Formal Specification of Scientific Applications Using Interval Temporal Logic

BOJANA KOTESKA and ANASTAS MISHEV, University SS. Cyril and Methodius, Faculty of Computer Science and Engineering, Skopje
LJUPCO PEJOV, University SS. Cyril and Methodius, Faculty of Natural Science and Mathematics, Skopje

Scientific applications simulate any natural phenomena in different scientific domains. Moreover, the problems they solve are usually represented by mathematical models. Taking that in advance, these problems can be described by using specific formal notation and mathematical formulas. Scientific applications are usually created by the scientists without using any software development engineering practices. Our main goal is to include formal methods in the testing process of scientific applications. In this paper, we adapt Interval Temporal Logic (ITL) as a flexible notation for describing software applications. We use Tempura framework and Ana Tempura tool for specifying the properties of the scientific software system. The correctness of the code is verified by comparing the results from the program output and functions written in Tempura. This process is especially important when some code changes or optimizations are made. To verify this concept we made a formal description of the code for calculating bound states of the Morse oscillator well.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Software/Program Verification —*Validation, Formal Methods*; G.4 [**Mathematics of Computing**]: Mathematical Software—*Certification and testing, Documentation, Verification*

General Terms: Verification
Additional Key Words and Phrases: Scientific application, Formal Specification, Interval Temporal Logic, Ana Tempura, Morse Oscillator

1. INTRODUCTION

Scientific applications are widely used nowadays in different scientific domains. Numerical simulations performed by scientific applications can solve problems in many research fields such as: computational chemistry, physics, engineering, mathematics, mechanics, informatics, bioinformatics, etc. Scientific application is defined as a software application that simulates activities from the real world by turning the objects into mathematical models [Ziff Davis Publishing Holdings 1995]. Simulations of scientific experiments require powerful supercomputers, high performance computing infrastructures, clusters or Grid computing [Vecchiola et al. 2009].

The testing process of scientific applications is not the same as testing of the commercial software applications. Problems related to testing come from the non formal specification of the scientific applications. Scientists are mostly interested in scientific research achievements and they do not have any documentation or formal specification for their software [Segal 2008]. When some code changes or code optimization should be performed it is usually hard to understand the code. According to Kelly and Sanders [Sanders and Kelly 2008], the risks of developing scientific applications can be divided

Author's address: Bojana Koteska, FCSE, Rugjer Boshkovikj 16, P.O. Box 393 1000, Skopje; email: bojana.koteska@finki.ukim.mk; Ljupco Pejov, FNSM, P.O. Box 1001, 1000 Skopje; email: ljupcop@iunona.pmf.ukim.edu.mk; Anastas Mishev, FCSE, Rugjer Boshkovikj 16, P.O. Box 393 1000, Skopje; email: anastas.mishev@finki.ukim.mk.

In: Z. Budimac, T. Galinac Grbac (eds.): Proceedings of the 3rd Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA), Lovran, Croatia, 19.-22.9.2014, published at http://ceur-ws.org.

into three categories: complexity of the theory and the difficulty of its validation; implementation risks (code and documentation) and application usage risk (concerning the use of the application by the target user groups). Validation is difficult because the results sometimes must be compared with the results obtained by the physical experiments. Also, scientists usually visually decide whether the results are satisfactory or not. The survey presented in [Koteska and Mishev 2013] shows that most of the interviewed scientists do not prefer formal software engineering testing methods, do not use any testing tools, do not use any specific test case generation technique and some of them do ad-Hoc testing.

To change the current development practices and to improve testing, we propose adapting the method of formal specification and verification of the scientific applications. We chose Interval Temporal Logic (ITL) as a powerful and flexible mathematical notation for propositional and First-order reasoning about periods of time which is used for describing hardware and software systems. We also use the Ana Tempura tool that is built upon C-Tempura and is used for runtime verification of systems. This tool uses ITL and the executable subset Tempura which is an interpreter for executable ITL formulas [Cau et al. 2002]. As a case study, we chose the code for calculating the bound states of the Morse oscillator well which is described in [Bittner 2009].

The rest of the paper is organized as follows. Related work is given in Section 2. In Section 3, we explain the importance of using the formal notation for scientific application description and we present the benefits of using ITL and Tempura. In Section 4, the ITL formal description of the code for calculating the bound states of the Morse oscillator well is given and the results of the testing process are shown. The conclusion and future work are specified in Section 5.

2. RELATED WORK

There are several research papers that present some methods for formal description of software systems and architectures. An overview of how formal (mathematical) methods can be used in the software development cycle and what methods and tools can support software development is given in [Ostroff and Paige 1998]. Siegel and Avrunin [Siegel and Avrunin 2004] write about issues related to finite-state verification techniques when applied to scientific computation software employing the Open MPI (open source High Performance Message Passing Interface implementation). Siegel and Rossi [Siegel and Rossi 2008] applied model checking techniques to BlobFlow (MPI scientific program consisted of 10K lines of code that implements a high order vortex method for solving the two-dimensional Navier-Stokes equations).ITL has been previously used for describing software or hardware systems, state machines or documents. For example, Sciavicco et al. [Sciavicco et al. 2009] consider the problem of formalizing a medical guideline in a logical language. These guidelines are documents supporting the health-care professionals in managing a disease in a patient. Zedan et al. [Zedan et al. 1999] present an object based formal method for the development of real-time systems which is called ATOM. It is based on the refinement calculus and also the formal specification contains ITL description of the behaviour of a real-time system. El-kustaban et al. [El-kustaban et al. 2012] proposed an executable specification model for an abstract transactional memory (lock-free technique that offers a parallel programming model for future chip multiprocessor systems). They used ITL and AnaTempura to build and validate the model. In [Rossi et al. 2004], the authors use temporal logic that combines points, intervals, and dates to formalize the semantics of UML state machines.

The development of scientific applications does not include software engineering practices or formal methods which means that no requirements, formal specification or any kind of documentation could be found. The problems come later when some code modifications or optimization should be performed and nobody knows what the code function is. ITL is usually used for modeling critical software systems, but it really helps when some code changes are made. For example, code changes must be made when some code optimizations are performed which happens very usually while programming scientific ap-

plications because of the memory lack and CPU performance. The program written in Tempura is one way to check the correctness of the results of the program. It also helps for understanding the order of the statements execution at a given moment of time. If no formal program description is used, then the chances for program errors are bigger and the program is less understandable for people being included in the development process later.

The research shows that no formal specification of the scientific applications using ITL/ Tempura are made. Our main goal is to give a formal code description in order to change the current practices, to give a mathematical representation of the scientific problem and to improve the process of verification.

3. FORMAL METHODS IN SCIENTIFIC APPLICATION DEVELOPMENT

This section presents the benefits of making formal description and verification of scientific applications by using Interval Temporal Logic and Tempura.

3.1 The Benefits of Using Formal Methods

Some of the most important benefits of using formal methods are given below [Hall 2005; Groote et al. ; Sommerville 2007; Woodcock et al. 2009; van der Poll 2010; Jaeger 2010; Clarke and Wing 1996]:

—The formal software description is abstract and precise description which means that a human reader can understand the big picture and all ambiguities can be removed.

—Formal description allows users to make rigorous analysis and to determine useful properties such as consistency or deadlock-freedom.

—Using formal methods when developing complex software results in higher quality, more correct software, and discovering errors that may not have been discovered through traditional testing.

—Formal specification allows users to find the problems and ambiguities in the system requirements.

—Formal methods are used for code verification which is attempt to prove the theorem that if certain condition are satisfied the program will achieve the expected results.

—Formal methods facilitate the production of quality and testing. Maintenance phases are shortened.

—The use of formal methods increases the development correctness confidence and has the potential to eliminate some types of errors in the system.

—Formal methods can increase the understanding of the system by removing inconsistencies, ambiguities and incompleteness.

Formal methods are usually applied to the critical systems development, but there are also critical scientific applications, especially in the bioinformatics research field. These applications have critical implications for life sciences and require strong quality assurance [Umarji et al. 2009].

3.2 Interval Temporal Logic and Tempura

Interval Temporal Logic (ITL) is a formalism that is an extension to standard predicate logic which includes time-dependent operators [Moszkowski and Manna 1984]. The key term of ITL is an interval. An interval is defined as a (in)finite sequence of states where each state is a mapping from the set of variables to the set of values [Cau et al. 2002].

Expression in ITL is defined as:

$exp ::= \quad z \mid a \mid A \mid g(exp_1, ..., exp_n) \mid \imath a : f,$

where z is an integer value, a is a static variable (its value cannot be changed within an interval), A is a state variable (its value can be changed within an interval), g is a function symbol, f is a formula.

Formula in ITL is defined as:

$f ::= \quad p(exp_1, ..., exp_n) \mid \neg f \mid f_1 \wedge f_2 \mid \forall v \cdot f \mid \textbf{skip} \mid f_1; f_2 \mid f^*$,
where p is a predicate symbol, ; is a chop symbol.

Formulas are building inductively as follows:

—Equality: $exp_1 = exp_2$

—Logical connectives: $\neg f$ and $f_1 \wedge f_2$

—Next: $\bigcirc f$

—Always: $\Box f$

The informal semantics can be represented as follows:

—$\imath a : \ f$ - choose value of a such that f holds

—skip - interval with length 1

—$f_1; f_2$ - the interval is decomposed into two intervals (prefix interval such that f_1 holds and suffix interval such that f_2 holds) or the interval is infinite and f_1 holds.

—f^* - the interval is decomposable into a finite number of intervals such that for each of them f holds, or the interval is infinite and it can be decomposed into an infinite number of intervals such that f holds [**?**].

For example, this is a valid ITL formula: $(I = 2) \wedge \bigcirc(K = 3)$. It can be interpreted as follows: in the current state I is 2 and in the next state K will be 3.

Interval Temporal Logic provides a basis for the programming language Tempura. The main syntactic categories in Tempura are: expressions (can be boolean or arithmetic), statements (temporal formulas that can be simple or compound) and locations (places where values are stored) [Moszkowski 1985]. A formula is executable in tempura if the following three characteristics are satisfied: the formula is deterministic, the length of an interval is known and the values of the variables are known through that interval [De Montfort University 2004].

4. A FORMAL SPECIFICATION OF THE CODE FOR CALCULATING BOUND STATES OF THE MORSE OSCILLATOR WELL

In this section, we refer to the problem of finding the bound states of Morse oscillator (i.e. solving the stationary Schrödinger equation for Morse potential) and we present the formal specification of the code by using ITL and Ana Tempura. The Morse oscillator well have not been modeled yet with ITL and Tempura. The reason that we chose it for modeling is that code is simple to be understand and it also can be considered as an example of a simple scientific application.

4.1 Bound States of the Morse Oscillator Well

Calculation of eigenenergies of bound states of Morse oscillator is a prototypical exercise in quantum mechanics. This is so since the particular potential serves as a model system for studying molecular vibrations. Morse potential has the following form:

$$U(r) = D_e \cdot [1 - exp(-\beta \cdot (r - r_e))]^2 \tag{1}$$

where D_e denotes the dissociation energy of the corresponding bond, while r_e is the interatomic distance corresponding to the minimum energy of the oscillator. Vibrational Schrödinger equation with the potential of the form (1) is analytically solvable and the corresponding vibrational eigenenergies are given by:

$$E_v = h \cdot c \cdot [(v + \frac{1}{2}) \cdot \omega_e - (v + \frac{1}{2})^2 * \omega_e x_e] \tag{2}$$

In eq. (2), v is the vibrational quantum number ($v \in 0, 1, 2,$), $\omega_e x_e$ is the so-called anharmonicity constant and is related to the parameter β and the reduced mass of the oscillator μ by:

$$\beta = 2 \cdot \pi \cdot c \cdot \omega_e \cdot \sqrt{\frac{\mu}{2 \cdot D_e}} \tag{3}$$

All other symbols in (2) have their usual meanings. The fact that the vibrational Schödinger equation for Morse oscillator is analytically solvable makes this system a rather convenient test case for a number of numerical methods aimed to solve the quantum vibrational problem.

In our particular application of the formal code specification approach, we solve the vibrational eigenvalue problem by the discrete variable representation methodology [J. C. Light and J. V. Lill 1985], following closely the approach adopted by Bittner [Bittner 2009]. To solve a problem in quantum mechanics by using numerical methods, the Hamiltonian operator should be represented in a finite polynomial basis. In this case, the Tchebychev polynomials are used as a basis.

One of the methods we made a formal specification for is ***thcheby(...)***. It returns a set of points pts[NPTS], the eigenstates the Laplacian operator, $(-^2\partial/\partial x^2)$, in the basis ke_fb[NPTS] (kinetic energy in finite basis), set of weights w[NPTS] and a transformation matrix T[NPTS][NPTS]. NPTS is the number of points. There are two representations: finite basis representation (FBR) and discrete variable representation (DVR). The transformation matrix carries one from the FBR to a DVR.

```
double thcheby(double xmin, double xmax, double pts[],double ke_fb[],
double w[],double T[][NPTS])
{
double  del, fb;
int i,j;
del=xmax-xmin;
for(i=0;i<NPTS;i++)
{
    pts[i]=((i+1)*del)/(NPTS+1)+xmin;
    ke_fb[i]=square((i+1)*M_PI/del);
    w[i]=del/(NPTS+1);
    for(j=0;j<NPTS;j++)
    {
        T[i][j]= sqrt(2.0/(NPTS+1))*sin(((i+1)*(j+1)*M_PI)/(NPTS+1));
    }
}
}
```

To solve the bound states of the Morse oscillator well we should define the number of points: NPTS= 100, and range: $xmin = -3$ and $xmax = 32$. The goal is to construct the Hamiltonian matrix in the DVR basis and then diagonalize it to determine the eigenvalues and eigenvectors. The eigenvalues (energies) below 0 are bound states. We use the eigenvectors and eigenvalues to plot the wave functions. We automated the part for checking the correctness of wavefunctions by checking the values of the first two eigenvectors. A wave function is correct if the difference between two neighbor values is smaller than 10^{-3} and the values gradually getting tend to zero.

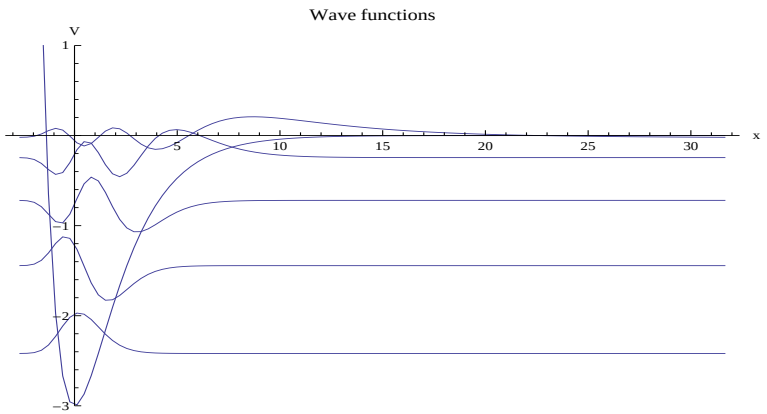

Fig. 1. Wave functions

4.2 The ITL Formal Specification using Ana Tempura

We made a formal specification of the code for calculating the bound states of the Morse oscillator well by writing a Tempura code and making tests to compare the output results from the .exe version of the C program and Tempura code. In order to establish communication between the tempura file and the .exe version of the C program assertions must be added in the C code. We will explain the C and Tempura code where the checking of the values in the **ke_fb[]** array is performed. The code for checking the other arrays and values is similar.

```
int i=0;
double ke_fb_i=0.00000;
assertion1 ("ke_fb_i", ke_fb_i);
assertion ("i", i);
while (i<NPTS) {
 ke_fb_i=square((i+1)*M_PI/del); assertion1 ("ke_fb_i", ke_fb_i);
     i=i+1;assertion ("i", i);
     }
```

The assertion functions are used for checking the values of the variables after each performed change. The values of the variables are compared to the values obtained from the Tempura code.

Here is the function for calculating the Y-th value of the **ke_fb[]** array written in Tempura language. The function **itof** returns the float corresponding to an integer number and $del = xmax - xmin$.

```
define calc_ke_fb(Y) = {
 if Y>=0 then {((itof(Y)+$1.0$)*M_PI/del)*((itof(Y)+$1.0$)*M_PI/del)
}
else empty
}.
```

Here is the test we defined to check the results:

```
/* run */ define test1() = {
```

```
 exists  ke_fb_i, Y :
{
check($0.00000$,ke_fb_i,Y);
while (Y<NPTS) do
{
check(calc_ke_fb(Y),ke_fb_i,Y)
}
}
}.
```

The function **check** compare the results from the **calc_ke_fb** function and the C program. If they are identical, test passed. The results from the first two iterations when a test is run by the Ana Tempura tool are shown in 2. These test cases pass successfully.

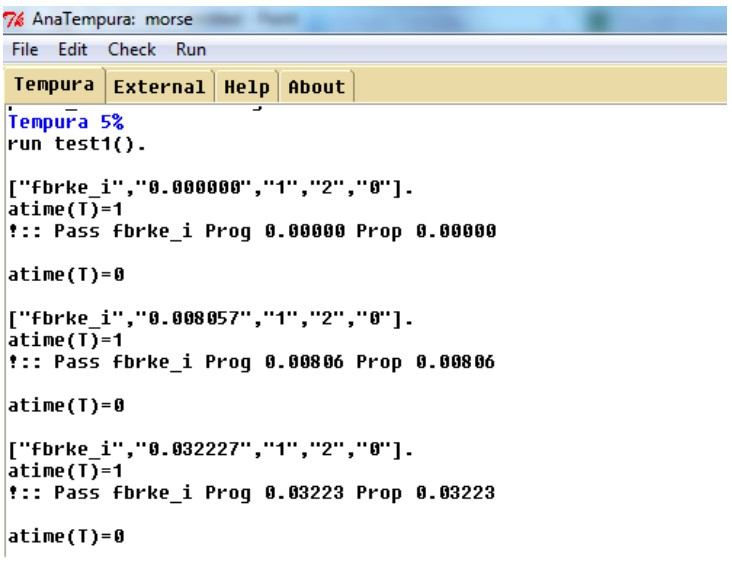

Fig. 2. Execution of test1 in Ana Tempura

5. CONCLUSION AND FUTURE WORK

In this paper we presented the importance of using formal methods for testing and specification of scientific software. A formal specification of the code for solving the bound states of the Morse oscillator well by using ITL and Ana Tempura was described. Also the implementation details and steps were given. Formal specification and description of the scientific software will improve its correctness by reducing the number of errors, especially when a code change is made. The testing also will be more accurate and mathematical model will be provided for the software. We chose ITL and Tempura as a powerful description language and extension to standard predicate logic which includes time-dependent operators. The programs written in Tempura and executed in Ana Tempura tool can communicate with .exe version of the programs. At this moment, there are examples of program codes

written in C and Java. There are no limitations of the number of test cases that can be covered. Tempura language is similar to predicate logic and it is easy to be learned. There are only several rules and categories such as expressions, statements and locations. The hard part for the scientists could be the programming with recursion.

We plan to make a formal specification of the scientific application which is developed within the HP-SEE (High-Performance Computing Infrastructure for South East Europe) project. We want to automate the testing process by creating and running tests using the Ana Tempura tool.

REFERENCES

E.R. Bittner. 2009. *Quantum dynamics: applications in biological and materials systems*. CRC Press. http://books.google.mk/books?id=wKrvAAAAMAAJ

Antonio Cau, Ben Moszkowski, and Hussein Zedan. 2002. Interval Temporal Logic. (2002). http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.4.6669

Edmund M. Clarke and Jeannette M. Wing. 1996. Formal Methods: State of the Art and Future Directions. *ACM Comput. Surv.* 28, 4 (Dec. 1996), 626–643. DOI:http://dx.doi.org/10.1145/242223.242257

De Montfort University. 2004. AnaTempura tool for runtime verification and animation. (2004). http://www.tech.dmu.ac.uk/STRL/research/software/anatempura.pdf

Amin El-kustaban, Ben Moszkowski, and Antonio Cau. 2012. Specification Analysis of Transactional Memory using ITL and AnaTempura. *Lecture Notes in Engineering and Computer Science* 2195, 1 (2012), 176–181.

J. F. Groote, A. A. H. Osaiweran, and J. Wesselius. *Benefits of applying formal methods to industrial control software*. Technical Report CS-Report 11–04. http://www.win.tue.nl/~jfg/articles/CSR-11-04.pdf

Anthony Hall. 2005. Realising the Benefits of Formal Methods. In *Formal Methods and Software Engineering*, Kung-Kiu Lau and Richard Banach (Eds.). Lecture Notes in Computer Science, Vol. 3785. Springer Berlin Heidelberg, 1–4. DOI:http://dx.doi.org/10.1007/11576280_1

I. P. Hamilon J. C. Light and J. J. V. Lill. 1985. DVR. *Chem. Phys.* 82, 1400 (1985).

E. Jaeger. 2010. *Study of the Benefits of Using Deductive Formal Methods for Secure Developments*. http://books.google.mk/books?id=EU2KMwEACAAJ

Bojana Koteska and Anastas Mishev. 2013. Software Engineering Practices and Principles to Increase Quality of Scientific Applications. In *ICT Innovations 2012*, Smile Markovski and Marjan Gusev (Eds.). Advances in Intelligent Systems and Computing, Vol. 207. Springer Berlin Heidelberg, 245–254. DOI:http://dx.doi.org/10.1007/978-3-642-37169-1_24

Ben Moszkowski. 1985. Executing temporal logic programs. In *Seminar on Concurrency*, StephenD. Brookes, AndrewWilliam Roscoe, and Glynn Winskel (Eds.). Lecture Notes in Computer Science, Vol. 197. Springer Berlin Heidelberg, 111–130. DOI:http://dx.doi.org/10.1007/3-540-15670-4_6

Ben Moszkowski and Zohar Manna. 1984. Reasoning in interval temporal logic. In *Logics of Programs*, Edmund Clarke and Dexter Kozen (Eds.). Lecture Notes in Computer Science, Vol. 164. Springer Berlin Heidelberg, 371–382. DOI:http://dx.doi.org/10.1007/3-540-12896-4_374

Jonathan S. Ostroff and Richard F. Paige. 1998. Formal Methods in the Classroom: The Logic of Real-Time Software Design. In *Proceedings of the Third IEEE Real-Time Systems Education Workshop (RTEW '98)*. IEEE Computer Society, Washington, DC, USA, 63–. http://dl.acm.org/citation.cfm?id=554225.828878

Carlos Rossi, Manuel Enciso, and Inmaculada P. de Guzmn. 2004. Formalization of UML state machines using temporal logic. *Software and Systems Modeling* 3, 1 (2004), 31–54. DOI:http://dx.doi.org/10.1007/s10270-003-0029-7

Rebecca Sanders and Diane Kelly. 2008. The Challenge of Testing Scientific Software. In *Proceedings of the Conference for the Association for Software Testing*. 30–36.

Guido Sciavicco, JoseM. Juarez, and Manuel Campos. 2009. Quality Checking of Medical Guidelines Using Interval Temporal Logics: A Case-Study. In *Bioinspired Applications in Artificial and Natural Computation*, Jos Mira, JosManuel Ferrndez, JosR. lvarez, Flix Paz, and F.Javier Toledo (Eds.). Lecture Notes in Computer Science, Vol. 5602. Springer Berlin Heidelberg, 158–167. DOI:http://dx.doi.org/10.1007/978-3-642-02267-8_18

J. Segal. 2008. Scientists and Software Engineers: A Tale of Two Cultures. In *Proceedings of the Psychology of Programming Interest Group*. http://www.ppig.org/papers/20th-segal.pdf

StephenF. Siegel and GeorgeS. Avrunin. 2004. Verification of MPI-Based Software for Scientific Computation. In *Model Checking Software*, Susanne Graf and Laurent Mounier (Eds.). Lecture Notes in Computer Science, Vol. 2989. Springer Berlin Heidelberg, 286–303. DOI:http://dx.doi.org/10.1007/978-3-540-24732-6_20

StephenF. Siegel and LouisF. Rossi. 2008. Analyzing BlobFlow: A Case Study Using Model Checking to Verify Parallel Scientific Software. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Alexey Lastovetsky, Tahar Kechadi, and Jack Dongarra (Eds.). Lecture Notes in Computer Science, Vol. 5205. Springer Berlin Heidelberg, 274–282. DOI:http://dx.doi.org/10.1007/978-3-540-87475-1_37

I. Sommerville. 2007. *Software Engineering*. Addison-Wesley. http://books.google.mk/books?id=B7idKfL0H64C

M. Umarji, C. Seaman, A.G. Koru, and Hongfang Liu. 2009. Software Engineering Education for Bioinformatics. In *Software Engineering Education and Training, 2009. CSEET '09. 22nd Conference on*. 216–223. DOI:http://dx.doi.org/10.1109/CSEET.2009.44

John A. van der Poll. 2010. Formal methods in software development: a road less travelled. *South African Computer Journal* 45 (2010), 40–52. http://dblp.uni-trier.de/db/journals/saj/saj45.html#Poll10

C. Vecchiola, S. Pandey, and R. Buyya. 2009. High-Performance Cloud Computing: A View of Scientific Applications. In *Proceedings of the 10th International Symposium on Pervasive Systems, Algorithms and Networks (I-SPAN 2009)*. IEEE Computer Society.

Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. 2009. Formal Methods: Practice and Experience. *ACM Comput. Surv.* 41, 4, Article 19 (Oct. 2009), 36 pages. DOI:http://dx.doi.org/10.1145/1592434.1592436

Hussein Zedan, Antonio Cau, Zhiqiang Chen, and Hongji Yang. 1999. ATOM: An object-based formal method for real-time systems. *Annals of Software Engineering* 7, 1-4 (1999), 235–256. DOI:http://dx.doi.org/10.1023/A:1018942406449

Inc. Ziff Davis Publishing Holdings. 1995. PC Magazine. (1995). http://www.pcmag.com/encyclopedia/term/50872/scientific-application