Component-Based Verification Model of Sequential Programs

Pei He[1, 2, 3*], Achun Hu[1], Dongqing Xie[1], Zhiping Fan[1]

[1] School of Computer Science and Educational Software, Guangzhou University, Guangzhou 510006, China
[2] Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China.
[3] School of Computer and Communication Engineering, Changsha University of Science and Technology, Changsha 410114, China.

* Corresponding author. Email: bk_he@126.com
Manuscript submitted January 10, 2014; accepted March 8, 2015.
doi: 10.17706/jsw.10.11.1319-1326

Abstract: Hoare's logic helps with program state descriptions, but is difficult to manipulate. Model checking emerged as a new trend in program verifications is best applied to system designs rather than implementations. This paper is committed to establish a component-based verification framework that combines both of them. The method applied consists of two steps: regarding predicates as states and connecting them with functional components in light of their relationships. Once a framework is set up, both program generation and verification can be automatically carried out. The principle presented here is not only applicable to sequential programs, but also to other types of program structures and paradigm such as iteration, branch structure and grammatical evolution, etc.

Key words: Finite state transition system, grammatical evolution, sequential programs, verification framework.

1. Introduction

Two major approaches to program reliability are Hoare's logic [1]-[6] and model checking [5], [7]-[10]. Hoare's logic is a classical semantic framework easily applied in description of changes of program states, but difficult to use. Model checking techniques emerged as new trend in verifications of programs are best applied to system designs rather than implementations [8]. Whether it is practicable to integrate these systems in a tightly coupled manner and technically how to achieve this end arouse great interest in computer community.

This paper is a sister work of [11]-[13] that initiated studies on combination of Hoare's calculus [1], [3]-[6], model checking [7], [9], automaton [14]-[15] and genetic programming [16]-[18]. In reference paper [11], we have paid great attention to reusable technologies widely used in mathematics and software developments, proposing a formal framework to verify and evolutionally construct computer programs. The measure employed is to construct transition systems over program components in terms of their semantic relations. Similarly, after introducing these techniques into genetic programming (GP) [18] as well as its important variant like grammatical evolution [16], [17], we obtained some formal GP frameworks [11]-[13]. These works make it possible to verify and generate programs in path checking and path searching technologies. However, to improve the performance of the systems, we should take notice of their structures. This forms the basis of the present work.

In the present paper, we will establish a component-based verification framework combining both of them. Compared with the work of [11], this system has such advantages as ease for representation, and convenience with parallel processing. Apart from being useful in program verifications, the presented model can also be applied to search for correct programs. The paper is organized as follows. Section 2 gives a brief introduction to Hoare's logic; sections 3 through 4 introduce the present approach and examples; finally discussions and the conclusion are summarized in sections 5 and 6.

2. Hoare's Calculus

Hoare's logic is a typical framework originally proposed by Hoare in 1969 [1], [5] for partial correctness of programs. In this system, a Hoare's formula is of the form {P}S{Q}, where both P and Q are logic formulae, called pre-/post- condition and S represents a program segment. {P}S {Q} means Q will hold after executing S on the premise of P. Hoare's logic includes one axiom and 4 inference rules as given in Fig. 1 [1], [11]. Based on the only axiom for assignment statement, and several inferences, we can verify programs through calculating on elementary program structures and the whole program. Fig. 2 shows the usage of proof rules in program verifications.

Ax1: Assignment

{P[t/x]} x:=t {P}

Rule 2: Composition rule

$$\frac{\{P\}S_1\{R\},\{R\}S_2\{Q\}}{\{P\}S_1;S_2\{Q\}}$$

Rule 3: If-then-else rule

$$\frac{\{P\wedge e\}S_1\{Q\},\{P\wedge\neg e\}S_2\{Q\}}{\{P\}\text{If e then } S_1 \text{ else } S_2\{Q\}}$$

Rule 4: While rule

$$\frac{\{P\wedge e\}S\{Q\}}{\{P\}\mathit{While}\,\text{e do } S\{P\wedge\neg e\}}$$

Rule 5: rewriting rule

$$\frac{P\to P_1,\{P_1\}S\{Q_1\},Q_1\to Q}{\{P\}S\{Q\}}$$

Fig. 1. Rules of Hoare's logic.

{x=5^y=2}
{x+1=6^y=2}

X:= x+1;

{X=6^y=2}
{x=6 ^ y-2=0}

y:= y-2;

{x=6 ^ y=0}

Fig. 2. Verification process.

3. Component-based Framework

By component based verification framework here, we mean this system just prove those programs constructed from a given set of trusted components such as functions, Booleans, iterative bodies, etc. If one doesn't trust such set of components, he can verify them downwards in the same way.

In this section, we are dedicated to the establishment of a component based verification framework, which combines both Hoare's logic and finite state automaton. Based on our consideration on Hoare system's modeling behaviour, we find a model equivalent to it.

Definition 1 (Validation Composition) Given two Hoare triples $\{P\}f\{Q\}$, $\{R\}g\{W\}$, fg is a valid composition if $Q\to R$.

Of course, it follows easy the composition of *f* and *g* defined here is valid under Hoare's logic algebra.

Definition 2 (Sequent) Let F be a set of component functions, HF the set of Hoare's triples for F. A string

$\alpha \in F^*$ is a sequent, if the composition of any two neighbour functions (if any) in the string is valid. Particularly, we think of ε as the identity element under concatenation Thus when in need, we often omit them.

Definition 3 (L-composite model) Given *F*, *HF* as above, a finite state transition graph is called the *L*-composite model on both *F* and *HF*, denoted by $M(F, HF)$, if it satisfies: $L(M) = \{\alpha \in F^* \mid \alpha \text{ is a sequent}\}$. Here *L*(*M*) is the set of strings concatenated from edge labels along all possible paths in $M(F, HF)$.

Obviously, composite model defines all possible legal computation on F. Note that each computation satisfies associativity.

Definition 4 ($\alpha - \varepsilon$ string) A $\alpha - \varepsilon$ string is what obtained from insertions of the symbol ε in the original string $\alpha \in F^*$.

Theorem 1 (Existence) Let $F = \{f_i \mid 1 \le \mathrm{i} \le \mathrm{n}\}$ be a set of component functions, *HF* a set of Hoare triples $\{\{P_i\} f_i \{Q_i\} \mid 1 \le \mathrm{i} \le \mathrm{n}\}$, then there exists a composite model $M(F, HF)$.

Proof. Firstly, let us construct the composite model.

By *HF*, we construct a predicate relation matrix as table 1. Draw a finite state transition diagram with either *P* or *Q* as nodes below.

Draw nodes for each predicate of $\{P_i \mid 1 \le \mathrm{i} \le \mathrm{n}\} \cup \{Q_i \mid 1 \le \mathrm{i} \le \mathrm{n}\}$

1) Draw an arrow from *P* to *Q* for each $\{P\} f \{Q\} \in HF$, and labeling it with a *f*.
2) Draw an arrow from R_i to R_j for each pair of (R_i, R_j) with $R_i \to R_j = T$ in table 1 , and labeling it with an ε . Here R_x is either P or Q.

This is the desired model $M(F, HF)$.

Secondly, let us prove: $L(M) = \{\alpha \in F^* \mid \alpha \text{ is a sequent}\}$.

=>: Assuming $\beta \in L(M)$ be a string concatenated from edge labels along a legal path $v_1 v_2 \cdots v_n$ in $M(F, HF)$, where v stands for either some P or Q. By the above mentioned graphic drawing rule, we follow each edge in the path corresponds to either a Hoare's triple, when in HF, or an ε arrow. Thus the composition of each pair of neighbour functions f_i and f_j is valid under definition 1. In fact, when f_i shares a common node with f_j, it's a trivial case; and when there are insertions of ε arrows between them, say $\cdots v_p \xrightarrow{f} v_i \xrightarrow{\varepsilon^*} v_q \xrightarrow{\beta} \cdots$, we have $\mid - (v_i \to v_q)$. Henceforth, β is a Sequent.

<=: Let $\beta \in L(M)$ be a Sequent $f_{i1} f_{i2} \ldots f_{im}$, according to picturing rule, we can easily prove the result by induction on the length of the arbitrary string β .

This completes the proof.

Clearly, the theorem for model existence presents a NFA like transition graph [14]-[15].

Theorem 2 Given F , *HF* as above, let $M(F, HF)$ be a composite model on them, and $\alpha = f_{i1} f_{i2} \ldots f_{im} \in F^*$, then $\{P\}\alpha\{Q\} \Leftrightarrow$ there exists a path in $M(F, HF)$ satisfying:

1) The concatenation $l_1 l_2 .. l_r$ of the edge labels along p is a $\alpha - \varepsilon$ string;
2) $\mid - (P \to P_{i1}) \wedge (Q_{im} \to Q)$. Here P_{i1}, Q_{im} stand for the pre- and the post-conditions of f_{i1} and f_{im} respectively.

Proof. =>: let $\{P\} f_{i1} f_{i2} \cdots f_{im}$ {Q} be a Hoare's triple, without losing the generality, also assuming the pre-/post-conditions of f_i s are distinct, then to get it by Hoare's calculus, $\mid - (P \to P_{i1}) \wedge (Q_{i1} \to P_{i2}) \wedge \cdots \wedge (Q_{im} \to Q)$ must hold, where $\{P_{ij}\} f_{ij} \{Q_{ij}\} \in HF$. According to the picturing rule above, we have a $\alpha-\varepsilon$ string of α from the labels on the path $P_{i1}\xrightarrow{f_{i1}}Q_{i1}\xrightarrow{\varepsilon}P_{i2}\xrightarrow{f_{i2}}\cdots\xrightarrow{\varepsilon}P_{im}\xrightarrow{f_{im}}Q_{im}$. Naturally, we get both (1) and (2).

<=: the proof for the necessity is easily. This completes the proof.

Based on the second theorem, we have an algorithm for the construction of a finite state automaton, and $\{P\}\alpha\{Q\}\Leftrightarrow$ there exists a path from P to Q such that the construction of labels along the path is a $\alpha-\varepsilon$ string. Particularly, the verification is simple if we further make the NFA deterministic.

Algorithm 1 (Verifying linear program) Given $F=\{f_i\,|1\le i\le n\}$, $HF=\{\{P_i\}f_i\{Q_i\}|1\le i\le n\}$ and the solving goal $\{P\}\alpha\{Q\}$, we can construct a NFA (see Fig. 3) as follows for verifying $\{P\}\alpha\{Q\}$.

1) Construct $M(F,HF)$ as proof of theorem 1;
2) Define the temporal initial states: $I=\{R\,|\,\{R\}\alpha\{W\}\in HF$ such that $P\to R\}$;
3) Define the temporal terminal states: $T=\{W\,|\,\{R\}\alpha\{W\}\in HF$ such that $W\to Q\}$;
4) Introduce the initial state P and connect it with all states in I by ε arrows
5) Introduce the finial state Q and connect all states in T with Q by ε arrows, which means $W\to Q$ for all Ws in T;
6) Find out the path from P to Q such that the string recognized is the $\alpha-\varepsilon$ string of α;

The problem verified is $P(X)\to Q(f)$, where f is the result from the composition operation on edge labels.

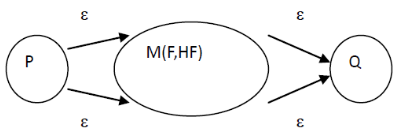

Fig. 3. Sketch of the model.

Notice that sequential model can also apply in verification of both branch and iterative structures. If regarding some f_i in the sequent $f_1 f_2\cdots f_n$ as a set of sequences $\{l_1,l_2,\cdots,l_m\}$ (l_i is a sequent), and furthermore make the convention $f_1 f_2\cdots f_{i-1}f_i f_{i+1}\cdots f_n=\{f_1 f_2\cdots f_{i-1}\}\times\{l_1,l_2,\cdots,l_m\}\ \times\{f_{i+1}\cdots f_n\}$, we will get the linear or sequential representation of branch structure. And iterative structure is a composition of linear and branching ones. Besides, this system can also work like that of [11] in evolutionally generating reliable programs.

The principle presented here is not only applicable to sequential programs, but also to other types of program structures and paradigm such as iteration, branch structure and grammatical evolution. For example, considering generation procedures of programs from the angle of grammatical derivations like constructing and interpreting sequences of productions, we can manage to obtain a grammatical model of grammatical evolution. Once a model is obtained this way, both syntactical and semantic calculations can be combined into a unified formal framework. We will discus them in another paper.

4. Examples

We will delineate the method by using the example of reference paper [11]. Given a set of Hoare triples and a predicate relation as shown in tables 1 and 2, verifying whether or not the program $f_4\ f_1\ f_3\ f_2\ f_4\ f_1$ constructed from the given components is correct with the pre-condition P1 and post-condition P4.

Solution: By theorem2, tables 1 through 2, we get the model of the given triples as shown in Fig. 4. Obviously, this model consists of three sub-modes: M1= ({P1,P4}, {f_1, f_3, f_2, f_4, ε}), M2= ({P2, P5}, {f_1, f_3, f_2, f_4, ε}), M3= ({P3, P6, P7},{f_1, f_3, f_2, f_4, ε}). Regarding them as transition diagrams, the possible languages they accept are shown in table 3. Since M1 recognizes the string $f_4\ f_1\ f_3\ f_2\ f_4\ f_1$, i.e. there exists a path in M1 to justify P1 f_4 P1 f_1 P4 f_3 P4 f_2 P1 f_4 P1 f_1 P4 as checked in Fig. 5, {P1} $f_4\ f_1\ f_3\ f_2\ f_4\ f_1$ {P4} must be a

Hoare triple. So $f_4\ \ f_1\ \ f_3\ \ f_2\ \ f_4\ \ f_1$ is correct with respect to the pre-condition P1 and post-condition P4

In addition, we can still prove $\{y+uz=xz \wedge u\geq 0 \wedge u>0\}\ f_1\ \ f_3\ \ f_2\ \ f_4\ \ \{y+uz=xz \wedge u\geq 0\}$ as in Fig, 6. According to Fig. 1, this Hoare triple means $\{\ y+uz=xz \wedge u\geq 0\ \}$ while $u>0$ do $\{f_1\ f_3\ \ f_2\ \ f_4\ \}$ $\{y+uz=xz \wedge \mathrm{u}=0\}$.

Solution:: by Fig.3 of theorem 2, to prove $\{y+uz=xz \wedge u\geq 0 \wedge u>0\}\ \ f_1\ \ f_3\ \ f_2\ \ f_4\ \ \{y+uz=xz \wedge u\geq 0\}$, for the sake of $(y+uz=xz \wedge u\geq 0 \wedge u>0)\rightarrow y+uz=xz$ and $(y+uz=xz \wedge u\geq 0 \wedge u>0)\rightarrow u>0$, we should prove both $\{y+uz=xz\}\ f_1\ \ f_3\ \ f_2\ \ f_4\ \{y+uz=xz\}$ and $\{u>0\}\ f_1\ \ f_3\ \ f_2\ \ f_4\ \{u\geq 0\}$. Having completed the proofs, combining them as shown in Fig. 6 will get the desired result. In this case, the proof concerns with two sub-models. They can work towards the desired problem in parallel.

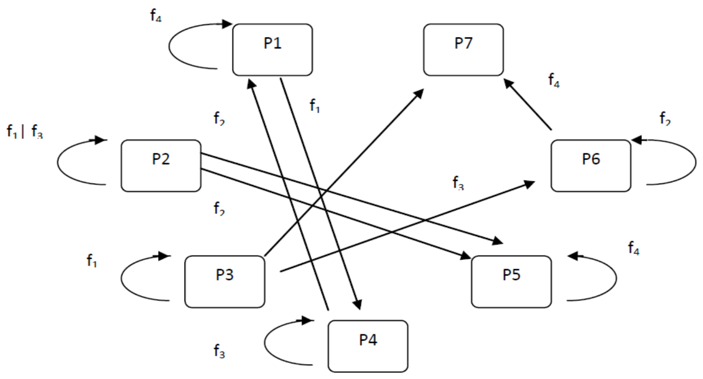

Fig. 4. Model of the given components. Unlabeled arrows are ones. Besides, arrows defined over states are omitted here.

Table 1. Relations between States (or Predicates)

	$\rightarrow$	State						
		P1	P2	P3	P4	P5	P6	P7
State	P1: $y+uz=xz$	T, f_4			f_1			
	P2: $u>0$		T, f_1, f_3			T, f_2		
	P3: $x=r+qz\wedge r\geq z\wedge z>0$			T, f_1			f_3	T
	P4: $y+(u-1)z=xz$	f_2			T, f_3			
	P5: $u\geq 0$					T, f_4		
	P6: $x=r+(q+1)z\wedge r\geq 0\wedge z>0$						T, f_2	f_4
	P7: $x=r+qz\wedge r\geq 0\wedge z>0$							T

Table 2. Transition Matrix Represented by Hoare Triples

	Transition function	Program component			
		f1	f2	f3	f4
State	P1: $y+uz=xz$	P4			P1
	P2: $u>0$	P2	P5	P2	
	P3: $x=r+qz\wedge r\geq z\wedge z>0$	P3		P6	
	P4: $y+(u-1)z=xz$		P1	P4	
	P5: $u\geq 0$				P5
	P6: $x=r+(q+1)z\wedge r\geq 0\wedge z>0$		P6		P7
	P7: $x=r+qz\wedge r\geq 0\wedge z>0$				

Table 3. The Languages of Sub-models

	Regular expression
M1	$(f_2\ \ f_4{}^*\ f_1\ f_3{}^*)^*\ \vert\ (f_4{}^*\ f_1\ \ \ f_3{}^*\ f_2)^*\ \vert\ (f_1\ f_3{}^*\ f_2\ \ f_4{}^*)^*\ \vert\ (f_3{}^*\ f_2\ \ f_4{}^*\ f_1)^*\ \vert\ f_1\ f_3{}^*\ \vert\ f_3{}^*\ f_2\ \vert\ f_2\ \ f_4{}^*\ \vert\ f_4{}^*\ f_1\ \vert\ f_4{}^*\vert\ f_3{}^*$
M2	$(f_1\ \vert\ f_3)^*\ f_2\ \ f_4{}^*\ \vert\ (f_1\ \vert\ f_3)^*\ f_4{}^*$
M3	$f_1{}^*\ f_3\ f_2{}^*\ f_4\ \vert\ f_1{}^*\ \vert\ f_1{}^*\ f_3\ \vert\ f_1{}^*\ f_3\ f_2{}^*\ \vert\ f_2{}^*\ f_4\ \vert\ f_2{}^*\ \vert\ f_3$

$\{$P1: $y+uz=xz\}$
f_4
$\{$P1: $y+uz=xz\}$
f_1
$\{$P4: $y+(u-1)z=xz\}$
f_3
$\{$P4: $y+(u-1)z=xz\}$
f_2
$\{$P1: $y+uz=xz\}$
f_4
$\{$P1: $y+uz=xz\}$
f_1
$\{$P4: $y+(u-1)z=xz\}$

Fig. 5 Tabular proof of {P1} *f*4 *f*1 *f*3 *f*2 *f*4 *f*1 {P4}.

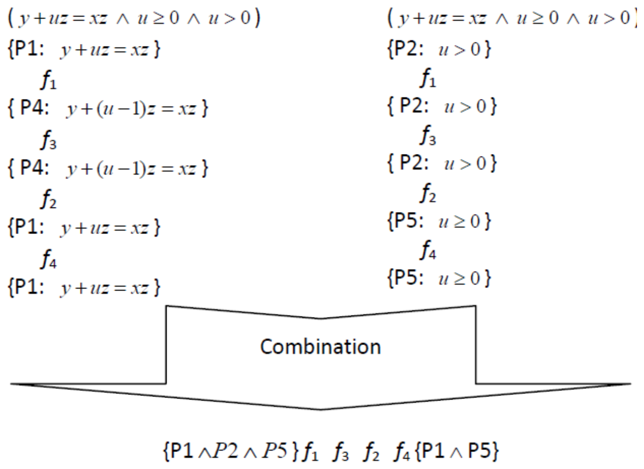

Fig. 6. Proof of {P1∧ P2 ∧ P5 } f1 f3 f2 f4 {P1 ∧ P5}.

5. Discussion

Up to now, we have introduced a relatively simpler way to verify and generate computer programs. This system shares commonness with some of the previous work [11] that combines formal methods [7], [9] like Hoare's logic [1], model checking [8]-[10], automaton [14]-[15] within genetic programming framework [19], but has unique characteristic in depicting the transition structure, and parallelisms. Traditionally, GP [18] automatically generates computer programs in terms of the principle of software testing [20], and don't touch formal semantics which are of concern to GP researchers recently [21] and recognized as important approaches to software reliability. We have made first research attempt in some of these areas. In view of the fact that reusability, component-based development method are of increasingly important topics in computer community, we provided component-based framework in this work and [11]-[13] for verification and generation of programs. Reference paper [11] is the basis of [12]-[13] which provide model alternatives to grammar-based genetic programming, applying path searching techniques in program generations, but it also leaves behind many worthwhile problems. To this end, the present approach is proposed. The major problems to be solved, as seen in this work, range from reducing the complexity of states of [11], automatically finding of sub-models, and component-based parallelism of computations.

6. Conclusion

A component-based verification framework developed under a closed environment is given for a reflection of Hoare's semantics embedded in sequential program structure. It can not only be regarded as a service-oriented model, but also a fundamental model of branches and iterations to be further extended. Our future work will focus on the model simplification, sub-model discovery, and related application in GP unification , information security [22], search-based software engineering [19], and some challenge problems [23], etc.

Acknowledgment

The research work was supported by National Natural Science Foundation of China (Grant No. 61170199, 61503087), the Scientific Research Fund of Education Department of Hunan Province, China (Grant No.11A004), the Natural Science Foundation of Guangdong Province, China (Grant No. 2015A030313501), and Guangzhou Teaching Reform Project (2013A022).

References

[1] Hoare, C. A. R. (1969). An axiomatic basis for computer programming. *CACM, 12(10)*, 576-583.

[2] Manna, Z. (1974). *Mathematical Theory of Computation*. New York: McGraw-Hill

[3] Winskel, G. (1993). *The Formal Semantics of Programming Languages: An Introduction*. Cambridge, MA: MIT Press.

[4] Zhang, Y. Z., & Xu, B. W. (2004). A survey of semantic description frameworks for programming language. *ACM SIGPLAN Notices, 39(3)*, 14-30.

[5] Huth, M., & Ryan, M. (2004). *Logic in Computer Science: Modeling and Reasoning about System*. Cambridge: Cambridge University Press.

[6] Manna, Z. (1971). Mathematical theory of partial correctness. *Journal of Computer and System Sciences (JCSS), 5(3)*, 239-253.

[7] Clarke, E. M., Wing, J. M. *et al*. (1996). Formal methods: state of the art and future direction. *ACM Computing Surveys, 28(4)*, 626-643.

[8] Havelund, K., & Visser, W. (2002). Program model checking as new trend. *International Journal on Software Tools for Technology Transfer, 4(1)*, 8-20.

[9] Quemada, J. (2004). Formal description techniques and software engineering: Some reflection after 2 decades of research, *LNCS 3235,* pp. 33-42.

[10] Rus, T., & Wyk E. V. (2002). Tom Halverson. Generating model checker from algebraic specification. *Formal Methods in System Design, 20(3)*, 249-284.

[11] He, P., Kang, L. S., Johnson C. G., & Ying, S. (2011). Hoare logic-based genetic programming, *Science China Information Sciences, 54(3)*, 623-637.

[12] He, P., Johnson, C. G., & Wang, H. F. (2011). Modeling grammatical evolution by automaton. *Science China Information Sciences, 54(12)*, 2544-2553.

[13] He, P., Deng, Z. L., Wang, H. F., & Liu, Z. S. (2015). Model approach to grammatical evolution: Theory and case study, *Soft Computing.*

[14] Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2007). *Compilers: Principles, Techniques, and Tools* (2nd ed.). Pearson Education, Inc.

[15] Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2008) *Automata theory, languages, and computation (3rd edition)*. Pearson Education, Inc.

[16] O'Neill M., & Ryan, C. (2001). Grammatical evolution. *IEEE Transactions on Evolutionary Computation, 5(4)*, 349-358.

[17] Sabar, N. R., Ayob, M., Kendall, G., & Qu, R. (2013). Grammatical evolution hyper-heuristic for combinatorial optimization problems. *IEEE Trans on Evolutionary Computation, 17(6)*, 840-861.

[18] Koza, J. R. (1992). Genetic programming: On the programming of computers by means of natural selection. Cambridge, MA: The MIT Press.

[19] Harman, M., Mansouri, S. A., & Zhang, Y. Y. (2012). Search-based software engineering: trends, techniques and applications. *ACM Computing Surveys, 45(1)*, 11:1-11:61.

[20] Juristo, N., Moreno, A. M., & Vegas, S. (2004). Reviewing 25 years of testing technique experiments. *Empirical Software Engineering*, *9(1-2)*, 7-44.

[21] Vanneschi, L., Castelli, M., & Silva, S. (2014). A survey of semantic methods in genetic programming, *Genet Program Evolvable Mach*, *15(2)*, 195-214.

[22] Li, J, Huang, X. Y, Li, J. W., Chen, X. F, & Xiang, Y. (2014). Securely outsourcing attribute-based encryption with checkability. *IEEE Transactions on Parallel and Distributed Systems*, *25 (8)*, 2201-2210.

[23] Hoare, T. (2003). The verifying compiler: A grand challenge for computing research. In G Hedin (Ed.), CC2003, *LNCS 2622,* 262-272.

Pei He received the M.S. degree in computer science from Institute of Software, Academia Sinica, and the B.S. and Ph D degrees in computer architecture and computer software and theory from Wuhan University, China in 1986 and 2008 respectively. He is currently a professor at Guangzhou University, China and a member of Guangdong Advisory Board on Higher School Computer Teaching. Before joining Guangzhou University, he was with school of computer and communication engineering at Changsha University of Science and Technology, China from 1989 to 2013. Prof. He Pei works on artificial intelligence, formal methods, and evolutionary computation, and has published many scientific papers in these areas.

Achun Hu is a librarian at Guangzhou University, China. She graduated from Hunan Normal University, majoring in linguistics and library & information. From 2002 to 2013, Ms Hu was with Changsha University of Science and Technology. Her work ranges from acquisition, cataloguing, information retrieval, to knowledge service, etc. Up to now, Ms Hu has published several papers. Her current interests center on knowledge representation and recommendation service.

Dongqing Xie received the B.S. and M.S. degrees in applied mathematics and computer software from Xidian University, China in 1985 and 1988 respectively, and the Ph D. degree in applied mathematics from Hunan University in 1999. He is currently a professor at Guangzhou University, China. His interests are in applied mathematics, network security, cryptography, etc. Prof. Xie is the dean at school of computer science and educational software, Guangzhou University, a member of advisory board on information security under Chinese ministry of education, a senior member of Chinese Institute of Electronics. He has published many scientific papers.

Zhiping Fan received the Ph D degree in agricultural electrification and automation from South China Agricultural University and the M.S. degree in computer technology from Sun Yat-Sen University in 2011 and 2005 respectively. He is currently an associate professor at Guangzhou University, China. He works on artificial intelligence, wireless sensor networks, and has published many scientific papers in these areas.