Apla语言中并发分布式机制的研究

游 珍[1,2] 薛锦云[1,2] 应 时[1]
(武汉大学软件工程国家重点实验室 武汉 430072)[1]
(江西师范大学省高性能计算技术重点实验室 南昌 330022)[2]

摘 要 从并发分布式程序设计的角度,对现有的并发分布式语言进行分析比较,选取Jayadev Misra教授近几年提出的全新结构化并发分布式语言Orc作为研究对象。通过深入分析Orc语言的基本原理和语言特征,提出了一个能够适合Apla抽象程序设计语言的并发分布式机制,设计了并发算子、并发语句、进程定义、进程通信和进程同步,并通过实例探讨了该设计方案的可行性和实用性,最后阐述了Apla语言中并发分布式机制具有通用性强、抽象层次高、简单易懂、便于并发分布式程序的开发等优点。

关键词 并发分布式程序设计,Orc语言,Apla抽象程序设计语言,并发分布式机制
中图法分类号 TP312 **文献标识码** A

Research on Concurrent and Distributed Mechanism of Apla Language

YOU Zhen[1,2] XUE Jin-yun[1,2] YING Shi[1]
(State Key Lab of Software Engineering, Wuhan University, Wuhan 430072, China)[1]
(Provincial Key Laboratory for High-Performance Computing Technology, Jiangxi Normal University, Nanchang 330022, China)[2]

Abstract From the viewpoint of concurrent and distributed programming, several concurrent and distributed programming languages were analyzed and compared. A novel structured distributed and concurrent language Orc was designed and implemented by Professor Jayadev Misra. After analyzing the fundamental principles and language-characters, a new concurrent and distributed mechanism of Apla (Abstract Programming LAnguage) was originally designed in the paper. The mechanism includes the concurrent operator, concurrent statements, the definition, communication and synchronization of process. The feasibility and practicability of the mechanism were illustrated by giving a representative example. Finally, many advantages (such as generality, simplicity, abstract, easy-to-writing) of the concurrent and distributed mechanism were presented.

Keywords Concurrent and distributed programming, Orc language, Apla language, Concurrent and distributed mechanism

1 引言

高性能计算技术和互联网技术的迅猛发展必将引领软件历史性地向并发和分布式方面靠拢,并发分布式计算成为当今热点研究课题。C++标准委员会主席Herb Sutter在2005年就提出了"免费午餐已经结束——软件历史性地向并发靠拢[1]"。并发分布式计算将开启新的可能,让软件功能更多、能力更强。

并发分布式程序设计的困难主要源于并发计算机体系结构的多样性、并发程序执行的不确定性,以及由于缺乏统一有效的计算机模型而导致并发程序的复杂性、低效性、不可移植性等问题。目前一些主流语言(如Java、C#等)具有并发分布式程序设计功能,但是由于需要涉及过多细节使得用起来过于复杂,从而影响了它们的实用性。本文选取最具有代表性和创新性的Orc语言作为主要研究对象;通过深入分析其基本原理和语言特征,提出了一个与Apla顺序抽象程序设计语言融合的并发分布式机制,并详细探讨了该设计方案的可行性和实用性;最后总结了其优点。

2 并发分布式程序设计语言概述

在计算机科学中,并发(Concurrency)是计算机系统的一种性质,该系统由多个能够同时执行且相互交互的计算(Computations)组成;这些计算可以在多核或多线程中执行,也可以在物理上分布的多个处理器上执行。并发程序设计是(Concurrent Programming)在多道程序设计的基础上发展起来的。分布式计算(Distributed Computing)研究如何把一个

到稿日期:2011-02-15 返修日期:2011-06-16 本文受国家自然科学基金项目(60573080,60773054),国家自然基金重大国际合作研究项目(61020100609),科技部国际科技合作项目(2008DFA11940),江西师范大学青年成长基金项目(3174)资助。

游 珍(1982-),女,博士生,讲师,主要研究方向为软件形式化验证和并发分布式计算,E-mail:youzhen.jxnu@yahoo.com.cn;薛锦云(1947-),男,教授,博士生导师,主要研究方向为软件形式化与自动化;应 时(1965-),男,教授,博士生导师,主要研究方向为面向服务的软件工程、高可信软件开发方法、语义软件开发技术、云计算时代的软件开发方法。

需要非常巨大的计算能力才能解决的问题分成许多小的部分，然后把这些部分分配给许多计算机进行处理，最后把这些计算结果综合起来得到最终的结果。采用分布式程序设计(Distributed Programming)时，一个程序由分布于系统各结点上的程序模块组成，这些模块可以同时执行，它们通过通信互相协调和交换数据。

1968年，E. W. Dijkstra首先引入了并发程序设计的概念，并研究了有关的同步和死锁问题。20世纪70年代人们开始将有关并发程序设计的概念引入程序设计语言中，出现了并发PASCAL、并发C、Modula、Ada和Occam等并发程序设计语言。并发程序设计语言(Concurrent Programming Language)[2]的主要特征是引入了进程描述，提供了进程同步、进程通信和进程产生等功能。严格地说，这里的进程是指进程的描述，非进程本身。并发分布式程序设计语言的研究内容包括并发模型、并发分布式程序设计语言的设计与实现等。

并发模型(Models of Concurrency)[2]是指描述并发系统行为的数学模型，可以分为两个层次：①描述性的并发模型(λ演算、时序逻辑、CSP、CCS、Petri网等)；②语义性的并发模型：每一种并发模型被提出时，通常都附有一种给定的语义，或者是一开始没有明确的语义，后来有人赋予它一种语义。例如，R. Milner为CCS配备了交叠式语义，而C. A. Petri为Petri网规定了真并发语义。

根据交互通信方式的不同，并发程序设计语言可以分成两类：①基于共享内存的并发程序设计语言(如Java、C#等)；②基于消息传递的并发程序设计语言(如Erlang、Occam等)。形式化规范语言RSL(RAISE Specification Language)[3]中的并发机制是基于“进程代数”——类似于CSP和CCS；它不仅提供了对并发系统定义规范的方法，而且提供了连接符(如并行连接符、外部/内部选择连接符、内锁连接符等)描述并发计算的表达式，还提供了“channel”作为通信原语。并发求精语言Circus[4]结合了CSP、Z语言和Dijkstra卫式语言的特点，是一种能够对并发分布式系统进行规范描述、设计和编程的语言。与图灵奖获得者Tony Hoare共同提出Verified Software[5]巨大挑战的计算机科学家Jayadev Misra对并发程序设计语言有深入研究，1998年他与K. M. Chandy教授设计出基于三元组{P}C{Q}的并行设计模型及其逻辑系统UNITY[6]；2001年提出了面向对象的并发程序设计模型Seuss[7]。Jayadev Misra当前主要研究一种全新的结构化并发分布式程序设计语言Orc[8-10]。语言实现者宣称Orc是一种通用程序设计语言，既能设计顺序程序，也能够设计并发和分布式程序，还能处理网络中服务的通信、同步和协作。

3 Apla语言

PAR方法[11-13]及其支撑平台是我们研究团队在13个国家级课题的连续资助下自主研发的通用软件设计方法和自动化工具。它由规约和算法描述语言Radl、抽象程序设计语言Apla、规约变换规则、Radl到Apla生成器、Apla到C++、Java、Delphi和VB等可执行程序的生成工具、系统的程序设计方法学构成。作为一种简单、实用、统一的算法设计及形式化推导和证明的方法，PAR方法和PAR平台提供的语言、变换规则和系列转换工具使得开发算法程序具有原理简单、使用方便、通用性强，可靠性高等特点，可以大幅度提高复杂算法程序的生产效率。

Apla(Abstract Programming LAnguage)语言[14]体现了数据抽象、功能抽象、重载、泛型等现代程序设计思想，使用了传统的数学符号和数学表达式，简单实用，便于程序开发。因此，使用Apla语言编写的程序易于阅读理解和验证，也便于转换成各种可执行程序设计语言程序。我们已经使用Apla语言开发了许多算法程序。它们中有简单问题，也有复杂问题；有数值计算问题，也有非数值计算问题；其中最具说服力的是成功地解决了计算机大师Knuth提出的一个挑战性问题[15]和图灵奖获得者Hopcroft与Tarjan的图平面性算法[16]。

3.1 丰富的数据类型

Apla语言的一个重要特色是其灵活丰富的数据类型系统：(1)标准数据类型(如整型integer、实型real、布尔型boolean、字符型char)；(2)自定义简单类型(如记录类型record、数组类型array)；(3)预定义ADT(如集合类型set、表类型list、二叉树类型btree、图类型diagraph)；(4)自定义ADT和泛型ADT机制(用户可以像使用标准数据类型一样使用预定义ADT)。

3.2 程序结构

Apla语言类似于Ada语言，是一种基于对象的抽象程序设计语言。Apla的程序结构如下所示，其中〈参数表〉是为支持泛型程序设计而设置的，允许数据、类型、子程序、构件、服务等作为程序的参数。

```
program 〈程序名〉[〈参数表〉]
  [〈常量说明〉;]
  [〈类型说明〉;]
  [〈变量说明〉;]
  [〈过程与函数说明〉;]
begin
  〈语句序列〉;
  end
```

Apla语言提供的程序语句有赋值语句、选择语句、循环语句和遍历语句等。Apla语言可以自定义过程和函数，允许递归，也允许类型和子程序等作过程和函数的参数。

```
procedure 〈过程名〉(形参表);
  ...
begin
  ...
end;
function 〈函数名〉(形参表):类型
  ...
begin
  ...
end;
```

4 新型并发分布式程序设计语言Orc

作为当今世界上并发程序设计领域的先驱，Jayadev Mis-

ra 教授先后提出了 UNITY[6] 和 Seuss[7] 两种并发程序设计语言。为了适应互联网技术、分布式计算技术、面向服务技术的发展，Orc 语言应运而生。作为一种新型的并发分布式程序设计语言，Orc 既是通用目标程序设计语言(General Purpose Programming Language)，又是网络脚本语言(Web Scripting Language)，还是可执行规范语言(Executable Specification Language)[10]。

4.1 Orc 语言的基本组成

Orc 由纯函数式语言 Cor(a pure functional language)、外部服务(external services)、连接符(combinator)和含有大量预定义函数/sites 的内库(a large library)组成。

4.1.1 Cor 表达式和 Orc 表达式

“表达式(expression)”是 Orc 语言中非常重要的概念。一个 Cor 程序可以是一个表达式，一个 Orc 程序也可以是一个表达式。Cor 表达式和 Orc 表达式的主要区别是：①Cor 是 Orc 的子集，所以 Cor 表达式也是 Orc 表达式；②Cor 表达式被求值，Orc 表达式被执行；③Cor 表达式每次求值得到一个确定的结果，Orc 表达式的执行具有随机性，例如执行同一表达式可能运行不同的行为，调用不同的服务，或者得到不同的响应等。

4.1.2 外部服务

Orc 表达式通过“site”来处理外部服务[9]。这些外部服务包括顺序计算、数据处理、web-service 通信、用户交互的代理服务器等。一个 web-service 就是一个 site[8]，它可以在客户机上运行，也可以在服务器上运行。

4.1.3 连接符

Orc 表达式可分为原子表达式(如执行一个计算；返回一个值；调用 web-service 等)和复合表达式(原子表达式通过连接符递归地构造出的复杂表达式)。Orc 语言提供了 4 个连接符[9]：并行连接符(Parallel Combinator)、顺序连接符(Sequential Combinator)、剪枝连接符(Pruning Combinator)和否则连接符(Otherwise Combinator)。关于 4 种连接符的详细分析和比较参见表 1。

表 1 连接器的比较和分析

连接符	功能	表达式结束条件	满足性质	优先级	实例
并行 F|G	并行执行 F 和 G	F 和 G 都完成	交换律：F|G≡G|F 结合律：F|G|H≡(F|G)|H≡F|(G|H)	②	1+0 | 1+1 | 1+2 发布 1,2,3(顺序任意)
顺序 F>x>G	F 的所有值通过 x 传递给 G	F 所有求值完成且实例化的 G 求值完成	右结合律：F>x>G>y>H≡F>x>(G>y>H)	①	((4, true) | (5, false) | (6, true)) >(x, true)>x 发布 4 和 6
剪枝 F<x<G	G 的第一个值通过 x 传递给 F	F 发布一个值且实例化的 G 求值完成	左结合律：F<x<G<y<H≡(F<x<G)<y<H	③	x+2 <x< (3 | 4) 发布 5 或者 6
否则 F;G	当 F 不能发布值时才求解 G	F 发布值并完成；或 F 和 G 都求值完成	结合律：F;G;H≡(F;G);H≡F;(G;H)	④	1+2;1+3 发布 3

4.1.4 内库

Orc 语言具有丰富的内库，提供了大量预定义的 site(如 array、semaphore、channel 等)和预定义的函数(如 each、fold、repeat 等)，也允许用户自定义 site 和函数。

4.2 Orc 语言的进程表示

Orc 语言的进程也使用“表达式”来描述。“信道 channel”可以被其他的表示式共享[8]，使用最频繁的“信道 channel”是 Buffer，它是异步的 FIFO 信道，提供了两个基本操作 get 和 put，其中 c. get()表示从信道 c 中取出并发布第一个值或者当信道 c 为空时阻塞等待；c. put(v)表示把 v 作为最后一项放到信道 c 中并发布一个信号。

例如：进程 P 从输入信道 c 中读取数据 x，再通过调用名称为 Computer 的 site 对该数据进行处理，然后把计算结果放到输出信道 e 内；重复上述操作。进程 P 表示为：

$$P(c,e) \triangleq c.\ \text{get} > x > \text{Compute}(x) > y > e.\ \text{put}(y) >> P(c,e)$$

进程网络(Process Network)由多个进程通过“连接器”组合而成。例如：进程网络 N 表示从分别独立地从两个输入信道 c 和 d 输入数据，再通过 Compute 计算后写入输出信道 e。则进程网络 N 表示为：

$$N \triangleq P(c,e) \mid P(d,e)$$

4.3 通过 site 来处理服务

“site”是 Orc 程序中的基本计算单元(fundamental unit of computation)，类似于其他语言中的函数和子程序，但具有远程性(remote)和不确定性(unreliable)[17]，这两个特性充分体现了 Orc 语言具有并发和分布式计算的机制。Orc 语言 site 内库中提供的预定义 semaphore 和 lock 可以作为同步控制算子来有效地避免并发分布式程序设计中由于资源共享带来的死锁和互斥等问题。

site 调用的格式类似于函数调用的格式，由 site 名称和参数序列组成。参数可以是常量、变量或者 site 本身。一个 site 调用最多得到一个响应[9]。如下为 site 调用的基本格式：

Site-name (parameters-list)

Orc 语言也允许用户自定义 site。创建 site 有两种方式：① 通过引入 Java 类来创建 site，这些 site 可以同时被 Orc 代码和 Java 代码共享使用；② 通过使用 Orc 底层的 API 来创建 site，这些 site 只能被 Orc 代码使用。

下面是 site 访问外部服务的具体实例。

```
include "search. inc"
each(results)
〈results〈
      Prompt("Search for:") 〉term〉
  (Yahoo(term) | Google(term) )
//用户输入需要搜索的内容 term
//调用 Yahoo 和 Google 两个 site 进行搜索
```

4.4 Orc 程序

Orc 程序通常由一个定义集合(a set of definitions)和一个需要运行程序来求解的目标表达式(a goal expression)组成。

下面是对数组 a 进行快速排序的 Orc 程序[18]。

```
def quicksort(a)=
 def swap(x,y)=a(x)? >z> a(x):=a(y)? >> a(y):=z
 def part(p,s,t)=
```

```
def lr(i)=if i<t && a(i)? <=p then lr(i+1) else i
def rl(i)=if a(i)? > p then rl(i-1) else i
(lr(s),rl(t)) >(s',t')>
( if (s'+1<t') >> swap(s',t') >> part(p,s'+1,t'-1)
| if (s'+1=t') >> swap(s',t') >> s'
| if (s'+1 > t') >> t'
)
def sort(s,t)=
  if s >=t then signal
  else part(a(s)?,s+1,t) >m> swap(m,s) >>
    (sort(s,m-1),sort(m+1,t)) >> signal
    sort(0,a. length()-1)
```

5 Apla 语言中并发分布式机制的研究

Apla 是一种抽象的顺序程序设计语言，它具有数据抽象、功能抽象和支持泛型程序设计等特征；PAR 平台提供的自动生成工具可以实现把 Apla 抽象程序转换成某一种可执行程序（如 Java，C++，C#，VB，Delphi 等），这种设计思想不仅减轻了使用者编写程序的工作量，而且避免了开发支持 Apla 语言的解释器或编译器。目前，Apla 语言在科研、教学、软件开发和软件服务外包等领域也得到了广泛的应用。

通过分析和理解 Orc 并发分布式程序设计语言的基本原理；探索性地提出一种并发分布式机制的设计方案，并分析了该机制与现有 Apla 语言的融合情况，从而使得 Apla 语言不仅能够开发顺序程序，而且可以开发并发分布式程序。

5.1 并发算子和并发量词

“简单实用”是 PAR 方法和 PAR 平台的主要特征，也是我们刻意追求的。为了延续该指导思想，在相对比较成熟的 Apla 抽象程序设计语言中实现并发性，仅添加一个二元并发算子“||”，该并发算子可以实现多种形式的并发执行，包括语句的并发执行，函数/过程的并发执行和进程的并发执行。并发算子“||”对应的并发量词为“|||”。其语法使用 BNF 表示：

① Con-part :: = Statement | Fun-call | Proc-call | Process

含义：并发执行的部分可以是程序语句、函数/过程调用或者进程。

② Con-exe:: =Con-part || Con-part

含义：并发执行的部分通过“并发算子||”进行连接。

③ Quantified-Con-exe :: = (||| variable-list: Boolean-exp:Con-part)

含义：并发量词“|||”可以看作是二元算子“||”的一般化形式。

并发算子“||”满足如下性质：

- 交换律：P||Q ≡ Q||P
- 结合律：P||Q||R ≡ (P||Q)||R ≡ P||(Q||R)

5.2 Apla 语言原顺序语句的并发性

Apla 抽象程序设计语言中语句的格式类似于 Dijkstra 卫式命名语言，但它限制了非确定性，因此，Apla 算法程序转换后的具体可执行程序是串行执行过程。下面讨论 Apla 顺序语句并发性的设计方案。

(1) 多重赋值语句的并发性

(||| $i:1\leqslant i\leqslant n:x_i:=e_i$)

(2) 选择语句的并发性

if $C_1\rightarrow S_1$

|| $C_2\rightarrow S_2$

…

|| $C_n\rightarrow S_n$

fi

当有两个条件 C_i 和 C_j 同时为真时，并发地执行其对应的语句组 S_i 和 S_j。

(3) 循环语句的并发性

do $C_1\rightarrow S_1$

|| $C_2\rightarrow S_2$

…

|| $C_n\rightarrow S_n$

od

当有两个条件 C_i 和 C_j 同时为真时，并发地执行其对应的语句组 S_i 和 S_j。

(4) 遍历语句的并发性

(|||〈遍历控制变量〉:〈范围〉:〈语句组〉)

5.3 Apla 语言中进程设计

与经典的程序设计相比，并发分布式程序设计的两个特点是分布和通信。分布是指程序分成若干个可独立执行的模块，它们分布在不同的结点上。在 Apla 语言中引入“进程”来表示独立执行的模块。由于分布在各个结点上的程序模块是相互关联的，它们在执行过程中需要交换数据和互相协调，因此通信是必不可少的，即在 Apla 语言中通过定义“信道”来完成进程之间的通信。进程的同步功能主要靠“互斥锁”来实现同步机制。

5.3.1 进程定义

参考 Apla 语言中函数和过程定义的风格，使用关键字“process”来定义进程，每个进程内部的代码分布在不同的结点上独立执行。进程定义的基本格式是：

```
process〈进程名〉(形参表);
  …
begin
  …
end;
```

5.3.2 进程通信

Apla 语言提供了预定义的 List 抽象数据类型，经过适当的改造就可以让 List 具备“信道”的所有功能。使用关键字“channel”来定义信道 c，参考 CSP 的通信机制，$c?x$ 表示信道 c 等待接收数据 x，$c!x$ 表示信道 c 中输出数据 x。

5.3.3 进程同步

进程通信主要通过访问共享数据来完成。当两个进程需要使用同一个共享数据时，存在交叉操作而破坏数据的可能性。因此，有可能出现两种错误：进程干扰和内存一致性错误。为了避免这些错误，引入“互斥锁”作为同步机制。

按照约定，需要对共享数据进行独占和一致性访问的进程，在进行访问之前，必须获得该数据的互斥锁，访问操作完

成之后必须释放互斥锁。在从获得锁到释放锁的时间段内，进程被称为拥有互斥锁。只要有进程拥有互斥锁，其他进程就不能获得同一个互斥锁，试图获得互斥锁的其他进程将被阻塞。

使用同步关键字“sync”来支持 Apla 语言中的互斥锁。sync 关键字可以放在变量的前面、函数/方法的前面、信道的前面、进程的前面等。

其基本用法如下：

```
sync(提供互斥锁的对象)
    { 临界代码 }
```

5.4 实例

如下为 5 位哲学家就餐问题的并发 Apla 程序。

```
program Dining-Philosophers;
sync var C0,C1,C2,C3,C4:channel(boolean);
//定义哲学家进程
process P(i:interger,u:channel(boolean),v:channel(boolean))
var x,a,b:boolean
begin
  do u! a=ture∧v! b=ture→
       x:=false;u? x||v? x;
       eating();x:=ture;u? x||v? x;
   ||¬(u! a=ture∧v! b=ture) → u? a||v? b;
od
end;
//5 个哲学家进餐主程序
begin
C0,C1,C2,C3,C4:=[true],[true],[true],[true];
do ture →
  P(0,C0,C1)||P(1,C1,C2)||P(2,C2,C3)||P(3,C3,C4)||P(4,C4,C0);
od
end.
```

结束语 并发分布式计算是未来计算机科学发展的方向，如何设计一个简单而实用的抽象并发分布式程序设计语言正成为一个研究热点。通过分析比较大量现有并发分布式语言，我们选取最具有代表性和创新性的 Orc 语言作为主要研究对象。Orc 既是并发分布式程序设计语言，又是 Web-service脚本语言，还是可执行的规范语言。

论文的主要贡献是深入剖析了 Orc 的基本原理和语言特征，提出了一个能够与 Apla 顺序抽象程序设计语言融合的并发分布式机制，并详细探讨了该机制的可行性和实用性。

与其他并发分布式语言相比，本文 Apla 语言并发分布式机制的优点主要体现在以下几个方面：①并发的通用性强，通过统一的并发算子“||”来实现多种形式的并发性；②尽量保持原 Apla 顺序程序设计的风格，进程定义的格式类似于函数/过程定义的格式；③借鉴 Orc 语言和 CSP 提出了进程、信道、同步和互斥的设计方案；④整个并发分布式机制抽象层次高、简单易懂、便于程序的开发。

同时，我们也意识到使用该并发分布式机制时需要程序员来考虑并发执行的兼容性和死锁等问题。在今后的工作中，将通过在 Apla 语言的内库中添加一些兼容性判定策略或同步算子（如 PV、Semaphore 等）来简化用户编程的工作量。作为一种形式化方法，PAR 方法使用分划、递推和谓词变换等技术能够对顺序程序进行严谨的形式化推导和证明，因此，笔者也希望从程序正确性角度考虑，深入探索并发分布式程序的形式化推导和验证。

参 考 文 献

[1] Sutter H. The Free Lunch is Over: A Fundamental Turn Toward Concurrency in Software[J]. Dr. Dobb's Journal, 2005, 30(3): 202-210

[2] 张效详. 计算机科学技术百科全书[M]. 北京：清华大学出版社，1998

[3] George C W. The RAISE Specification Language: A Tutorial[C]// Proceedings of VDM'91, volume 551 of Lecture Notes in Computer Science. Springer-Verlag, 1991

[4] Woodcock J C P, Cavalcanti A L C. A Concurrent Language for Refinement[C]// Butterfield A, Pahl C, eds. IWFM'01: 5th Irish Workshop in Formal Methods. Dublin, Ireland, July 2001

[5] Hoare T, Misra J. Verified software: theories, tools, experiments [C]// LNCS 4171. Springer Verlag, July 2005: 1-18

[6] Chandy K M, Misra J. Parallel Program Design: A Foundation [M]. Addison-Wesley Publishing Co., 1988

[7] Misra J. A Discipline of Multiprogramming: A Programming Theory for Distributed Applications [M]. Springer-Verlag, 2001

[8] Misra J, Cook W R. Computation Orchestration: A Basis for Wide-Area Computing [J]. Journal of Software and Systems Modeling, March 2007

[9] Kitchin D, Quark A. William Cook and Jayadev Misra. The Orc Programming Language [C] // Proceedings of FMOODS/FORTE, LNCS 5522. Springer Verlag, 2009: 1-25

[10] Orc Language Project [OL]. http://orc.csres.utexas.edu/index.shtml, 2011-04-21

[11] Xue Jin-yun. A Unified Approach for Developing Efficient Algorithm of Programs [J]. Journal of Computer Science and Technology, 1997, 12(4)

[12] Xue Jin-yun. A practicable approach for formal development of algorithmic programs[C]// Proc of the International Symposium on Future software Technology 99'. Nanjing China, 1999: 158-160

[13] 薛锦云，等. 程序设计方法学[M]. 北京：国家高等教育出版社，2002

[14] 薛锦云. 抽象程序设计语言 Apla 报告[R]. 江西师范大学技术报告. 2001

[15] Xue Jin-yun, Davis R. Simple Program Whose Derivation and Proof is Also[C]// Proceedings of the International Conference on Formal Engineering Methods, ICFEM. 1997: 132-139

[16] 谢武平. Hopcroft-Tarjan 图平面性算法在 Apla 语言中的设计与实现[R]. 江西师范大学技术报告. 2006

[17] Orc in 15 Minutes [OL]. http://orc.csres.utexas.edu/tutorial.shtml, 2011-04-21

[18] Kitchin D, Quark A, Misra J. Quicksort: Combining Concurrency, Recursion, and Mutable Data Structures. Reflections on the Work of C. A. R. Hoare[M]. A Festschrift in honor of his 75th birthday. Springer-Verlag, 2010