Academic Journal of Computing & Information Science
ISSN 2616-5775 Vol. 2, Issue 1: 22-34, DOI: 10.25236/AJCIS.010015

A Survey on Formal Specification and Verification of System-Level Achievements in Industrial Circles

Feng Zhang, Wensheng Niu

School of Computer Science and Engineering, Beihang Univerisity, Beijing, China
zhang_wenruo@outlook.com

ABSTRACT. *Formal methods have been applied more and more in industrial circles. They use mathematical logic and rigorous models for analysis and verification, can be used at all the system life cycles, and provide verified software without bugs with respect to certain properties. The increasing industrial applications show that formal methods not only are theoretical research anymore, but also can be deployed in many concrete industrial applications. This paper surveys the important formal specification and verification of system-Level achievements in industrial circles.*

KEYWORDS: *formal methods, formal specification and verification, system-Level achievements in industrial circles*

1. Introduction

Formal methods have been applied more and more in industrial circles. They use mathematical logic and rigorous models for analysis and verification, can be used at all the system life cycles, and provide verified software without bugs with respect to certain properties. The increasing industrial applications show that formal methods not only are theoretical research anymore, but also can be deployed in many concrete industrial applications.

Formal methods are mathematical methods that support the strict specification, design, and verification of computer systems [1]. The mathematical foundations used in formal methods include mathematical logic, discrete mathematics, computer languages, etc., which aid in the establishment of correct, robust software design through mathematical analysis. The formal approach ensures that: (1), accuracy to describe the requirements of the software system; (2), no-confusion to achieve accurate communication between the engineers; (3), to provide accurate and consistent software requirements for the use of formal methods verified evidence; (4), providing proof of conformity and consistency between formal descriptions.

The contents of formal methods contains two main elements: the formal specification and formal verification. A **formal specification** is a software functional description developed using a formal language with strict grammar and semantics, which is the basis for the post-design, coding, and verification process. At present, there are two major classifications of specification methods: functional programming and state machine. Functional programming can be used in later interactive theorem proving, fine-grained formalization of objects of any abstraction layer, or refinement construction. There are many types of state machines, such as Timed Automata and Label Transition Systems. **Formal verification** is to verify that the system implementation conforms to the formal specification of the system. Traditional verification methods include review, analysis, and testing. Formal verification can be mathematical reasoning, simulation, simulation, and testing. Formal verification strives to achieve exhaustive testing of software, automated testing, and Conducted at the earliest stages of software development.

There are four important review articles on formal methods in recent years that can be used by readers for reference, which are: [2] published by Abrial, the founder of Method B, in 2007, [3] published by York University's Woodcock J in 2009, and [4] and [5] published by Zhao Yongwang in 2014 and 2017 respectively. Among them, the literature of Abrial and Woodcock J is mainly about the successful application of formalization in industry. Zhao Yongwang's research is more detailed and targeted. It mainly introduces the application of formal methods in Separation Kernels, which include both software and hardware.

2. Formalization of Application Software

2.1 Airbus & Boeing

The large-scale and successful use of formal methods by Airbus in a number of civil aircraft is a strong force in the industrial application of formal methods. There is nothing more persuasive than the large-scale application of formalized technology on large, commercial and civil aircraft used by the world. Boeing also uses formal methods in some new models. The same thing is that the formalization of the application software part uses SCADE.

At the beginning of this century, Airbus began to develop the SCADE toolset [6] on a large scale into the development of key software for civil aircraft from their A340-500 models, including the Electric Load Management Unit (ELMU) and Backup Flight Control Computer (FCSC).

At the same time, the A380 successfully used the formal method in the avionics certification phase [7]. The literature [6] lists several advantages of using the formalized SCADE toolset: (1) The code is more automated and the code errors are significantly reduced: 70% of the A340 aircraft's code is automatically generated. (2) Shorter time to change requirements: The SCADE toolset makes A340's requirements change management faster and traceability improved. (3) Increase productivity: Although each new Airbus requires about twice as much software as its predecessor.

In view of the previous successful experience, Airbus continued to use the formal method on a large scale in the world's largest civilian passenger aircraft A380, mainly based on the SCADE toolset. In fact, most of the onboard computers developed by Airbus and its suppliers benefit from the use of the SCADE toolset. The SCADE package is used to develop most of the key onboard software for the A380 and A400M military transport aircraft, as well as the SCADE Suite for the A340-500/600 passenger aircraft's auxiliary flight command system; the A380 and A400M cockpit control display systems and airborne The display of the airport navigation system was developed using SCADE Display and they all conform to the specifications of the graphical interface.

The Boeing 787 Dreamliner also uses the SCADE toolset to develop applications. Unfortunately, we were unable to find an official Boeing official statement, but only in SCADE's official technical report [6], the Boeing 787's landing gear and brake system used SCADE.

2.2 Paris Metro

This article examines three Paris Metro application cases that use formal methods.

In 1989, the SACEM system [8] [9] using the B method was successfully delivered to the Paris Metro. SACEM was proposed from the improvement project of the fast network commuter train in the RER region of Paris [10]. The improvement was initiated in 1988 by GEC Alsthom, MATRA Transport and RATP (Paris Public Transport Operator). They jointly studied computer signal systems used to control fast network commuter trains in the RER region of Paris. The goal of the signal system is to increase commute traffic by 25% while maintaining the existing level of safety. After the use of SACEM, it successfully operated on the A line train in the RER area of Paris. As of this, the resulting SACEM system with embedded hardware and software was delivered in 1989 and controlled the speed of the Paris RER A train, which has involved 7 billion passenger journeys since its introduction. The proposal of SACEM originated in 1988

The formal specification in the SACEM system uses the B method [11], and the formal proof obligation is automatically generated, but the proof process is done manually. The verification of the entire system (including non-safety critical procedures) took approximately 100 person-years. At the beginning of the SACEM project, communication between formal team members and signal engineers was a dilemma, as signal engineers generally did not understand the B method. To this end, the B-method training was specially conducted for the signal engineering personnel to solve the communication problem between the two teams. SACEM's approach to ensuring security includes formal protocols and certifications for online error detection, software verification, and fault tolerance for on-board ground composite systems.

In view of this successful application, the Paris researchers deployed two traffic projects using the B method: Paris Metro Line 14 [12] and Paris Charles de Gaulle Airport commuter shuttle [13]. In these two systems, the key software part accounts for about 1/3 of the total software. These key software parts are developed using the B method. The operation of Metro Line 14 is fully automated, with safety critical aspects involving the operation and stopping of trains and the opening and closing of train and platform doors. No unit tests were performed on the Line 14 or Roissy Shuttle projects, which were replaced by some of the most successful overall tests. This significantly reduces overall development costs.

3. Formalization of OS Kernels

The formal proof of the Australian seL4 [14] [15] kernel is a milestone in the complete verification of system-level software, especially the underlying software operating system kernel.The seL4 embedded OS kernel is an evolution of the L4 microkernel that enhances the security feature. The seL4 microkernel includes 8700 lines of C and 600 lines of assembly, proving that the process uses approximately 100,000 lines of Isabelle/HOL protocol and proof code. A total of 160 defects were discovered during the entire validation process, 16 of which were discovered during the testing phase, while the remaining 144 were discovered during the proof of use form of Isabelle/HOL.

This chapter will introduce two other important OS kernels that use formal methods.

3.1 INTEGRITY-178B

INTEGRITY-178B [16] is an operating system product chain developed by Green Hills that focuses on software certification and guarantees a high level of safety & security. INTEGRITY-178B is a real-time, partitioned operating system that complies with DO-178B's highest-rated Class A safety standard (2002) and is the first EAL 6 to comply with the security standard Common Criteria (the highest level is EAL 7,) 2008) operating system. It has been successfully applied to military aircraft such as the B-2, F-16, F-22 and F-35, as well as the Airbus A380 large passenger aircraft. The INTEGRITY-178B also supports multi-core hardware platforms.

The Security Standard Common Criteria requires that, starting with EAL 5, a formal method must be used. If you are conducting EAL Level 7 certification, you must also have a formal model and proof [17] for the entire development process. Finally, the product will be submitted to the IAD (Information Assurance Directorate) for information flow penetration testing. SKPP specifically requires the “NSA Evaluator” to perform vulnerability analysis, penetration testing and covert channel review. The assessment includes these specific activities and unconstrained searches for vulnerabilities. The certification process requires submission of some certification evidence, such as formal verification. These formal verifications are required to demonstrate the abstract functionality of the kernel unrelated to the hardware platform [18].

INTEGRITY-178B uses GWV [19] as a security policy, and uses ACL2 to create a three-layer protocol, namely functional requirements specification, advanced and low-level design. Functional conventions are the formalization of interfaces. The other two are semi-formal descriptions of systems of different abstraction levels. The low-level design directly corresponds to the implementation, simplifying the “code-to-protocol” analysis requirements during CC certification. The GWV-based strategy they adopted is based on the MASK data separation strategy, which is designed to model a split kernel that can implement partitioning of applications running on a single-processor system. The GWV attribute requires that the execution step of modifying any memory segment must follow the mapping of a set of memory regions bound to the current partition and allow interaction with that memory segment.

3.2 CertiKOS & mCertiKOS

Yale University's CertiKOS [20] is an embedded OS microkernel for cloud computing security issues. It addresses the issue of functional correctness and sensitive information disclosure issues to be addressed during the certification process. For traffic security, CertiKOS researchers used Coq to formalize the information flow control of the kernel runtime. For ease of certification and formalization, CertiKOS uses a highly modular design that divides complex systems into smaller modules that are formalized and certified one by one. However, CertiKOS is not a fully formalized kernel. For example, virtual memory components are not subject to formal specifications and certification. In addition, the complex resource management algorithms in cloud virtualization are not formalized, but the permissions check is implemented in the kernel. This eliminates complex policy management and scheduling tasks from the kernel without sacrificing functionality, thereby reducing formal authentication of kernel code.

In the literature [21] and [22] publied in 2015, the research team proposed the concept of deep specifications to fully complex the complex system-level software.

The basis for their deep statutes is the abstraction of software. They believe that "modern computer systems consist of multiple layers of abstraction (eg, OS kernels, hypervisors, device drivers, network protocols, etc.), each of which defines an interface to the implementation details of the hidden functions. Built on each layer The client program on top can be understood based on the interface only, without having to care about the concrete implementation of each layer. Although they are of obvious importance, the abstraction layer is mainly regarded as a system concept; they are almost never officially specified or verified. This makes it difficult to establish strong correctness attributes and it is difficult to verify [21] across multiple layers of extensions." Accordingly, they proposed a deep specification: a powerful abstract form of a rich set of protocols. Just as data abstraction in a typed function language leads to important representation independence, the abstraction of depth conventions is characterized by important implementation independence. This deep implementation requires that any two implementations of the same depth specification must have context-equivalent behavior. They also proposed a new layer calculus framework for formal protocol, programming, validation, and abstraction layer combinations. An important feature of the deep protocol concept is that the concept framework uses a formal-certified and certified CompCert [23] compiler to compile C into a complete assembly object code.

mCertiKOS [24] is a fully formalized operating system microkernel based on the deep protocol concept. To achieve full kernel formalization, mCertiKOS is a single-processor version of the simplified implementation of CertiKOS, designed for 32-bit x86 architectures, which also uses Coq form tools. mCertiKOS uses a separate virtual address space to provide a multi-process environment for applications where communication between different applications is established through messaging. mCertiKOS used 9.5 person-months in the planning and development phase and 2 people in the link phase.

4. Formalization of Compilers

The CompCert [23] project presents a thorough, math-based solution to the problem of miscompilation: a form of the compiler itself, proof of the tool's operation. Applying program-proven techniques to the source code of the compiler, mathematical determinism can be used to prove that the behavior of the executable code generated by the compiler is exactly the same as the semantics of the source C program, thus eliminating all risks of compile errors [25]. Compiler validation is not a new idea: the first compiler correctness proof (for converting arithmetic expressions to stack machines) was published in the literature [26] in 1967, and then used Stanford LCF Proof Assistant in 1972. Make mechanization proof [27]. Since then, compiler formal verification has been the subject of much academic research.

The CompCert project brings a range of work to a complete, optimized compiler, not for the production of critical embedded software systems. CompCert is the first compiler to use formal authentication and optimization and is IEC 60880 certified. It is a "lightweight optimization", machine-validable C compiler that is designed for the aerospace industry and contains a subset of C. The most important omission in CompCert verification is that it does not support concurrent and separate compilation, both of which are current research topics.

Xavier Leroy of the INRIA Institute in France led the team to develop CompCert and formalized it using Coq. The reason for using Coq is that the compiler can be extracted from the proof of correctness. The CompCert implementation itself and the formalization process contained a total of 42K lines of code, which took three years. CompCert supports online evaluation, or you can purchase [28] commercially. The code generated by Compcert is good, but at a fair speed. It generates code at twice the speed of gcc at level 0, 7% slower than gcc-01, and 12% slower than gcc-02. On PowerPC, the performance of the generated code is approximately 90% of GCC version 4 at optimization level 1. For use in the aerospace industry, CompCert has passed the DO-178B certification requirement [29]. It supports PowerPC, ARM, RISC-V and x86 (32 and 64 bit) platforms.

The difference between CompCert and other compilers is that it is officially verified with Coq, which eliminates the problem of miscompilation on some targeted issues. That is, in the proven field, the executable code it produces is proven to be identical to the semantics of the source C program. CompCert opens up a viable path to the use of formal methods in the compiler world, and the results of this project are unprecedented in the compiler world. Yale's CertiKOS [20] and mCertiKOS [24] kernels and the deep protocol concept [21] both use CompCert as a compiler to ease verification and certification. CompCert's formal proof covers everything from abstract syntax trees to generating assembly code. To preprocess and generate executable object files, CompCert also uses external C precompilers, linkers, assemblers, and C libraries. These processing stages are not formally verified, but they are also well understood and robust. They claim that errors in intermediate processing stages found in other compilers have been eliminated in CompCert. As of early 2011, the development version of CompCert was the only compiler that Csmith could not find the error code. They spent about six CPU years completing the task in the Csmith test. CompCert uses formal verification to guarantee an excellent unbreakable feature. At the same time, it provides a proof framework for developing and optimizing the compiler, which is used to perform machine-enable security checks on the compiler.

5. Formalization of Processors

5.1 AAMPs

In 1995, Miller and Srivas published their formal use of PVS [30] on the AAMP5 chip microcode in the literature [31][32]. The work was initiated by NASA, and Rockwell Collins and SRI International (Stanford International Research Institute) jointly completed. AAMP5 is a proprietary microprocessor within Rockwell Collins. It has a stack-based architecture, a large instruction set, and extensive use of microcode, a pipelined architecture and complex processing units with overall performance between Intel 386 and 486.

Rockwell Collins formalizes AAMP5 at both the register transfer level and the instruction set level, and establishes a refinement relationship between the two to prove the correctness of the microcode instructions. The main experience of the project is to demonstrate the feasibility of the application of the formal method on the microprocessor. Second, the development engineer can read and write the form code after training. The unexpected result of motivating engineers was that they found two errors in the procedural process, even if they had not yet reached the formal proof stage.

Formal engineering for AAMP5 has encountered difficult problems in progress: labor costs are too large, and an average of 300 people per instruction is required. However, the project team believes that this is the first time they have learned, used

PVS and formal methods. If similar projects are carried out in the future, it should be possible to significantly reduce labor time. Therefore, they then started a follow-up project: Proof of the microcode for the AAMP-FV chip [33]. The project is still being initiated by NASA and is being developed by Rockwell Collins and SRI International. In addition to verifying the chip itself, the project has another experimental goal: skilled formalists can significantly reduce formal engineering time to an acceptable level. The progress of the AAMP-FV project also confirms the predictions of formalists: proficiency in formal methods and the accumulation of reusable code or modules for previous projects, resulting in a significant reduction in the progress of this project. . Although AAMP-FV is more complicated for AAMP5.

What's even more exciting is that Rockwell Collins's formal work for the AAMP7 chip, [34], received the highest level of EAL-7 (US) National Security Agency security certification for the Common Criteria [17] standard. Multiple independent security level devices for encryption applications. The AAMP7 chip uses a microkernel architecture that supports partitioning. The formalization work uses the ACL2 [35]. The project established a line-by-line protocol model with AAMP7 microcode and added some security features for inter-partition communication in the protocol model.

5.2 Transputers

As early as the late half of the 1980s, Inmos et al. used Occam [36] [37] (a simple, low-level, operational process algebraic CSP subset) to implement the Transputer [38] series of microprocessor chips. Formal work. Transputer is designed for parallel processing. Among them, T800 includes floating point calculation, 32-bit reduced instruction set, memory, four bidirectional communication links, and so on. The T9000 is more complex, and it also includes a memory model and a processor pipeline.

As the scope of the test was clearly confirmed, Transputer developers began formalizing the use of correct-by-construction formal methods to build floating-point units.

They first used the Z [39] [40] method to formalize the IEEE-754 floating-point arithmetic standard for natural languages [41] and revealed some of the problems in the standard. For example, the standard requires that diagnostic information for invalid operations (such as the square root of a negative number) be propagated through further operations, but this is sometimes impossible.

Next, their task is to prove that floating-point packages written in Occam and used in previous Transputers are the correct implementation of IEEE-754. Trial verification using Hoare Logic [42] found several errors in rounding and remainder operations. Occam is too abstract, it can't be used directly in hardware design. The Occam conversion system only applies the rules of Occam programming to generate equivalent microcode programs.

Although Occam is still immature and has many flaws, using it for formal development of floating-point units is at least three months faster than simultaneous informal development [43]. More importantly, two errors were found in floating-point microcode using Occam [44]. Inmos, the head of the University of Oxford and Transputer, also won the Queen's Technology Achievement Award in 1990.

6. Formalization of the Multi-Level Systematic HACMS

Aware of the many important achievements of the formal approach, the Defense Advanced Research Projects Agency (DARPA) launched a multi-level, system-level formal project in 2012 to address military traffic. The security of the tool prevents the military vehicles, especially the unmanned traffic equipment, from being hijacked by the network. This project is called "High-Assurance Cyber Military Systems" (HACMS [45] [46]). Such military transportation systems may be military unmanned vehicles, military unmanned aerial vehicles, and the like.

The goal of HACMS is to create technologies that build high-assurance CPS systems. The high guarantee here is defined as functionally correct and satisfies the appropriate safety and security. Achieving this goal requires a completely different approach than mainstream software engineering. Therefore, the goal of HACMS is to adopt a clean-slate, formal-based software engineering approach; at the same time, this formalization method is required to generate source code automatically or semi-automatically. In addition to generating code, the project requires proof of machine-checkable proof that the generated code meets functional requirements as well as safety and security policies. A key technical challenge is to develop techniques to ensure that such evidence can be combined, allowing the construction of high-assurance systems using high-assurance components. It can be seen that the project is different from other research projects. Its main goal is to use the existing formal method results to integrate different forms of different formal methods into a military transportation system. The project is more focused on the security of military vehicles that are linked via a network rather than physically linked.

HACMS brings together many well-known research units, such as universities MIT, Princeton, UCLA, Boeing, research units NICTA, SRI and so on. At the same time, HACMS integrates several existing mature formalizations, such as the seL4 operating system, the CertiKOS operating system, the CompCert compiler, and the SCADE formal toolset. The entire HACMS consists of five technical areas (TA), they are [46]: TA1-military vehicles, TA2-operating system, TA3-control system, TA4-research integration and TA5-red team (Read Team). The red team in TA5 is the blue army in the military exercise. It is responsible for the security attacks in the network environment for the traffic systems developed by other teams.

Different from the general research projects of universities and research institutions, the biggest feature of HACMS is that it has convened a group of cyber attack experts, Red Team, to carry out security attacks in the network environment for the vehicles developed by the project to establish a safety assessment baseline. And iteratively modify the traffic system software. HACMS is divided into three phases in time, each phase being 18 months. HACMS requires that the developed system be delivered to the Red Team for security attack testing and evaluation during the final period of each phase. During this time of attack, Red Team found a serious vulnerability [45] on all platforms. For example, the Red Team can control an in-flight quad-copter like a legitimate operator and prevent the aircraft from rejecting legitimate commands.

7. Conclusion

Following the achievements of seL4, the completion of CompCert, CertiKOS and HACMS projects greatly invigorated the confidence of formal researchers. Most importantly, the industry has seen the commercial viability of formal application in commercial and industrial grades from the results of these system-level projects.

The current stage is a period of rapid advancement in the theory of formal methods and its applications. From the current results, formal methods may achieve more results in the last decade or two and become more widely used in industry. There are three main reasons:

1) Improvements of hardware computing power} makes it possible to perform some formal simulation and proof work that could not be completed, even if these formal methods have not made any progress.

2) More and more formalization tool [46] can be used in a variety of fields, including aerospace, automotive, and networking. Well-known formal methods or tools now known include: Isabell/HOL, Event-B&Rodin, STCSP&PAT, Coq, ACL2, Alt-Ergo, Astree, Bedrock, Boogie, CVC4, Frama-C, KLEE, PVS , SLAM, TLA+, VCC, Nices2, Z3, etc.

3) The formalization tool of more and more perfect can greatly reduce labor time. Most of these tools can be developed for a specific domain, reusable module library, and enhance the automatic proof function as much as possible.

References

[1] Formal Methods Europe. Formal methods. http://www. fmeurope.org/ formalmethods.
[2] Jean-Raymond Abrial. Formal methods: Theory becoming practice. Journal of Universal Computer Science, 2007, 13 (5): 619-628.
[3] Woodcock J., Larsen P. G., Bicarregui J., et al. Formal methods: Practice and experience. ACM computing surveys (CSUR), 2009, 41 (4): 19: 1-19: 36.
[4] Zhao Yongwang, Ma Dianfu, and Yang Zhibin. 2014 A survey on formal specification and verification of separation kernels. Technical report, Tech. rep., National Key Laboratory of Software Development Environment (NLSDE). Beihang Univerisity.
[5] Zhao Yongwang., Sanán David, Zhang Fuyuan, et al. High-assurance separation kernels: a survey on formal methods. arXiv preprint arXiv:1701.01535, 2017.

[6] Berry, G. 2008. Synchronous design and verification of critical embedded systems using SCADE and Esterel. Lecture Notes in Computer Science, In Formal Methods for Industrial Critical Systems. Heidelberg: Springer, 2008, 4916: 2-2.

[7] Souyris, J. , Wiels, V. , Delmas, D., et al. Formal Verification of Avionics Software Products. Proceedings of The International symposium on formal methods. Berlin: Springer, 2009: 532-54.

[8] Guiho G. and Hennebert C.. SACEM software validation. Proceedings of the 12th International Conference on Software Engineering. Nice: IEEE Computer Society Press, 1990: 186-191.

[9] Hennebert C. and Guiho G.. SACEM: A fault tolerant system for train speed control. Proceedings of the 23th International Symposium on Fault-Tolerant Computing. Toulouse, France: IEEE Computer Society Press, 1993: 624-628.

[10] Jonathan Bowen and Victoria Stavridou. Safety-Critical Systems, Formal Methods and Standards. Software Engineering Journal, 1993, 8 (4): 189-209.

[11] Jean-Raymond Abrial. The B-book: Assigning Programs to Meanings. Cambridge: Cambridge University Press, 1996.

[12] Behm P., Benoit P., Faivre A, et al. M\eteor: A successful application of B in a large project. International Symposium on Formal Methods. Berlin, Heidelberg: Springer, 1999: 369-387.

[13] Badeau F. and Amelot A. Using B as a high level programming language in an industrial project: Roissy VAL. International Conference of B and Z Users. Berlin, Heidelberg: Springer, 2005: 334-354.

[14] Klein G., Elphinstone K., Heiser G., et al. seL4: Formal verification of an OS kernel. Proceedings of the 22nd ACM Symposium on Operating Systems Principles. ACM, 2009: 207-220.

[15] Murray T., Matichuk D., Brassil M., et al. seL4: From General Purpose to a Proof of Information Flow Enforcement. Proceedings of The IEEE Symposium on Security and Privacy. IEEE Press, 2013: 415-429.

[16] GreenHills. INTEGRITY-178B RTOS. http://www.ghs.com/products/safety critical/integrity-do-178b.html. (2015). Accessed: 2015-07.

[17] Common Criteria. Common criteria for information technology security evaluation (3.1 r4 ed.). National Security Agency, 2012.

[18] National Information Assurance Partnership (NIAP). Separation Kernels on Commodity Workstations. http://www.niap-ccevs.org/ announcements/ Separation\%20Kernels\%20on\%20Commodity\%20Workstations.pdf. 2010.

[19] Greve, David, Matthew Wilding, and W. Mark Vanfleet. A Separation Kernel Formal Security Policy. Proceedings of The Fourth International Workshop on the ACL2 Theorem Prover and Its Applications. 2003.

[20] Liang Gu, Alexander Vaynberg, Bryan Ford, et al. CertiKOS: a certified kernel for secure cloud computing. Proceedings of the Second Asia-Pacific Workshop on Systems. ACM, 2011: 3.

[21] Gu R., Koenig J., Ramananandro T., et al. Deep specifications and certified abstraction layers. Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - (POPL). 2015: 595-608.

[22] Appel A. W., Beringer L., Chlipala A., et al. Position paper: the science of deep specification. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 2017, 375 (2104): 20160331.
[23] Xavier Leroy. The CompCert C verified compiler. Documentation and user's manual. INRIA Paris-Rocquencourt, March 2012. http://compcert. inria.fr/man/ manual.pdf.
[24] David Costanzo, Zhong Shao, and Ronghui Gu. End-to-end Verification of Information-flow Security for C and Assembly Programs. Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). In Proc. of PLDI'16. ACM Press, 2016: 648-664.
[25] Xavier Leroy. Formal verification of a realistic compiler. Communications of the ACM, 2009, 52 (7): 107–115.
[26] John McCarthy and James Painter. Correctness of a compiler for arithmetical expressions. Proceedings of the Symposia in Applied Mathematics. American Mathematical Society, 1967: 33–41
[27] Robin Milner and Richard Weyrauch. Proving compiler correctness in a mechanized logic. Proceedings of the 7th Annual Machine Intelligence Workshop. Edinburgh University Press, 1972: 51–72.
[28] CompCert. See http://compcert.inria.fr.
[29] Franca RB, Favre-Felix D, Leroy X, Pantel M, Souyris J. Towards formally verified optimizing compilation in flight control software [A]. Proceedings of The Predictability and Performance in Embedded Systems (PPES), 2011, 18: 59-68.
[30] Owre, S., Rushby, J. M., and Shankar, N.. PVS: A prototype verification system. Proceedings of 11th International Conference on Automated Deduction. Berlin: Springer, 1992: 748-752.
[31] Miller S.P. and Srivas M.. Formal verification of the AAMP5 microprocessor: A case study in the industrial use of formal methods.Proceedings of 1995 IEEE Workshop on Industrial-Strength Formal Specification Techniques. Boca Raton: IEEE, 1995: 2-16.
[32] Srivas, M. and Miller, S.. Applying formal verification to a commercial microprocessor. In IFIP Conference on Hardware Description Languages and their Applications (CHDL). Makuhari, Chiba, Japan: IEEE, 1995.
[33] Miller, S. P., Greve, D. A., and Srivas, M. K.. Formal verification of the AAMP5 and AAMP-FV microcode. Proceedings of the Third AMAST Workshop on Real-Time Systems. Salt Lake City, Utah. 1996.
[34] Greve D. and Wilding M.. Evaluatable, high-assurance microprocessor. Proceedings of the Second Annual HighConfidence Systems and Software Conference (HCSS). National Security Agency, Linthicum, 2002.
[35] Matt Kaufmann, Panagiotis Manolios, and J Strother Moore. Computer-Aided Reasoning: ACL2 Case Studies. Springer Science \& Business Media, Vol. 4.
[36] Barrett G. occam 3 reference manual. Inmos Limited, March 1992. Available at: http://wotug. ukc. ac. uk/parallel/occam/documentation, 1992.
[37] Wexler J. Concurrent programming in OCCAM 2. New York: Ellis Horwood, 1989.

[38] May David, and Roger Shepherd. The transputer. Neural Computers. Springer, 1989. 477-48.
[39] Spivey J. M. and Abrial J.-R.. The Z Notation, second edition. Prentice Hall International, Hemel Hempstead (U.K.), 1992.
[40] Woodcock, J. and Davies, J.. Using Z: Specification, Refinement, and Proof. International Series in Computer Science. Prentice Hall, 1996.
[41] Geoff Barrett. Formal methods applied to a floating-point number system. IEEE transactions on software engineering, 1989, 15 (5): 611-621.
[42] Hoare C A R. An axiomatic basis for computer programming. Communications of the ACM, 1969, 12 (10): 576-580.
[43] Barrett, G.. 1990. Verifying the Transputer. Proceedings of the first conference of the North American Transputer Users Group on Transputer research and applications. IOS Press, 1990.
[44] Jeremy Gibbons. Formal methods: Why should I care? The development of the T800 Transputer floating-point unit. Proceedings of the 13th New Zealand Computer Society Conference. Auckland: New Zealand Computer Society, 1993: 207-217.
[45] Kathleen Fisher. HACMS: High assurance cyber military systems. Proceedings of the 2012 ACM Conference on High Integrity Language Technology. Boston, USA, 51–52.
[46] Fisher K., Launchbury J., Richards R.. The HACMS program: using formal methods to eliminate exploitable bugs. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 2017: 375 (2104), 20150401.