\square(\beta\square\gamma)(t)=\alpha[\beta\square\gamma[t]]=\alpha[y]=z.$$

因此,$(\alpha\square\beta)\square\gamma=\alpha\square(\beta\square\gamma)$;

(3) 设$\pi_e=\{1,2,3,\ldots,n\}\in\nabla(S)_n$,$\forall\alpha\in\nabla(S)_n$,有

$$\alpha\square\pi_e=\{\alpha[\pi_e[1]],\alpha[\pi_e[2]],\ldots,\alpha[\pi_e[n]]\}=\{\alpha[1],\alpha[2],\ldots,\alpha[n]\},$$

$$\pi_e\square\alpha=\{\pi_e[\alpha[1]],\pi_e[\alpha[2]],\ldots,\pi_e[\alpha[n]]\}=\{\alpha[1],\alpha[2],\ldots,\alpha[n]\}.$$

因此,$\alpha\square\pi_e=\pi_e\square\alpha$,$\nabla(S)_n$中存在幺元$\pi_e$;

(4) $\forall\alpha\in\nabla(S)_n$,必定存在着对应的$\alpha^{-1}\in\nabla(S)_n$,使得$\alpha\square\alpha^{-1}=\alpha^{-1}\square\alpha=\pi_e$. □

定理 3. 给定初始无序的序列 S,对其按非降序排列的过程就是求$\nabla(S)$逆元的过程.

设将序列 S 中所有元素按非降序排列的结果为 T 序列,由定理 2 有$\nabla(T)=\pi_e(\in\nabla(S)_n)$.基于此,有定理 3 成立.

3 排序类算法生成模型

基于以上代数性质,我们分析排序类算法的共性和可变性,确定了排序算法领域的实现构件以及它们之间的依赖性,建立了体现算法间共性、参数化算法间差异性的算法程序生成模型,并使用 PAR 及算法形式化开发策略[36]给出了构件的实现.

3.1 领域分析和设计

为确定起见,本文只对非降序排序问题进行讨论,它可描述为:将给定序列 $a[0:n-1]$中所有元素按非降序排列.首先为该问题构造了一个如下所示的形式化算法规约:

Specification1: sorting

|[**in** n: integer; **out** $a[0:n-1]$: list of integer; **aux** $b[0:n-1]$: list of integer]|

AQ1: $n\geqslant 0\wedge a=b$;

AR1: $sort(a,0,n-1)\equiv ord(a,0,n-1)\wedge perm(a[0:n-1],b[0:n-1])$,

$ord(a,0,n-1)\equiv(\forall k{:}0\leqslant k<n-1{:}a[k]\leqslant a[k+1])$,

$perm(a[0:n-1],b[0:n-1])\equiv(\forall i{:}0\leqslant i<n{:}(\mathrm{N}j{:}0\leqslant j<n{:}a[j]=a[i])=(\mathrm{N}k{:}0\leqslant k<n{:}b[k]=a[i]))$;

这里,b 是一个辅助变量,刻画了待排序序列的初始状态;a 刻画了终止状态,即结果序列.实际上,b 和 a 可看成是对不同状态下同一个序列的描述.后置断言中,$sort(a,0,n-1)$的定义可理解为:对序列$a[0:n-1]$排序,且排序结果仍存于 a;$ord(a,0,n-1)$表示结果序列$a[0:n-1]$有序;$perm(a[0:n-1],b[0:n-1])$使用了计数量词 N 来表示两个大小相等的序列 a 和 b 互为置换,即即对于任意的 i,$0\leqslant i<n$,序列 a 中值等于 $a[i]$的元素个数和序列 b 中值等于 $a[i]$的元素个数相同.

基于定理 3,对给定的初始序列 S 排序,也即求$\nabla(S)$逆元的过程可表示为

$$S\xrightarrow{\pi_1'}S_1\xrightarrow{\pi_2'}S_2\xrightarrow{\pi_3'}\ldots\xrightarrow{\pi_m'}S_m,$$

$$\nabla(S)\xrightarrow{\pi_1}\nabla(S_1)\xrightarrow{\pi_2}\nabla(S_2)\xrightarrow{\pi_3}\ldots\xrightarrow{\pi_m}\nabla(S_m)=\pi_e,$$

其中,$\pi_i'\in S_n$,$\pi_i\in\nabla(S)_n$,$i=1,2,\ldots,m$.因此,求$\nabla(S)$逆元的过程,实际上是通过对 S 的逐步置换,最后达到其序抽取结果为幺元的状态的过程.其中,应用不同的策略和技巧就可以获得不同的中间过程.

把一个序列分成若干子序列,构成一个分段序列,这是有关序列运算中一个重要技巧,它可以把序列的运算转化为若干子序列的运算,使运算更为简明.在上述求$\nabla(S)$逆元的过程中,可先将 S 分划成若干子序列,然后通过对子序列的逐步置换使得 S 达到最终状态.而分划的方式则具多样性,或者按设定的子序列个数和大小将 S 直接分段,或者将 S 预处理后再分划.对应到求解排序问题的过程,就是先将原问题分划成更小的子问题,然后通过求解子问题来求解原问题.而问题的分划则通常是将原问题固定分划或函数分划成两个子问题来完成.固定分划可以事先确定子问题的规模及内容,函数分划之前没有办法确定子问题的规模或内容,而是由某个分划函数

的执行结果确定.为便于讨论,我们将固定分划求解排序问题的方式称为 DBP(determined bi-partition)方式.即在某一个固定的位置将输入的待排序序列分裂成两个子序列,再分别对这两个子序列排序,从而将排序问题固定分划成两个可预先确定的子问题.通过求解子问题,并合并它们的解来求解原问题;将函数分划求解排序问题的方式称为 UBP(uncertain bi-partition)方式,即引入一个函数来分划出两个子问题,从而达到求解原问题的目的.

H-增量排序采用了较特殊的固定分划方式,它将待排序序列按某个递减变量 h 分裂成若干个小组,距离为 h 的元素在同一个小组中,然后对同组元素进行排序.当 h 递减到 1 时,整个序列构成一个小组,对其排序即得到原问题的解.

所用问题分划不同,求解子问题所得到的有序子序列和原问题解的关系也不同.DBP 方式求解得到的子问题的解各自有序,而互相之间无序,尚需将子解归并成原问题解;UBP 方式下,由子解得到原问题解的方法则取决于分划函数的性质;H-增量排序中,子解已直接构成原问题的解.在这个算法问题求解的过程中,包含着对序列的分划操作、有序子序列的合并操作等操作,对其数据抽象可形成一个用于排序的序列类型.此外,UBP 方式下的堆排序是在序列结构的基础上组建一个堆进行问题求解.

接下来,我们为上述分析中确定的概念和功能引入构件.广义上,可将构件定义为可复用的、较为独立的软件单元,它可以有不同的大小和分类,即构件具有不同的粒度.根据上述对排序算法程序特定领域的分析,我们将领域构件分为类型构件和算法构件,前者抽象出排序领域算法对序列的基本操作,如将序列分割为子序列、有序性判断、有序子序列的合并、序列输入、输出等,并将数据及其上的这些操作封装成抽象数据类型;后者则是对序列排序的功能抽象,其中会使用到类型构件提供的操作.为了提高构件的可复用程度,我们将构件的可变部分参数化,即设计泛型化的类型构件和算法构件.

根据上述 3 种求解排序问题的方式,我们设计了 3 个泛型算法构件 DBPSort,UBPSort 和 HSort,并将它们使用的数据和基本操作封装成两个类型构件 SortingList 和 Heap.泛型算法构件 DBPSort 使用 DBP 方式来求解排序问题,而 UBPSort 构件则使用 UBP 方式.HSort 使用某个增量序列划分原序列并对同组内的元素进行排序,最终达到求解原问题的目的.类型构件 SortingList 提供分裂、合并、划分、h-增量插入等排序算法用到的基本操作,并负责存储元素;Heap 构件根据堆的性质实现了有关操作.它们均可在 PAR 平台预定义类型构件的基础上实现.用一条带箭头的连线表示一个构件对另一个构件的依赖关系,则上述构件间的依赖关系可描述成如图 1 所示,体现的主要思想是:一个给定层的构件需要下一层构件的某些服务.例如:Hsort,DBPSort 和 UBPSort 都使用到 SortingList 提供的某些服务,而 UBPSort 还会使用到 Heap;SortingList 和 Heap 均使用序列构件 list 的服务,此外, Heap 还使用二叉树类型构件 btree.

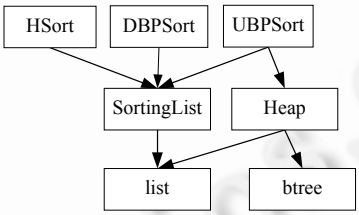

Fig.1 Dependencies between components

图 1 构件依赖关系

PAR 平台 Apla 语言提供了自定义 ADT 机制和泛型程序设计机制,可以方便地描述领域概念,本文采用它定义和实现构件建立排序算法领域的特定语言.

3.2 模型形式化构建

一个层的构件需要从低于它的直接或者间接层构件中获取信息,根据图 1,我们首先构建底层构件.

list 为 Radl 和 Apla 中的预定义抽象数据类型.设 S 表示一个序列,则有 $S[h]$,$S[t]$分别表示序列 S 的头元素和尾元素,整型变量 h 和 t 为指针;$S[i]$表示序列 S 中的第 i 个元素;$\#S$ 计算序列 S 的元素个数;S:=[],将序列 S 置为

空;$S\uparrow T$ 为并置运算,将序列 T 连接到序列 S 的后面,合成一新序列;S:=[e]↑S,S:=S↑[e],S:=S[h…i−1]↑[e]↑S[i...t],分别将表达式 e 的值作为一元素插在序列 S 的最前面、最后面及 S[i−1]的后面;S:=S[h+1...t],S:=S[h...i−1]↑S[i+1…t],分别删除序列 S 的头元素及第 i 元素 S[i].

btree 为预定义二叉树类型.设 t 为一个二叉树,则相关运算有:t:=%,表示将 t 置为空树;$t.d$ 产生二叉树 t 的根结点值;$t.l$ 和 $t.r$ 分别产生二叉树 t 的左子树和右子树.

这两种类型已在 PAR 平台中实现.对于其他构件,我们基于 PAR 平台及相关形式化推导策略来实现它们,从而保证构件的可靠性.

3.2.1 类型构件

我们使用 Apla 中的自定义 ADT 机制,基于预定义抽象数据类型 list 定义了 SortingList 类型,并结合 Hoare 公理化方法给出了它的形式化规约.

Hoare 公理化方法用前置断言和后置断言来定义 ADT 的每个操作,这些定义构成一组公理,可用来验证使用该类型的程序的正确性,保证用户正确使用该 ADT 及其操作,也为 ADT 的具体实现提供了依据.为了解决使用前置断言和后置断言时遇到的新类型尚未定义的困难,Hoare 公理方法首先给出所定义类型的抽象表示,然后用前、后置断言来刻画类型操作对于这种抽象表示的性质和行为.

SortingList 抽象表示为序列类型 list,允许使用类型变量 *elem* 来参数化元素类型,其长度 *size* 若缺省,则表示 SortingList 为无界序列(详见附录 A).SortingList 中包含系列操作,其中的创建排序表、输出排序表等子过程的实现比较简单,可以直接给出,而归并、分裂、划分等函数的实现将根据规约变换策略[36]由从规约出发的形式化推导来得到.这里,我们仅概要列出 *ordmerge* 的形式化开发过程.

为简单起见,记 *ordmerge* 函数返回的结果为 a,输入的排序表为 $a1$,

$$ord(a,0,n-1)\equiv(\forall k{:}0\leqslant k<n-1{:}a[k]\leqslant a[k+1]),$$

即有 $a[i{:}j]=ordmerge(a1,i,s,j)$.对于 *ordmerge* 规约中的 *perm*,我们有:

$perm(a1,a)$

≡{将量词 N 转换到∑}$(\forall i{:}0\leqslant i<n{:}(\sum j{:}0\leqslant j<n\wedge a1[j]=a[i]{:}1)=(\sum k{:}0\leqslant k<n\wedge a[k]=a[i]{:}1))$

≡{在 $0\leqslant s\leqslant n-1$ 点范围分裂}$(\forall i{:}0\leqslant i\leqslant n-1{:}(\sum j{:}0\leqslant j\leqslant s\wedge a1[j]=a[i]{:}1)+(\sum j{:}s+1\leqslant j\leqslant n-1\wedge a1[j]=a[i]{:}1)=$
$(\sum k{:}0\leqslant k\leqslant n-1\wedge a[k]=a[i]{:}1))$

≡{对两个∑量词分别应用范围分裂、单点范围分裂}$(\forall i{:}0\leqslant i\leqslant n-1{:}(\sum j{:}a1[0]=a[i]{:}1)+(\sum j{:}1\leqslant j\leqslant s\wedge a1[j]=$
$a[i]{:}1)+(\sum j{:}a1[s+1]=a[i]{:}1)+(\sum j{:}s+2\leqslant j\leqslant n-1\wedge a1[j]=a[i]{:}1)=(\sum k{:}0\leqslant k\leqslant n-1\wedge a[k]=a[i]{:}1))$

≡{对∀量词范围分裂.令∀量词的函数部分为 allF}$(\forall i{:}i=0{:}allF)\wedge(\forall i{:}1\leqslant i\leqslant n-1{:}allF)$

≡{单点范围分裂}$(\sum j{:}a1[0]=a[0]{:}1)+(\sum j{:}1\leqslant j\leqslant s\wedge a1[j]=a[0]{:}1)+(\sum j{:}a1[s+1]=a[0]{:}1)+$
$(\sum j{:}s+2\leqslant j\leqslant n-1\wedge a1[j]=a[0]{:}1)=(\sum k{:}0\leqslant k\leqslant n-1\wedge a[k]=a[0]{:}1)\wedge(\forall i{:}1\leqslant i\leqslant n-1{:}allF)\Leftarrow$
{若 $a1[0]\leqslant a1[s+1]$,则 $a[0]=a1[0]$;若 $a1[0]>a1[s+1]$,则 $a[0]=a1[s+1]$}
$(a1[0]\leqslant a1[s+1]\wedge a[0]=a1[0]\wedge perm(a1[1{:}s]\uparrow a1[s+1{:}n-1],a[1{:}n-1]))\vee$
$(a1[0]>a1[s+1]\wedge a[0]=a1[s+1]\wedge perm(a1[0{:}s]\uparrow a1[s+2{:}n-1],a[1{:}n-1]))$

将上述结果代入 *ordmerge* 规约,有:

$a[0{:}n-1]=ordmerge(a1,0,s,n-1)$
$\Leftarrow(ord(a1,0,s)\wedge ord(a1,s+1,n-1)\rightarrow ord(a,0,n-1))\wedge$
$((a1[0]\leqslant a1[s+1]\wedge a[0]=a1[0]\wedge perm(a1[1{:}s]\uparrow a1[s+1{:}n-1],a[1{:}n-1]))\vee$
$(a1[0]>a1[s+1]\wedge a[0]=a1[s+1]\wedge perm(a1[0{:}s]\uparrow a1[s+2{:}n-1],a[1{:}n-1])))$
⇐{合取对析取的分配律,卷叠}$(a1[0]\leqslant a1[s+1]\wedge a[0]=a1[0]\wedge a[1{:}n-1]=ordmerge(a1[1{:}s],a1[s+1{:}n-1]))\vee$
$(a1[0]>a1[s+1]\wedge a[0]=a1[s+1]\wedge a[1{:}n-1]=ordmerge(a1[0{:}s],a1[s+2{:}n-1]))$

使用分支来描述上述结果,得到一般情况下 *ordmerge* 的递推关系:

$a[i{:}j]=ordmerge(a1[i{:}s],a1[s+1{:}j])\Leftarrow$if $a1[i]\leqslant a1[s+1]$ then

$a[i]=a1[i]\wedge a[i+1:j]=ordmerge(a1[i+1:s],a1[s+1:j])$ else

$a[i]=a1[s+1]\wedge a[i+1:j]=ordmerge(a1[i:s],a1[s+2:j]),0\leqslant i\leqslant j\leqslant n-1,s=(i+j)/2$

递推关系体现了归并两个有序段成一个有序段的算法思想,据此可以得到其 Radl 算法和 Apla 程序.在预定义抽象数据类型 btree 的基础上,我们实现了带类型参数的泛型最小堆 Heap,这里均不再详列.

3.2.2 算法构件

从 Specification1 出发,我们使用 PAR 方法,按 DBP 方式形式化推导出泛型算法构件 DBPSort 的实现,并由推导过程得到其中泛型参数的约束.此外,按照 UBP 方式或 H-增量排序的分划方式,可分别形式化推导出 UBPSort 和 HSort 的实现.这里,概要给出 DBPSort 的开发过程,对于后两者仅于附录列出开发结果.

引进一个 *split* 函数来给出分裂的位置,其规约如下:

Specification2: split

|[**in** a: SortingList; **out** s: integer]|

AQ2: $a\neq[]$;

AR2: $s=split(a[0:n-1])$ s.t. $a.h\leqslant s<a.t$.

则原问题分划为 $sort(a,0,n-1)\equiv F(sort(a1,0,s),sort(a1,s+1,n-1))$.这里:$F$ 是待确定的函数;$a1$ 是一个中间序列变量,表示序列 a 的中间状态.

假设 $sort(a1,0,s)$和 $sort(a1,s+1,n-1)$这两个子问题已求解,接下来,我们通过规约变换来构造 F:

$$\begin{aligned}
&sort(a,0,n-1)\equiv ord(a,0,n-1)\wedge perm(a[0:n-1],b[0:n-1])\\
&\quad\equiv\{\text{将 } b \text{ 序列在 } s \text{ 处分成两个子序列}\}ord(a,0,n-1)\wedge perm(a[0:n-1],b[0:s]\uparrow b[s+1:n-1])\\
&\quad\Leftarrow\{\text{引入 } a1 \text{ 来表示序列 } a \text{ 的中间状态,满足 } perm(a,a1)\}\\
&\quad\quad ord(a,0,n-1)\wedge perm(a1[0:s],b[0:s])\wedge perm(a1[s+1:n-1],b[s+1:n-1])\wedge perm(a,a1)\\
&\quad\Leftarrow\{ord(a1,0,s)\wedge ord(a1,s+1,n-1)\wedge(ord(a1,0,s)\wedge ord(a1,s+1,n-1)\rightarrow ord(a,0,n-1))\}\\
&\quad\quad ord(a1,0,s)\wedge ord(a1,s+1,n-1)\wedge(ord(a1,0,s)\wedge ord(a1,s+1,n-1)\rightarrow ord(a,0,n-1))\wedge\\
&\quad\quad perm(a1[0:s],b[0:s])\wedge perm(a1[s+1:n-1],b[s+1:n-1])\wedge perm(a,a1)\\
&\quad\equiv\{\text{卷叠}\}sort(a1,0,s)\wedge sort(a1,s+1,n-1)\wedge(ord(a1,0,s)\wedge ord(a1,s+1,n-1)\rightarrow ord(a,0,n-1))\wedge perm(a,a1)
\end{aligned}$$

引入一个新函数 *merge*,满足上述结果中除 *sort* 外的谓词,并将 *merge* 函数的结果存放在序列 a 中,则有递推关系:

$$sort(a,0,n-1)\Leftarrow sort(a1,0,s)\wedge sort(a1,s+1,n-1)\wedge a[0:n-1]=merge(a1[0:s],a1[s+1:n-1]) \quad (1)$$

$a[0:n-1]=merge(a1[0:s],a1[s+1:n-1])$, s.t. $(ord(a1,0,s)\wedge ord(a1,s+1,n-1)\rightarrow ord(a,0,n-1))\wedge perm(a1,a)$

我们详细列出 merge 函数的规约如下:

Specification3: merge

|[**in** n, s: integer; $a1[0:n-1]$: SortingList; **out** $a[0:n-1]$: SortingList]|

AQ3: $n\geqslant 0\wedge 0\leqslant s<n\wedge a=a1$;

AR3: $a[0:n-1]=merge(a1,0,s,n-1)\equiv(ord(a1,0,s)\wedge ord(a1,s+1,n-1)\rightarrow ord(a,0,n-1))\wedge perm(a1[0:n-1],a[0:n-1])$;

这里的 *ord* 和 *perm* 谓词的定义与 Specification1 中的相同.

将 *split* 和 *merge* 函数定义为操作参数,在公式(1)、*split* 规约及 *merge* 规约的基础上,可以得到泛型 Radl 算法(略).接下来,我们使用循环不变式开发新策略,开发 DBPSort 算法程序的循环不变式,并为获得高效率的非递归 Apla 程序做好准备.将每个子问题表示成有序对$(i:j)$的形式,$(i:j)$确定了进行排序的序列段范围,并将它看成一个长度为 2 的序列$[i,j]$.定义两个序列变量 q 和 S,q 序列长度为 2,$q[h]$和 $q[t]$分别存放当前待排序序列段的起始位置和结束位置,即 $sort(a,q[h],q[t])$为当前正准备解决的子问题;S 是一起堆栈作用的序列变量,用于存放没有排序而尚待排序的序列段.S 的内容由下面定义的函数 F 递归给出:

(1) $F([])=[]$;

(2) $F(q\uparrow S)=sort(a,q[h],q[t])\rightarrow merge(a[q[h]:q[t]],F(S))$.

这里的箭头"→"表示先求解子问题 *sort*(*a*,*q*[*h*],*q*[*t*]),再将其解 *a*[*q*[*h*]:*q*[*t*]]与 *F*(*S*)的解归并得到有序结果.根据所得的递推关系以及上述定义,我们可构成如下所需的循环不变式:

$$\rho\text{: } sort(a,0,n-1)\equiv a[0:n-1]=merge(a[0:q[h]-1],sort(a,q[h],q[t])\rightarrow merge(a[q[h]:q[t]],F(S))).$$

由 Radl 算法和循环不变式,可归纳得到非递归的 DBP 分划求解排序问题的 Apla 抽象程序(见附录 B).

通过提供具体的问题分划函数和合并函数,分别实例化 DBPSort 的分划函数参数 *split* 和合并函数参数 *merge*,就可以生成系列不同的基于 DBP 分划的排序算法程序.为保证泛型程序实例化的正确性,必须对泛型参数进行约束.抽象排序算法推导过程提供的 *split* 函数规约 Specification2 和 *merge* 函数的规约 Specification3 就起到约束泛型参数的作用.即只有分别满足这两个规约的具体子程序才能用来替换 DBPSort 中的子程序参数.

算法构件 UBPSort 带有一个分划函数参数 *partition*,该参数规约来自于形式化推导过程,起到对泛型参数约束的作用.通过形式化推导可得到基于 UBP 分划求解排序问题的非递归 Apla 抽象子过程(见附录 B).

算法构件 HSort 将计算增量序列的函数和组内元素排序的函数定义为操作参数(见附录 B).

3.3 小结与实例

抽象是对付复杂性的有效方法.基于定理 3,我们借助领域工程的概念和方法,对排序类算法进行分析和抽象,将算法间的差异性,如问题分划方式、子解的组合等,作为泛型构件的参数来定义泛型算法构件,并使用前后置断言描述参数的性质和行为;而将使用到的数据类型以及基本操作抽象成泛型类型构件,建立了约束条件下由类型构件实例化算法构件生成算法程序的模型,有效支持领域算法程序生成.我们确定并通过 PAR 方法实现了带分划和归并两个函数参数的泛型算法构件 DBPSort、带一个分划函数参数的泛型算法构件 UBPSort、实现增量排序的泛型算法构件 Hsort、带类型参数的泛型 SortingList 类型构件以及带类型参数的泛型 Heap 类型构件.

在得到 Apla 描述的构件后,容易使用 PAR 平台将其变换成 Java,C++等可执行语言级构件,从而建立高可靠领域构件库.使用 Apla 提供的泛型实例化机制组合这些构件,以参数替换的方式可自动生成排序领域的问题求解算法程序.

这里,我们给出一个特定排序算法生成的实例.使用 SortingList 中提供的计算增量函数 *geth*2 和分组选择函数 *hselect* 实例化算法构件 HSort,可生成使用增量序列{*n*/2,*n*/4,…,1}来分组待排序序列,并采用选择排序对组内元素排序的增量选择排序算法 selectHsort1,加上类型定义、变量定义、输入/输出语句等程序要素,调用 selectHsort1 进行排序的 Apla 算法程序示例如下:

```
program selectHsort;
const n=10;
ADT intSortingList: new SortingList(integer,n);
var intL: intSortingList;
procedure selectHsort1: new HSort(intL.geth2,intL.hselect);
begin
    intL:=intL.create(·);
    selectHsort1(intL);
    intL.output(intL);
end.
```

一个函数只要满足 HSort 中参数 *geth* 的规约,就可以用来计算增量序列,从而基于 HSort 产生不同的增量排序算法.下面给出另外两种:

- **procedure** *selectHsort*2: new *HSort*(*intL.geth*22,*intL.hselect*);
- **procedure** *selectHsort*3: new *HSort*(*intL.knuthgeth*,*intL.hselect*);

这 3 种算法均根据增量计算函数提供的增量 *h* 将待排序序列中等距离者放入一个小组,并使用选择排序对同组元素排序.下面我们仅对 selectHsort1 算法作简要的分析.

首先,对于 SortingList 中 *h*-增量选择函数 hselect 的实现,较之选择排序,差别在于循环体中变量变化的步长为 *h* 而不是 1.因此,元素间的比较次数是一个有关 *h* 的函数 *f*.

$$f(h)=\frac{n-1}{h}+\frac{n-2}{h}+\ldots+\frac{n-1-(n-1-h)}{h}=\frac{(n-1)+(n-2)+\ldots+h}{h}=\frac{(n-h)(n-1+h)}{2h}=\left(\frac{n^2-n}{2}\right)\frac{1}{h}-\frac{1}{2}h+\frac{1}{2}.$$

增量计算函数 *geth*2 产生 *h* 递减序列{*n*/2,*n*/4,...,1},因此,selectHsort1 算法的总比较次数 *T*(*n*)为

$$f\left(\frac{n}{2}\right)+f\left(\frac{n}{4}\right)+\ldots+f(1)=\left(\frac{n^2-n}{2}\right)\left(\frac{2}{n}+\frac{4}{n}+\ldots+1\right)-\frac{1}{2}\left(\frac{n}{2}+\frac{n}{4}+\ldots+1\right)+\frac{\log_2 n}{2}=n^2-\frac{5n-\log_2 n+3}{2}.$$

此外,增量选择排序算法 selectHsort1 是不稳定的.例如,当 *n*=4、初始序列为 5,5,2,6 时,经过 selectHsort1 排序后,前后两个 5 的位置将颠倒过来.

该算法是经形式化开发得到的算法构件 HSort 和类型构件 SortingList 满足泛型约束下的组合而产生的,从而有效地保证了该算法的正确性.我们通过 Apla-Java 程序生成系统自动生成了相应的 Java 程序,经实际运行检测,程序的运行结果与预期一致.

显然,基于 HSort 构件,提供以其他方法来计算增量的函数,还可以生成这里未列出的分组排序算法.

图 2 展示了部分排序算法程序的生成过程.可以看出,同样的实际参数,通过不同的组合可生成不同的具体算法,说明我们的模型具备较强的生成能力.

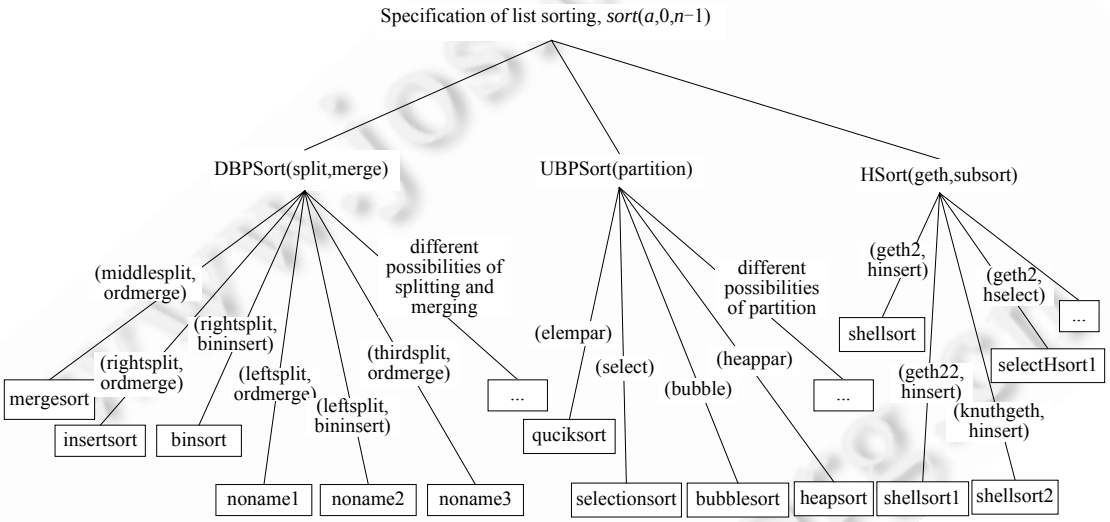

Fig.2 Generation process for some of the sorting algorithmic programs

图 2 部分排序算法程序的生成过程

我们使用 PAR 和相应的算法形式化开发策略来形式化地设计和生成泛型类型构件及算法构件,并得到其中泛型参数的约束.由此,使用满足泛型参数约束的形式化开发的子程序替换经形式化推导得到的泛型算法中的参数,并在扩充的 Apla-Java 程序生成系统的支持下生成特定的排序算法程序,可以保证所生成算法程序的正确性.

4 系统支持的算法生成

Apla-Java 程序生成系统的目标是将 Apla 语言描述的程序自动转换成 Java 程序.它由转换器和构件库组成,目前支持自定义 ADT、泛型程序设计等机制.我们对其进行了扩充,以支持基于上述模型的特定排序算法生成.

首先,我们将形式化方法开发的排序算法领域的 Apla 构件,经 Apla-Java 程序生成系统转换为 Java 构件,以自扩展的方式建立一个称为 sortinglib 的排序算法领域构件库.

在转换器部分,扩充对新类型的识别以及对泛型构件的识别,并支持用以组合基本构件的泛型实例化语句以及实例化后所得子程序的运行,使得在后台 sortinglib 构件库的支持下,通过书写抽象简洁的 Apla 语句,就可以自动生成一个排序算法,从而简化了用户的工作量,显著提高了算法的开发效率和可靠性.

在平台的支持下,我们自动生成了上述基于模型的各种特定排序算法的 Java 程序.经实际运行检测,程序的运行结果均与预期相符.使用算法生成模型及系统来为排序问题生成系列具体的算法程序,不但可以提高算法程序的可靠性和设计效率,而且通过揭示算法设计的决策,清晰地表达了算法怎样完成给定的任务,获得很好的可读性和可维护性.

5 总 结

本文将形式化技术、泛型和生成式程序设计技术以及抽象数据类型机制综合应用于排序算法生成领域,进行了形式化方法制导下的排序类算法自动生成的研究.通过刻画排序问题的代数性质,分析排序类算法的共性和可变性,设计并形式化开发了泛型类型构件和算法构件,以参数的形式定义了排序领域算法间存在的差异性,并使用前后置断言作为泛型约束机制刻画了其中操作参数的性质和行为,建立了领域特定语言和算法生成模型.在扩充的 Apla-Java 程序生成系统的支持下,以参数替换的方式自动生成了归并排序、快速排序、堆排序、Shell 排序等 20 余种典型的已知算法,以及增量选择排序等未见于现有文献的算法.通过高可靠上层框架和底层构件满足约束条件下的组合及系统的支持,有效保证了生成算法程序的正确性,提高了算法程序的设计效率.而通过十几种实际参数替换就可生成 20 余种算法,则说明了我们的模型和系统具备较强的生成能力.本文生成的特定排序算法中包括多种无名算法,并且还存在着开发本文所列之外其他算法的可能.这说明该种生成途径确实可以用于生成新的算法,而不只是解释或证明一些已知算法.

我们已将该排序算法的生成方法成功地拓展应用于以置换为基础的其他类问题以及搜索问题,说明了本文的算法自动生成方法扩展到其他类问题的有效性,并有望发展成求解一系列类似问题的方法.进一步的工作还包括由算法生成模型实现 GUI,使得用户可通过勾选构件的方式来获取所需的特定算法.

References:

[1] Upson S. Computer software that writes itself. Newsweek Int'l, 2005-12-25.

[2] Anthes G. In the labs: Automatic code generators. ComputerWorld, 2006-3-20.

[3] McLaughlin L. Automated programming: The next wave of developer power tools. IEEE Software, 2006,5-6:91−93. [doi:10.1109/MS.2006.66]

[4] Gray J. What next? A dozen information-technology research goals. Journal of the ACM, 2003,50(1):41−57. [doi: 10.1145/602382.602401]

[5] Batory D. Thoughts on automated software design and synthesis. In: Proc. of the FSE/SDP Workshop, Future of Software Engineering Research (FoSER 2010). New York: ACM Press, 2010. 29−32. [doi: 10.1145/1882362.1882369]

[6] Knuth DE. The Art of Computer Programming, Vol.3: Sorting and Searching. 2nd ed., Read: Addison Wesley, 1998.

[7] Xue JY. A unified approach for developing efficient algorithmic programs. Journal of Computer Science and Technology, 1997,12(4): 314−329. [doi: 10.1007/BF02943151]

[8] Xue JY. PAR method and its supporting platform. In: Proc. of the 1st Asian Working Conf. on Verified Software (AWCVS 2006). 2006. 29−31.

[9] Backhouse RC. Program Construction and Verification. London: Prentice-Hall, 1986.

[10] Dromey RG. Derivation of sorting algorithms from a specification. The Computer Journal, 1987,30(6):512−518.

[11] Dromey RG, Billington D. Stepwise program derivation. Technical Report, SQI-91-02, Software Quality Institute, Griffith University, 1991.

[12] Clark KL, Darlington J. Algorithm classification through synthesis. The Computer Journal, 1980,23(1):61−65. [doi: 10.1093/comjnl/23.1.61]

[13] Merritt SM, Lau KK. A logical inverted taxonomy of sorting algorithms. In: Kuru S, Caglayan MU, Akin HL, eds. Proc. of the 12th Int'l Symp. on Computer and Information Sciences (ISCIS'97). 1997. 576−583.

[14] Broy M. Program construction by transformations: A family tree of sorting programs. In: Biermann AW, Guiho G, eds. Proc. of the Computer Program Synthesis Methodologies. Holland: D. Reidel Publishing Company, 1983. 1−49. [doi: 10.1109/32.21743]

[15] Borges PR, Ravelo J. Formal construction of a sorting algorithm. Technical Report, CI-1993-003, Department of Computation, University of Simón Bolívar, 1993.

[16] Bird RS. A calculus of functions for program derivation. In: Turner DA, ed. Proc. of the Research Topics in Functional Programming. Read: Addison-Wesley, 1990. 287−307.

[17] Almeida JB, Pinto JS. Deriving sorting algorithms. Technical Report, DI-PURe-06.04.01, Department of Information, University of Minho, 2006.

[18] Guttmann WN. Deriving an applicative heapsort algorithm. Technical Report, UIB-2002-02, University of Ulm, 2002. [doi: 10068/129135]

[19] Guttmann WN, Partsch H. Tool support for the interactive derivation of formally correct functional programs. Journal of Universal Computer Science, 2003,9(2):173−188. [doi: 10.3217/jucs-009-02-0173]

[20] Ward M. Derivation of a sorting algorithm. Technical Report. Computer Science Department, Durham University, 1999.

[21] Smith DR. Mechanizing the development of software. In: Broy M, Steinbrüggen R, eds. Proc. of the Calculational System Design. Amsterdam: IOS Press, 1999.

[22] Smith DR. Generating programs plus proofs by refinement. In: Meyer B, Woodcock J, eds. Proc. of the Verified Software: Theories, Tools, Experiments. LNCS 4171, Heidelberg: Springer-Verlag, 2008. 182−188. [doi: 10.1007/978-3-540-69149-5_20]

[23] Leavens GT, Abrial JR, Batory D, Butler M, Coglio A, Fisler K, Hehner e, Jones C, Miller D, Peyton-Jones s, Sitaraman M, Smith DR, Stump A. Roadmap for enhanced languages and methods to aid verification. In: Jarzabek S, Schmidt D, Veldhuizen T, eds. Proc. of the 5th Int'l Conf. on Generative Programming and Component Engineering (GPCE 2006). New York: ACM Press, 2006. 221−236. [doi: 10.1145/1173706.1173740]

[24] Czarnecki K, Eisenecker UW. Generative Programming: Methods, Tools, and Applications. Read: Addison-Wesley, 2000.

[25] Idate S, Patil SH, Mali DJ. Automated code generation using generative programming approach for a mathematical expression. In: Proc. of the Int'l Conf. on Systemics, Cybernetics and Informatics. Pentagram Research Centre Limited, 2008. 134−137.

[26] Nedunuri S, Cook WR. Synthesis of fast programs for maximum segment sum problems. In: Siek J, Fischer B, eds. Proc. of the 8th Int'l Conf. on Generative Programming and Component Engineering (GPCE 2009). New York: ACM Press, 2009. 117−126. [doi: 10.1145/1621607.1621626]

[27] Rayside D, Benjamin Z, Singh R, Joseph PN, Milicevic A, Jackson D. Equality and hashing for (almost) free: Generating implementations from abstraction functions. In: Fickas S, Atlee J, Inverardi P, eds. Proc. of the 31st Int'l Conf. on Software Engineering (ICSE 2009). California: IEEE Computer Society Press, 2009. 342−352. [doi: 10.1109/ICSE.2009.5070534]

[28] Batory D, Hofner P, Kim J. Feature interactions, products, and composition. In: Denney E, Schultz U, eds. Proc. of the 10th Int'l Conf. on Generative Programming and Component Engineering (GPCE 2011). New York: ACM Press, 2011. 13−22. [doi: 10.1145/2047862.2047867]

[29] Li YL, Novak G. Generation of geometric programs specified by diagrams. In: Denney E, Schultz U, eds. Proc. of the 10th Int'l Conf. on Generative Programming and Component Engineering (GPCE 2011). New York: ACM Press, 2011. 63−72. [doi: 10.1145/2047862.2047874]

[30] Fu JC, Bastani FB, Yen I. Iterative planning in the context of automated code synthesis. In: Chang C, Belli F, Mei H, McMillin B, eds. Proc. of the 31st Annual IEEE Int'l Computer Software and Applications Conf. (COMPSAC 2007). California: IEEE Computer Society Press, 2007. 251−259. [doi: 10.1109/COMPSAC.2007.132]

[31] Xue JY, Davis R. A simple program whose derivation and proof is also. In: Proc. of the 1st IEEE Int'l Conf. on Formal Engineering Method (ICFEM'97). California: IEEE Computer Society Press, 1997. 132−139. [doi: 10.1109/ICFEM.1997.630419]

[32] Gries D, Xue JY. The hopcroft-tarjan planarity algorithm, presentations and improvements. Technical Report, 88-906, Computer Science Department, Cornell University, 1988.

[33] Xue JY, Yang B, Zuo ZK. A linear in-situ algorithm for the power of cyclic permutation. In: Preparata FP, Wu X, Yin JP, eds. Proc. of the 2nd Int'l Frontiers of Algorithmics Workshop (FAW 2008). LNCS 5059, Heidelberg: Springer-Verlag, 2008. 113−123. [doi: 10.1007/978-3-540-69311-6_14]

[34] Zheng YJ. Formal calculation of highly-dependable material support algorithms based on PAR [Ph.D. Thesis]. Beijing: Institute of Software, the Chinese Academy of Sciences, 2009 (in Chinese with English abstract).

[35] Zuo XL, Li WJ, Liu YC. Discrete Mathematics. Shanghai: Shanghai Scientific and Technological Literature Publishing House, 1982 (in Chinese).

[36] Shi HH, Xue JY. PAR-Based formal development of algorithms. Chinese Journal of Computers, 2009,32(5):982−991 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2009.00982]

附中文参考文献:

[34] 郑宇军.基于 PAR 的高可信装备保障算法形式化推演[博士学位论文].北京:中国科学院软件研究所,2009.

[35] 左孝凌,李为鑑,刘永才.离散数学.上海:上海科学技术文献出版社,1982.

[36] 石海鹤,薛锦云.基于 PAR 的算法形式化开发.计算机学报,2009,32(5):982−991. [doi: 10.3724/SP.J.1016.2009.00982]

附录 A. 类型构件规约

SortingList 构件的部分规约如下:

define ADT *SortingList*(***sometype*** *elem*,[*size*]);

type *SortingList=private*;

function *create*(): *SortingList*;

R: *create=list*(*elem*);

function *ordmerge*(*L*:*SortingList*;*l*,*s*,*r*:*integer*):*SortingList*;

Q: $L\neq[]\wedge l\leqslant s\leqslant r\wedge(\forall k{:}l\leqslant k<s{:}L[k]\leqslant L[k+1])\wedge(\forall k{:}s+1\leqslant k<r{:}L[k]\leqslant L[k+1])$

R: $(\forall k{:}l\leqslant k<r{:}\ ordmerge[k]\leqslant ordmerge[k+1])\wedge perm(ordmerge,L[l{:}r])$

function *elempar*(*L*:*SortingList*;*l*:*integer*;*var p*:*integer*;*r*:*integer*):*SortingList*;

Q: $L\neq[]\wedge l<r$

R: $l\leqslant p\leqslant r\wedge(\forall k{:}l\leqslant k\leqslant p-1{:}elempar[k]\leqslant elempar[p])\wedge(\forall k{:}p+1\leqslant k\leqslant r{:}elempar[p]\leqslant elempar[k])\wedge$
$perm(elempar,L)$

…

function *geth*2(*seed*:*integer*,*L*:*SortingList*):*integer*;

Q: $L\neq[]\wedge seed\geqslant 1$

R: $geth2=seed/2$;

function *geth*22(*seed*:*integer*,*L*:*SortingList*):*integer*;

Q: $L\neq[]\wedge seed\geqslant 1$

R: $geth22=seed/2.2$;

function *knuthgeth*(*seed*:*integer*,*L*:*SortingList*):*integer*;

Q: $L\neq[]\wedge seed\geqslant 1$

R: $(seed=\#(L)\wedge knuthgeth\leqslant seed/3)\vee(seed\neq\#(L)\wedge knuthgeth=(seed-1)/3)$

function *hinsert*(*L*:*SortingList*;*h*:*integer*):*SortingList*;

Q: $L\neq[]\wedge h\geqslant 1$

R: $(\forall i{:}h\leqslant i\leqslant\#(L)-1{:}hinsert[i-h]\leqslant hinsert[i])$

function *hselect*(*L*:*SortingList*;*h*:*integer*):*SortingList*;

Q: $L\neq[]\wedge h\geqslant 1$

R: $(\forall i{:}0\leqslant i\leqslant\#(L)-h-1{:}hselect[i]\leqslant hselect[i+h])$

function *shellsublist*(*L*:*SortingList*;*h*:*integer*):*SortingList*(*SortingList*);

Q: $L\neq[]$

R: $(\forall i,j{:}0\leqslant i<h\wedge 0\leqslant j<shellsublist[i].t-1{:}(\exists m,n{:}0\leqslant m<n<L.t\wedge L[m]=$
$shellsublist[i][j]\wedge L[n]=shellsublist[i][j+1]{:}(n-m)=h))$

procedure *output*(*L*:*SortingList*);

Q: $L\neq[]$;

enddef;

implement ADT *SortingList*(***sometype*** *elem*);

type *SortingList=list*(*elem*);

…

endimp.

附录 B. 算法构件实现

(1) 算法构件 DBPSort 的 Apla 实现

```
procedure DBPSort(somefunc split(b:SortingList):integer;somefunc merge(b:SortingList;l,s,r:integer):
                  SortingList;var a:SortingList;left,right:integer);
var p: integer; q: list(integer,2); S: list(list(integer,2));
begin
    S,q[h],q[t]:=[],left,right;
    do (q≠[]∧q[h]<q[t])→p:=split(q);q[t],S:=p,[p+1,q[t]]↑S;
        [](q≠[]∧q[h]≥q[t])→a[left:q[t]],q:=merge(a,left,q[h],q[t]),[];
        [](q=[]∧S≠[])→q,S:=S[h],S[h+1...t];
    od;
end;
```

(2) 算法构件 UBPSort 的 Apla 实现

```
procedure UBPSort(somefunc partition(b:SortingList;l:integer;var p:integer;r:integer):SortingList;var a:
                  SortingList;left,right:integer);
var p: integer; q: list(integer,2); S: list(list(integer,2));
begin
    S, q[h], q[t]:= [], left, right;
    do (q[h]<q[t])→a[q[h]:q[t]]:=partition(a,q[h],p,q[t]);
                  q[t],S:=p−1,[[p+1,q[t]]]↑S;
    [](q[h]≥q[t]∧S≠[])→q,S:=S[h],S[h+1...t];
    od;
end;
```

(3) 算法构件 HSort 的 Apla 实现

```
procedure HSort(somefunc geth(seed:integer;b:SortingList):integer;somefunc subsort(b:SortingList;h:integer):
                SortingList;var a:SortingList);
var d: integer;
begin
    d:=geth(#(a),a);
    do (d≥1)→a:=subsort(a,d);d:=geth(d,a);od;
end.
```

石海鹤(1979－),女,江西乐平人,博士,副教授,CCF 学生会员,主要研究领域为软件形式化与自动化.

薛锦云(1947－),男,教授,博士生导师,CCF 高级会员,主要研究领域为软件形式化与自动化,面向服务的软件工程.