# PAR: A Practicable Formal Method and Its Supporting Platform

Jinyun Xue(✉), Yujun Zheng, Qimin Hu, Zhen You, Wuping Xie, and Zhuo Cheng

State International S&T Cooperation Base of Networked Supporting Software, Jiangxi Normal University, Nanchang 330022, China
jinyun@vip.sina.com

**Abstract.** The use of formal methods can significantly improve the reliability, correctness and efficiency of software development. Although formal methods has been invented for more than 40 years, but academia and industry do not have a unified understanding of what are formal methods and its essential characteristics. Formal methods has not been recognized and widely applied by academia and industry. The authors of this paper have long been engaged in the study of the essential features of Formal methods. The authors propose a new definition: **Formal methods are a strict technology based on mathematics and tool support for software and hardware system, including high-level abstract specification, modeling language and different levels of model transformation tools.** Based on this definition, this paper develops a practicable formal methods and its supporting platform, called ***PAR method*** and ***PAR platform***, *short for PAR*. PAR consists of the following elements: requirement modeling language SNL, algorithm modeling language Radl, abstract program modeling language Apla, a set of rules for the model transformation and a set of automatic transformation tools from requirement models to algorithm models, to abstract program models and to executable programs. The goal of the transformations is to generate executable program. The elements embody 6 innovative ideas given in Sect. 2. There are two kinds of applications of PAR. One is that many nontrivial algorithms and programs have been developed formally. Another is formal developing several safety-critical information systems.

**Keywords:** Formal methods · Par platform · Modeling language
Model transformation · Formal specification

## 1 Introduction

The use of Formal methods can significantly improve the reliability, correctness and efficiency of software development [4]. Although Formal methods has

This work was funded by the NSF of China under Grant No. 61662036, 61472167, 61462041, 61272075, 61020106009, 60773054, 60573080, 60273092, 69983003, 69783006; MOST of China Grant No. 2008DFA11940, 2003CCA02800.


J. Sun and M. Sun (Eds.): ICFEM 2018, LNCS 11232, pp. 70–86, 2018.
https://doi.org/10.1007/978-3-030-02450-5_5

been invented for more than 40 years [1,3,5,14], but academia and industry do not have a unified understanding of what is Formal methods and its essential characteristics. Woodcock [22] thought "formal methods are mathematical techniques, often sup-ported by tools, for developing software and hardware systems". Bjørners opinion [4] is "by a formal method we shall understand a method whose techniques and tools can be explained in mathematics". The definition by Wikipedia [45] is "formal methods are a particular kind of mathematically based techniques for the specification, development and verification of software and hardware systems". For these reasons, Formal methods has not been recognized and widely applied by academia and industry. Gargantini [8] thought "many practitioners are still reluctant to adopt formal methods. Besides the well-known lack of training, this skepticism is mainly due to: the complex notations; the lack of easy to-use tools supporting a developer during the life cycle activities of system development". Bjørner said [4] that when ask ourselves the question: Have formal methods for software development in the current sense been successful? Our answer is, regretfully, no! He thought "The academic and industry obstacles can be overcome. Still, a main reason for formal methods not being picked up, and hence "more" successful, is the lack of scalable and practical tool support."

The above situation shows that the main reason for the failure of Formal methods to be applied in large scale is that the language provided by the existing formal methods is very complex and difficult to learn. The tools for supporting formal methods are simple and cannot provide effective help for the users of the methods. We think the main reason for this is that the degree of abstraction-level of formal languages is not high enough. On formal methods support tool, people are not clear about its main goal. At present, the support tool of formal methods can only form a formal verification for the related components developed by software (program). It's far from enough.

The authors of this paper have long been engaged in the study of the essential features of formal methods. The authors propose a new definition: **Formal methods are a strict technology based on mathematics and tool support for software and hardware system, including high-level abstract specification, modeling language and different levels of model transformation tools. The specification and modeling language should be as abstract as possible and reflect different levels of abstraction according to the characteristics of software and hardware. All models and model transformation tools should be verified formally or automatically.**

Based on this definition, this paper develops a practicable formal methods and its supporting platform, called PAR method and PAR platform, short for PAR. PAR consists of the following elements: requirement modeling language SNL, algorithm modeling language Radl, abstract program modeling language Apla, a set of rules for the model transformation and a set of automatic transformation tools between requirement model, algorithm model, abstract program model and executable program. The goal of the transformations is to generate executable program. One of the distinct features of the PAR platform is the

agile genericity mechanisms. In PAR not only a value, a data type and an ADT can be generic parameter, and a computing-action (including operator, method, function and procedure, transaction, subsystem and web service, etc.) can be generic parameter also. The elements embody 6 innovative ideas given in the Sect. 2.

The first one is that many nontrivial algorithms and programs have been developed formally, formal derivation or formal proof, including graph algorithms [29], travel tree algorithms [32], array section algorithms [25,26], Knuths famous hard problem of cyclic permutation [10,24,33]. The abstract Hopcroft-Tarjan planarity algorithm was described in Apla. The Apla program was transformed by PAR platform to C++ code that can correctly test the planarity and generate improved planar embedding [10]. A more convincing example is formal development of Knuth's challenging program that converts a binary fraction to decimal fraction with certain condition [13,27,28].

The second one is to develop several safety-critical information systems, including shuttle transportation problems [36], discrete optimization algorithm design [37], fire evacuation [38], population classification in earthquakes [39], Emergency railway transportation planning [40], active services support for disaster rescue [41], Airline passenger profiling [42] and Industrial Accident Early Warning [43], etc. A general distributed transaction processing system is implemented in PAR and used in student information management system [23].

In Sect. 2, we give a brief description of key ideas and innovative techniques of PAR. The third Section describes the main elements of PAR method and PAR. Two kinds of applications of PAR and two cases study described in Sect. 4. Related work is described in Sect. 5. Finally, conclusions and future research are presented.

## 2 Key Ideas and Innovative Techniques of PAR

Compared with other formal methods and program generating system, say B method [1], RAISE [18], Kestrelwares [19], Orc [6,12,16], rCOS, [15,21], NDAUTO [44] and NDADAS [44], etc.,s the PAR method and PAR platform have following innovative techniques.

### 2.1 A Unified Approach for Designing Algorithm Based on Quantifier Transformation

The efficiency of an algorithm is mainly influenced by the method of algorithm design and implementation, the data structures and programming language of describing algorithm. In PAR [26], we put emphasis on the method of algorithm design and implementation. In general, it is easier to design algorithms using enumeration or exhaustive search, but the algorithms have low efficiency; in contrast, it is more difficult to design efficient algorithm using effective design method. Generally speaking, using traditional method, e.g. dynamic programming, greedy, divide-and-conquer, etc., one can get efficient algorithm, but the

difficult is in choosing suitable one. Since implementing algorithm using iteration has higher efficiency than using recursion, we stick to iteration rather than recursion in our methodology. We generalize the recurrence relation concept of a sequence of numbers (difference equation) to problem solving sequence and propose a unified approach for designing algorithmic program [26]. It covers several existed algorithm design techniques including divide-and-conquer, dynamic programming, greedy, enumeration and some nameless methods. The designer of algorithms using PAR method can partly avoid the difficulty in making choice among various existed design methods. Using the approach, we have formally developed many nontrivial algorithmic programs, including graph algorithms [29], travel tree algorithms [32].

### 2.2 A New Representation of Algorithms

The most important notion in PAR is recurrence relation of problem solving sequence, short for recurrence. Based on the recurrence relation, the structure of an algorithm is defined as follows [23,24]:

**ALGORITHM**: ⟨algorithm name⟩
**SPECIFICATION**: ⟨algorithm specification⟩
**BEGIN**: ⟨initialization of variables and function in the recurrence⟩
**TERMINATION**: ⟨termination condition of the recurrence⟩
**A_I**: ⟨algorithm invariant⟩
**RECUR**: ⟨set of recurrences⟩
**END**

We get a new representation of algorithm, mainly a set of recurrences and initiations. That is exactly a set of mathematical formulae and is easy for formal proof and derivation. It characterizes main idea of an efficient algorithm and is more precise and simple than the representation of algorithm in natural language, flowchart and program. **The Radl expressions have referential transparency and make the formal derivation of algorithms possible. The particular merit of the new representation of algorithm is easiness of understanding and demonstrating the ingenuity and correctness of an algorithm.**

### 2.3 The New Techniques About Loop Invariants

The recursive program corresponding to recursive algorithm can be developed directly based on the recurrence relation. We just pay main attention on developing the program corresponding to iterative algorithm. The key for developing correct iterative program is loop invariant. This is recognized by not only the advocator of formal methods of design algorithms and programs but also some specialist of algorithm design, for example, Kingston, Baase, etc. However, the existing standard strategies for developing loop invariants are only suitable for some simple problems. There are many complicated algorithms and programs that cannot get satisfying loop invariants using these techniques. This leads to

that many computer scientists doubt the possibility of deriving or proving algorithms and programs using loop invariant. In [25], we exposed new properties of loop invariant and presented the new definition of loop invariant and two new strategies for developing it. Following is the new definition and one of the two new strategies.

The new definition and strategies are quite powerful, especially in using the recursive definition technique of new strategy to develop the loop invariant of an iteration program with inherent recursive property. Using the new techniques, we have formally proved and derived many nontrivial algorithmic programs, including graph algorithms, travel tree algorithms, sorting algorithms, array section algorithms and some numeric algorithms. A convincing example is formal development of Knuth's challenging program that converts a binary fraction to decimal fraction with certain conditions [13,27,28].

### 2.4 Genericity for Modeling

The generic mechanisms in executable programming languages such as Java and C++ play an important role in increasing the reusability and reliability of software and efficiency of software development. However, how to apply genericity to modeling language is rarely successful, although generic mechanism such as classifier templates, operation templates and package templates is provided in Unified Modeling language. Due to the complexity of UML itself and the difficulty of use, the wide application of genericity in MDE is seriously affected. To solve the above problems, the author of this paper proposes a generic modeling language mechanism implexmented in PAR [34]. One of the distinct features of the PAR platform is the agile genericity mechanisms. In PAR not only a value, a data type and an ADT can be generic parameter, and a computing-action (including operator, method, function and procedure, transaction, subsystem and web service, etc.) can be generic parameter too.

### 2.5 The New Techniques for Generating Database Application Program

In Radl and Apla, accessing to database is an expression of abstract operations rather than SQL statements. The expression is much simpler and shorter than SQL statement. It gives us one methodology to develop the database application system with high reliability and productivity. This makes formal derivation or proof of a database application program possible.

### 2.6 Distributed Transaction Processing in PAR

The popularization and application of advanced technologies such as cloud computing, big data and information system make computer software more and more complex, and the reliability and development efficiency of software cannot be guaranteed. The distributed transaction processing technology can be widely

used in software development. Here, an abstract language mechanism of concurrent distributed transaction processing is proposed and integrated into the Apla modeling language. The expanded Apla language of the platform is called Apla+. Based on Apla+, a general transaction processing system is implemented [23]. It is very easy and convenient to build abstract general transaction processing programs and then convert them into Java and other executable language programs. Because the strict correctness of abstract programs can be proved by the standard program proving method, the reliability of the transaction processing system is improved. This paper introduces the whole process of transaction processing program design and code generation through a typical example.

## 3 Main Elements in PAR

PAR method given in [25,26] is a formal methods. It provides the methodology that supports formal development of algorithmic programs described using some executable language; say Ada, Java, C++ and C#, from their formal specification. The Forming of PAR is a long term research plan and been supported by a series of national research foundations including NSFC and 863 High-Tech program.

### 3.1 Data Type and Action in PAR

#### 3.1.1 Standard Data Type

The standard data type referred to in this article is the data type set in the common executable programming language such as C++ and Java, character type, floating-point type, Boolean type and so on.

#### 3.1.2 Predefined Composed Data Type

These type types are stored in the component library in the form of abstract data type (ADT) designed by the system designer, and can be used as standard data types. The system predefines set, List, bag, binary tree, graph, relational data and other predefined data types. The two implementations of each predefined data were predefined.x

#### 3.1.3 Self-defined Abstract Data Type (ADT)

The PAR platform provides an abstract data type language mechanism defined self for building system models. The above three data types can be used as generic parameters of modeling language.

#### 3.1.4 Standard Action

Standard actions are operators in common executable programming language.

- Numeric operators: $+, -, *, /, \%$
- Logic operators: $\wedge, \vee, \equiv, \neg, \Box$
- Comparison Operator$>, \geq, <, \leq, =$

#### 3.1.5 Self-defined Action

User defined action self: Subprogram, function, procedure, method, subsystem, transaction, service, thread, etc.

The above two actions can be used as generic parameters of modeling language.

#### 3.1.6 Pre-defined Action

In PAR, the predefined action including several quantifiers. See Sect. 3.3.

### 3.2 Formal Modeling Language Radl

#### 3.2.1 Requirement Modeling

Using the data types and actions provided in the Radl modeling language, we can describe the formal requirements of various typical algorithms and applications. Based on these formalized requirements, we can make use of the change rules of quantifier and other logical expressions to realize the system function refinement and data refinement, and get a system model which is closer to the executable program language model. That is the algorithm model and the abstract program design model.

#### 3.2.2 Algorithm Modeling

Radl was designed for the description of algorithm specifications, transformation rules for deriving algorithms and algorithms itself. We presented a set of abstract notations for expressing pre-defined abstract data type, say array, set, sequence, binary tree and graph, etc. The motivation of developing these mathematics-oriented notations is aimed at making specification transformation, algorithm derivation and program proof like operating traditional mathematical formula. The most important notion in PAR method is *recurrence relation of problem solving sequence*, short for *recurrence*. Based on the recurrence, the core of an algorithm is defined as a set of recurrences, see Sect. 2.2.

The Radl expressions have referential transparency that makes the formal derivation of algorithms possible. Radl was designed for the description of algorithm specifications, transformation rules for deriving algorithms and algorithms itself. We presented a set of abstract notations for expressing pre-defined data type, say array, set, list, tree, graph and database, etc. Radl provides a user-defined mechanism for abstract data type.

### 3.3 Rules of Specification Transformation

Most of specification transformation rules are quantifier properties and are proved in [15,16]. Following are used in this paper. Let $\theta$ be an binary operator and big $\theta$ be the quantifier of operator $\theta$, then,

$$(\theta\ i : r(i) : f(i)) \tag{1}$$

Means the quantity of $f(i)$ where $i$ range over $r(i)$. We write the quantifier of binary operator $+$, $\bullet$, $\wedge$, $\vee$, $\diamondsuit$ (minimum), $\blacklozenge$ (maximum), $\cap$ (intersection),

$\cup$ (union) and $\uparrow$ as $\sum, \prod, \forall, \exists, \ \Diamond, \blacklozenge, \cap, \cup$ and $\uparrow$. Obviously operator $+, \bullet, \wedge, \vee, \Diamond, \blacklozenge, \cap, \cup$ are associative and commutative and their quantifier $\theta$ have following properties:

(a) Multi-dummies

$$(\theta\ i, j : r(i) \wedge s(i, j) : f(i, j)) = (\theta\ i : r(i) : (\theta\ j : s(i, j) : f(i, j))) \tag{2}$$

(b) Split with no overlap

$$(\theta\ i : r(i) : f(i)) = (\theta\ i : r(i) \wedge b(i) : f(i))\ \theta\ (\theta\ i : r(i) \wedge \neg b(i) : f(i)) \tag{3}$$

(c) One point split

$$(\theta\ i : 0 \leq i < n + 1 : f(i)) = (\theta\ i : 0 \leq i < n : f(i))\ \theta\ f(n) \tag{4}$$

(d) Generalized Associativity and Commutativity

$$(\theta\ i : r(i) : s(i) \theta\ f(i)) = (\theta\ i : r(i) : s(i))\ \theta\ (\theta\ i : r(i) : f(i)) \tag{5}$$

### 3.4 Modeling Language Apla

The purpose of developing Apla is to implement functional abstract and data abstract in program development perfectly, so that any Apla program is simple enough and is ease for understanding, formal derivation or proof. It is also ease to transform into some OOP language programs, say C++, C#, and Java, etc.

Apla is a object-based programming with convenient generics. The purpose of developing Apla is to implement functional abstract and data abstract in program development perfectly so that any Apla program is simple enough and is ease for understanding, formal derivation or proof. It is also ease to transform into some OOP language programs, say C++, Java, C# etc... Apla and Radl have same standard procedures and functions. The data types and Pre-defined ADTs and are also same. We borrow some control structure from Dijkstras Guarded Command Language, but restrict the nondeterminism.

### 3.5 Generic Constructions [34]

#### 3.5.1 Type Region and Type Variable

In Radl and Apla, we define the set of types that satisfy some properties as type region and a type parameter in a program unit as type variable. The syntax of the type region declaration is as follows:

$$some\ type\ =\ \{set\ of\ types\ satisfied\ some\ properties\};$$

#### 3.5.2 Action Region and Action Variable

We define the set of all action that satisfy some properties as action region and an action parameter in a program unit as action variable. The syntax of the action

region declaration is similar with type region. The action can be the predefined operators of Apla and defined procedures, functions and services by users.

$$Action\ region:\ someaction\ =\ \{set\ of\ actions\ satisfied\ some\ properties\};$$

### 3.5.3 ADT Region and ADT Variable

The ADT consists of the set of data and the set of operations. The model of ADT can be a algebra system. We define the set of all ADT that satisfy some properties as ADT region and an ADT parameter in a program unit as ADT variable. The syntax of the action region declaration is similar with type region.

$$ADT\ region:\ someadt\ =\ \{set\ of\ ADT\ satisfied\ some\ properties\};$$

## 3.6 Mechanism of Distributed Transaction Processing

Following is two components described using Java, where running of TransactionThread will make PAR enter concurrent and distributer environment, DistTransaction is used to implement distributed transaction process. Following operation appeared in Radl and Apla that make PAR implement simply Distributed Transaction Processing.

**TransactionThread**: component of transaction processing thread;
**DistTransaction**: distributed transaction processing component;
*commit*
*rollback*
*exception*
**TransactionThread**: component of transaction processing thread;

## 3.7 Automatic Model Transformation Tools

Radl algorithms and Apla programs are simple enough and ease for formal derivation and proof. But, it cannot be executed in a computer. Therefore, we developed the PAR platform that consists of 5 automatic transformation tools of algorithms or programs. One of them would be able to transform a Radl algorithm into Apla program. Others may transform Apla programs to the programs of target language, says C++ and Java, etc. Based on the PAR platform, the efficiency of developing algorithms and program and reliability of the programs are increased obviously.

## 3.8 Correctness of Model and Correctness of Model Transformation

### 3.8.1 Correctness of Model

The model defined in this paper can be refined by manual refinement of the requirement model, and then the algorithm model, abstract program model and executable language program model can be obtained in turn. These models can also be automatically generated by model automatic converters. Such models can be based on the definition of loop invariant and development strategy given

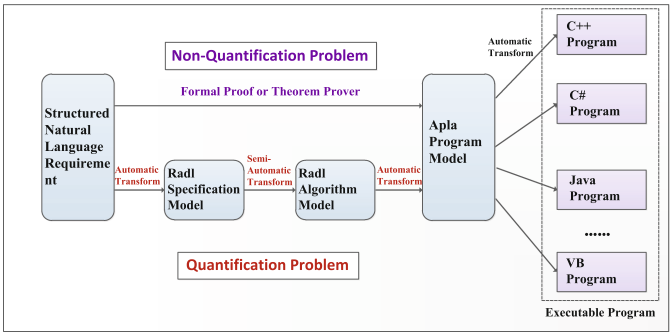


**Fig. 1.** Architecture of PAR platform

by Xue [25], using Dijkstra-Gries standard algorithm and program correctness proof method, formalized or automatically proved the relevant algorithms and program correctly. It can also be calculated the reliability of the program by software testing method.

#### 3.8.2 Correctness of Model Transformation

The correctness of model transformation directly affects the correctness of the model. The correctness of model transformation can guarantee the correctness of the algorithm and program generated by the transformation. To prove the correctness of the model transformation, it is necessary to use the knowledge of category theory.

### 3.9 Architecture of PAR Platform

The architecture of PAR Platform is show in Fig. 1. There are two ways to generate codes. The first way is for processing quantification problem. PAR Platform can transform SNL requirement model to Radl specification model, then to Radl algorithm model, and to Apla abstract program model, finally to executable program. The second way is for processing non-quantification problem. Users can directly design Apla program manually and give it's formal proof, then transform it to executable program.

## 4 Applications and Case Studies

### 4.1 There are Two Kinds of Applications of PAR

The first one is that many nontrivial algorithms and programs have been developed formally, formal derivation or formal proof, including graph algorithms [29], travel tree algorithms [32], array section algorithms [25,26], Knuths famous hard problem of cyclic permutation [10,24,33]. The abstract Hopcroft-Tarjan planarity algorithm was described in Apla. The Apla program was transformed by PAR platform to C++ code that can correctly test the planarity and generate

improved planar embedding [10]. A more convincing example is formal development of Knuths challenging program that converts a binary fraction to decimal fraction with certain condition [13,27,28].

The second one is to develop several safety-critical information systems, including shuttle transportation problems [36], discrete optimization algorithm design [37], fire evacuation [38], population classification in earthquakes [39], Emergency railway transportation planning [40], active services support for disaster rescue [41], Airline passenger profiling [42] and Industrial Accident Early Warning [43], etc. A general distributed transaction processing system is implemented in PAR and used in student information management system [23].

### 4.2 Case Studies

The case studies consists of two typical nontrivial examples. One is a simple problem, but the solution in not. Another is one Apla program section that comes from a student information management system.

**The Cubes of the First n Natural Numbers.** Given is an integer $n$, $0 \leq n$. Develop a program to store in array $b[0 \ldots n-1]$ the cubes of the first $n$ natural numbers. Thus, upon termination, for each $i, 0 \leq i < n, a[i] = i^3$. The program may not use exponentiation, multiplication, or division, only use addition and subtraction.

The problem seems quite simple but in my teaching experience no student is be able to solve the problem correctly. Even more, no student is able to understand the correctness of the program after we show them the final solution. Here, we develop the algorithm and the program based on PAR method.

■ **Step 1.** Describe the formal functional specification of an algorithmic problem using Radl;

**AQ1:** Given is an integer $n, 0 \leq n$

**AR1:** $(i : 0 \leq i < n : a(i) = i^3)$

For satisfying the postcondition AR2, we repeat following two steps:

■ **Step 2.** Partition the problem into a couple of subproblems each of which has the same structure with the original problem;

■ **Step 3.** Formally derive the algorithm from the formal functional specification. The algorithm is described using Radl and represented by recurrences and initialization;

Based on the postcondition AR2 and algebra law, we have

$a(i+1) = (i+1)^3 = i^3 + 3i^2 + 3i + 1$ [Partition the problem]

Let $a(i) = i^3$

Then $a(i+1) = a(i) + 3i^2 + 3i + 1$ [Derive the recurrence]

For computing $3i^2$

Let $b(i) = 3i^2$

Then $b(i+1) = 3(i+1)^2 = 3i^2 + 6i + 3$ [Partition the problem]

$= b(i) + 6i + 3$ [Derive the recurrence ]

For computing $6i$

Let $c(i) = 6i$

Then $c(i+1) = 6(i+1) = 6i + 6 = c(i) + 6$.....(1)
$b(i+1) = b(i) + c(i) + 3$.....(2)
For computing $3i$
Let $d(i) = 3i$
Then $d(i+1) = 3i + 3 = d(i) + 3$.....(3)
$a(i+1) = a(i) + b(i) + d(i) + 1$.....(4)

Obviously, the algorithm invariant **AI1** is

**AI1**: $a(i) = i^3 \wedge b(i) = 3i^2 \wedge d(i) = 3i \wedge c(i) = 6i \wedge 0 \leq i < n$

Combining recurrence (1)–(4), then assign the initial value to each function, we got the algorithm for solving the problem:

**ALGORITHM**: Cube
**SPECIFICATION**: $\{AQ1; AR1\}$
$\{AI1\}$
**RANGE**: $0 \leq i < n$
**BEGIN**: $i := 0 : a(i) = 0 : b(i) = 0 : c(i) = 0 : d(i) = 0;$
**RECUR**: $a(i+1) = a(i) + b(i) + d(i) + 1;$
$b(i+1) = b(i) + c(i) + 3$
$c(i+1) = c(i) + 6;$
$d(i+1) = d(i) + 3;$
**END**

■ **Step 4.** Develop loop invariant directly based on our new strategy [23] straightforward; Considering the algorithm, we need an array variable, say a, to store the cube of integer i ; simple b, c, d to store the value of function b(i), c(i), d(i). According to our definition [23], the loop invariant of a loop program is an assertion that reflects verification law of each variable in the loop. Thus, we have the loop invariant:

**LI.1**: $a(i) = i^3 \wedge b(i) = 3i^2 \wedge d(i) = 3i \wedge c(i) = 6i \wedge 0 \leq i < n$

■ **Step 5.** Based on the loop invariant, transform the Radl algorithm and algorithm specification to Apla program and program specification mechanically or automatically;

In this example, the program specification $\{PQ1; PR1\}$ is same as the algorithm specification $\{AQ1; AR1\}$. We transform the algorithm and algorithm invariant into Apla program as follows:

**PROGRAM**: Cube
$\{PQ1; PR1\}$
$i := 0 : a(i) = 0 : b(i) = 0 : c(i) = 0 : d(i) = 0;$
$\{LI.1\}$
**DO** $i \neq n - 1 \rightarrow a(i+1) = a(i) + b + d + 1;$
$b = b + c + 3$
$c = c + 6;$
$d = d + 3;$
**OD**

This program was described using Extended Guarded Command Language (Apla), but it cannot be run on a computer. For getting an executable program, we must do:

■ **Step 6.** Transform the Apla program to an executable language program, say Ada, C++, Java, etc. mechanically or automatically.

**Apla Program Section for Information Processing.** The problem is to connect the student database and to get student name from the database, to store the names into an array, then to store and sort the data in the array to the binary tree according to inorder and dictionary order.

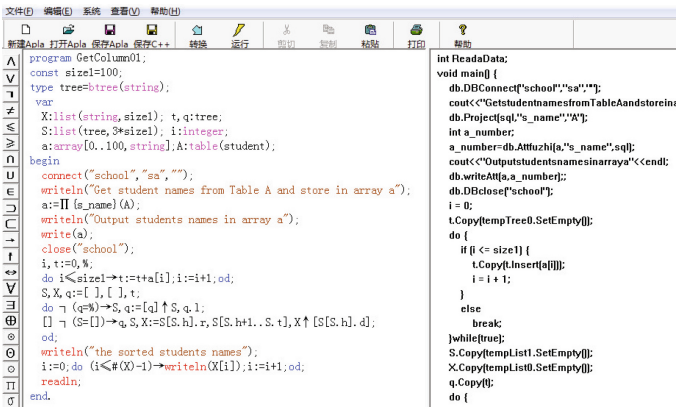


**Fig. 2.** Apla and C++ Program section for accessing and sorting student names

The Apla program of solving the problem is in the left part of the Fig. 2. The right part of the Fig. 2 is the corresponding C++ code generated by automatic model transforming tools. Observing the Apla program is quite short, only 32 lines, but C++ code has 330 lines. The Apla program access the student name from student database, then to build the binary tree stored by inorder, then to store the node data according to inorder tree walk. The student name in the array is ordered according a nondecreasing sequence.

This example show us 3 points: the abstract lever of Apla is very high; the Apla program is ease to read; the correctness of Apla program is quite ease to verifying. The details showed in [32].

## 5 Related Work

The PAR method and PAR platform is a unified and systematic approach for formal development of algorithm and programs. There are some similar formal approaches with ours.

**Program calculus**, this approach is the most famous formal methods for developing algorithm and program, which was created by Dijkstra [7] and developed into practical techniques by Gries [9], Backhouse et al. [3]. The approach treats program and algorithm as same thing. A program is developed hand-in-hand with its loop invariant and proof of correct-ness. Our development of the algorithm and program has some similarity with theirs. However our emphasis is different: we separate algorithm, represented by recurrence, from program, then pay special attention on formal derivation of algorithm rather than program calculus. The algorithm represented by recurrence relation is exactly a set of mathematical formula and has mathematical transparency. Therefore, it can be produced directly by formal derivation based on some simple mathematical tools. After getting correct algorithm represented by recurrence, trans-forming it into correct program is a trivial work. The work can be done mechanically. It is

possible to develop a software system to finish it automatically. However, using calculus approach, one derive program by weakening program specification. The program statement cannot be produced directly by formal derivation because it has no mathematical transparency. There is a big gap between weakened program specification and program. It is difficult to remove the gap mechanically and automatically.

**B method** [1], **RAISE** [18], **Kestrelwares** [19], **Orc** [6,12,16], **rCOS**, [15,21], **NDAUTO** [44] and **NDADAS** [44], etc., Comparing with those formal methods and program generating system, PAR method and PAR platform have following innovative techniques that described in Sect. 2: 2.1 A Unified Approach for Designing Algorithm Based on Quantifier Transformation; 2.2 A New Representation of Algorithms; 2.3 The New Techniques About Loop Invariants; 2.4 Generosity for Modeling; 2.5 The new techniques for generating database application program; 2.6 Distributed Transaction Processing In PAR.

## 6 Conclusion and Future Work

1. On the basis of the study of the essential features of formal methods, this paper focuses on overcoming complex, difficult to understand and inconvenient for application in the existing formal methods specification and design language. We put forward a new formal methods definition. Based on the definition, we research successful and practical formal methods and support platform, namely the PAR method and PAR Platform. PAR method and the PAR platform have been successfully applied in the formal development of complex algorithms and information processing software. The effect is remarkable, which proves the correctness and effectiveness of the new definition of formal method proposed in this paper.
2. The paper describes the effectiveness of the 6 innovative theories and techniques given in Sect. 2.2. These innovative theories and techniques, which guarantee the advanced and innovative nature of the PAR and PAR platforms, and become the theoretical and technical basis for the realization of the PAR platform, and can also effectively carry out the development and promotion of the development of formal software.
3. The formal methods proposed and implemented by PAR have been applied in the formalized development of complex algorithms and programs. It has solved a number of international problems and has important theoretical value. Using the PAR method and the PAR platform, many Safety-Critical information processing systems have been developed, in the actual earthquake, fire, and aviation accidents. The success of disaster rescue work proves that PAR has great practical application value.
4. The formal methods of software development and the model driven engineering MDE have their own advantages and disadvantages in practical application. Combining these two methods can serve as a complement to each other and will effectively promote the renewal and development of software development methods. We will work in this area.

**Acknowledgments.** At first, our thanks go to anonymous reviewers for helpful suggestion and comments that changed the presentation of this paper.

Our warm thanks go also to Prof. David Gries for his kind encouragement and suggestions. His academic thought deeply influenced our research team.

We would like to thank Xu Wensheng, Zuo Zhengkang, Shi Haihe, Wang ChangJing, who attended the previous work of the project.

## References

1. Abrial, J.-R.: The B Book - Assigning Programs to Meanings. Cambridge University Press, Cambridge (1996)
2. Adesina, O.: Integrating formal methods with model-driven engineering. In: International Conference on Model-Driven Engineering Languages and Systems, Ottawa, Canada (2015)
3. Backhouse, R.C.: Program Construction and Verification. Prentice Hall International, London (1986)
4. Bjørner, D., Havelund, K.: 40 years of formal methods: some obstacles and some possibilities? In: Jones, C., Pihlajasaari, P., Sun, J. (eds.) FM 2014. LNCS, vol. 8442, pp. 42–61. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06410-9_4
5. Clarke, E.M., Wing, J.M.: Formal methods: state of the art and future directions. ACM Comput. Surv. **28**(4), 626–643 (1996)
6. Cook, W., Misra, J.: Structured interacting computations. In: Wirsing, M., Banâtre, J.-P., Hölzl, M., Rauschmayer, A. (eds.) Software-Intensive Systems and New Computing Paradigms. LNCS, vol. 5380, pp. 139–145. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89437-7_9
7. Dijkstra, E.W.: A Discipline of Programming. Prentice Hall, Upper Saddle River (1976)
8. Gargantini, G., Riccobene, A.E., Scandurra, P.: Combining formal methods and MDE techniques for model-driven system design and analysis. Int. J. Adv. Softw. **3**(1 and 2), 1–18 (2010)
9. Gries, D.: The Science of Programming. Springer, New York (1981). https://doi.org/10.1007/978-1-4612-5983-1
10. Gries, D., Xue, J.: Generating a random permutation. In: BIT28, vol. 10 pp. 569–572 (1988)
11. Gries, D., Xue, J.: The hopcroft-tarjan plannarity algorithm presentations and improvements. TR88-906, CS Department of Cornell University, pp. 1–20 (1988)
12. Kitchin, D., Quark, A., Cook, W., Misra, J.: The Orc programming language. In: Lee, D., Lopes, A., Poetzsch-Heffter, A. (eds.) FMOODS/FORTE -2009. LNCS, vol. 5522, pp. 1–25. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02138-1_1
13. Knuth, D.: A simple program whose proof isn't. In: Beauty is Our Business. A Birthday Salute to E.W. Dijkstra (1990). (Ed. by, W.H.J. Feijen et al.)
14. Jones, C.B.: Systematic Software Development Using VDM. Prentice-Hall International, New York (1986)
15. Liu, Z., Morisset, C., Stolz, V.: rCOS: Theory and tool for component-based model driven development. In: Arbab, F., Sirjani, M. (eds.) FSEN 2009. LNCS, vol. 5961, pp. 62–80. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11623-0_3

16. Misra, J., Cook, W.R.: Computation orchestration: a basis for wide-area computing. J. Softw. Syst. Model. (2017)
17. Paull, M.C.: Algorithm Design—A Recursion Transformation Framework. Wiley, New York (1987)
18. RAISE. http://spd-web.terma.com/Projects/raise
19. Smith, D.R.: KIDS: a semiautomatic program development system. IEEE Trans. Softw. Eng. **16**(9), 1024–1043 (1990)
20. Spivey, J.M.: Introducing Z: A Specification Language and Its Formal Semantics. Combridge University Press, New York (1988)
21. Ke, W., Li, X., Liu, Z., Stolz, V.: rCOS: A formal model-driven engineering method for component-based software. Fronti. Comput. Sci. China **6**(1), 17–39 (2012)
22. Woodcock, J., Larsen, P.G., Bicarregui, J., Fitzgerald, J.: Formal methods: practice and experience. ACM Comput. Surv. **41**(4), 19 (2009)
23. Xia, J., Xue, J.: Design and implementation of concurrent distributed transaction in modeling language Apla. In: NCTCS (2018, to appear)
24. Xue, J., Gries, D.: Developing a linear algorithm for cubing a cycle permutation. Sci. Comput. Program. **11**, 161–165 (1988)
25. Xue, J.: Two new strategies for developing loop invariants and its applications. J. Comput. Sci. Technol **8**(3), 147–154 (1993)
26. Xue, J.: A unified approach for developing efficient algorithmic programs. J. Comput. Sci. Technol. **12**(4), 314–329 (1997)
27. Xue, J., Davis, R.: A simple program whose derivation and proof is also. In: First IEEE International Conference On Formal Engineering Method (1997)
28. Xue, J., Davis, R.: A derivation and proof of Knuths binary to decimal program. In: Software: Concepts and Tools, vol. 12, pp. 149–156 (1997)
29. Xue, J.: Formal derivation of graph algorithmic programs using partition and recur. J. Comput. Sci. Technol. **13**(6), 553–561 (1998)
30. Xue, J.: A practicable approach for formal development of algorithmic programs. In: International Symposium on Future Software Technology (ISFST-1999), Masami Noro, October 1999
31. Xue, J.Y.: Developing the generic path algorithmic program and its instantiations using PAR method. In: The Second Asia Workshop On Programming Languages and Systems, Korea (2001)
32. Xue, J.: PAR method and its supporting platform. In: International Workshop on Formal Method for Developing Software, Annual Report, Macao: UNU-IIST, no. 348 (2006)
33. Xue, J., Yang, B., Zuo, Z.: A linear in-situ algorithm for the power of cyclic permutation. In: Preparata, F.P., Wu, X., Yin, J. (eds.) FAW 2008. LNCS, vol. 5059, pp. 113–123. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69311-6_14
34. Xue, J.: Genericity in PAR platform. In: Liu, S., Duan, Z. (eds.) SOFL+MSVL 2015. LNCS, vol. 9559, pp. 3–14. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31220-0_1
35. Xue, J.: PAR: a model driven engineering platform for generating algorithms and software. In: Symposium on Programming: Logics, Models, Algorithms and Concurrency to recognize Jayadev Misra's Accomplishments, University of Texas, 29th and 30th April 2016. https://www.cs.utexas.edu/symposium
36. Zheng, Y.J., Xue, J.Y.: A simple Greedy algorithm for a class of shuttle transportation problems. Opt. Lett. **3**(4), 491–497 (2009)
37. Zheng, Y.J., Xue, J.Y.: A problem reduction based approach to discrete optimization algorithm design. Computing **88**(1–2), 31–54 (2010)

38. Zheng, Y.J., Ling, H.F., Xue, J.Y., Chen, S.Y.: Population classification in fire evacuation: a multiobjective particle swarm optimization approach. IEEE Trans. Evol. Comput. **18**(1), 70–81 (2014)
39. Zheng, Y.J., Ling, H.F., Chen, S.Y., Xue, J.Y.: A hybrid neuro-fuzzy network based on differential biogeography-based optimization for online population classification in earthquakes. IEEE Trans. Fuzzy Syst. **23**(4), 1070–1083 (2014)
40. Zheng, Y.J., Zhang, M.X., Ling, H.F., Chen, S.Y.: Emergency railway transportation planning using a hyperheuristic approach. IEEE Trans. Intell. Transp. Syst. **16**(1), 321–329 (2015)
41. Zheng, Y.J., Chen, Q.Z., Ling, H.F., Xue, J.Y.: Rescue wings: mobile computing and active services support for disaster rescue. IEEE Trans. Serv. Compt. **9**(4), 594–607 (2016)
42. Zheng, Y.J., Sheng, W.G., Sun, X.-M., Chen, S.Y.: Airline passenger profiling based on fuzzy deep machine learning. IEEE Trans. Neural Netw. Learn. Syst. **28**(12), 2911–2923 (2017)
43. Zheng, Y.J., Chen, S.-Y., Yu, X., Xue, J.-Y.: A pythagorean-type fuzzy deep denoising auto-encoder for industrial accident early warning. IEEE Trans. Fuzzy Syst. **25**(6), 1561–1575 (2017)
44. Xu, J.: The Automation of software. IEEE Trans. Syst. (1993). Qinghua Published Company
45. Formal methods C Wikipedia. https://en.wikipedia.org/wiki/Formal_methods