Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Национальный исследовательский университет «Высшая школа экономики»

Факультет Бизнес-информатики
Отделение Программной инженерии

Утверждаю
Заведующий отделением
Программной инженерии
С.М. Авдошин

_____
«___» _____ 201_ г

Программа дисциплины «Формальные методы программной инженерии»

(на английском языке)

для направления 231000.68 - Программная инженерия
подготовки магистра

Автор программы
Профессор, д.ф.-м.н. И.А.Ломазова
ilomazova@hse.ru

Одобрена на заседании кафедры управления разработкой программного обеспечения
«___»_____ 2012 г
Зав. кафедрой С.М. Авдошин

Рекомендована секцией УМС факультета бизнес информатики
«___»_____ 2012 г
Председатель Ю.В. Таратухина

Москва, 2012

I *Introductory Note*

Program Author:

Professor, Dr. Irina A. Lomazova, Dr. Sci. (Comp.Sci.)

General Description of the Curriculum:

The course is delivered to master students of software engineering department, business informatics faculty, The State University - Higher School of Economics/HSE (master program "Software engineering").
It is a part of general scientific curricula unit, and it is delivered in modules 1-4 of the first academic year. The course length is ***144*** academic hours of audience classes divided into ***72*** lecture hours and ***72*** seminar hours and ***216*** academic hours for students self-study.
The covered number of credits is ***10***. Academic control forms are one home assignment, one test, one written exam after module 2, and one written exam after module 4.

Pre-requisites

The course is based on the knowledge of foundations of discrete mathematics, computer science, mathematical logic, algorithm theory and computer programming.

Course Objective

The objective of the Formal methods in software engineering course delivery is to train students to treat the specification of software as a very important stage of software development, and also to appreciate the advantages and problems associated with this approach for future projects.

One of the important aspects of formal methods is that, even for quite simple problems, they force the students to think very carefully about the specification, and not to get involved in the coding too quickly. Even for students who have done a lot of programming before the ideas behind formal methods are likely to be completely new, and can draw their attention to problems of program correctness and reliability.

Another very important reason for teaching formal methods is that they are gradually being used in more industrial projects, and thus students should be familiar with at least the ideas associated with the approach, even if they have not learnt the specific formal specification language that their particular industry may require.

Abstract

In computer science and software engineering, formal methods are a particular kind of mathematically-based techniques for the specification, development and verification of software and hardware systems. The use of formal methods for software and hardware design is motivated by the fact that, as in other engineering disciplines, performing appropriate mathematical analysis can contribute to the reliability and robustness of a design.

Formal methods are best described as the application of a fairly broad variety of theoretical computer science fundamentals, in particular logic calculi, formal languages, automata theory, and program semantics, but also type systems and algebraic data types to problems in software and hardware specification and verification.

Formal methods can:

- Be a foundation for describing complex systems.
- Be a foundation for reasoning about systems.
- Provide support for program development.

In contrast to other design systems, formal methods use mathematical proof as a complement to system testing in order to ensure correct behavior. As systems become more complicated, and safety becomes a more important issue, the formal approach to system design offers another level of insurance.

Formal methods differ from other design systems through the use of formal verification schemes, the basic principles of the system must be proven correct before they are accepted. Traditional system design has used extensive testing to verify behavior, but testing is capable of only finite conclusions. Dijkstra and others have demonstrated that tests can only show the situations where a system won't fail, but cannot say anything about the behavior of the system outside of the testing scenarios. In contrast, once a theorem is proven true it remains true.

It is very important to note that formal verification does not obviate the need for testing. Formal verification cannot fix bad assumptions in the design, but it can help identify errors in reasoning which would otherwise be left unverified. In several cases, engineers have reported finding flaws in systems once they reviewed their designs formally.

Roughly speaking, formal design can be seen as a three step process, following the outline given here:

1. Formal Specification: During the formal specification phase, the engineer rigorously defines a system using a modeling language. Modeling languages are fixed formalisms which allow users to model complex structures out of predefined types. This process of formal specification is similar to the process of converting a word problem into algebraic notation.

   In many ways, this step of the formal design process is similar to the formal software engineering technique developed by Rumbaugh, Booch and others. At the minimum, both techniques help engineers to clearly define their problems, goals and solutions. However, formal modeling languages are more rigorously defined. And the clarity that this stage produces is a benefit in itself.

2. Verification: As stated above, formal methods differ from other specification systems by their heavy emphasis on provability and correctness. By building a system using a formal specification, the designer is actually developing a set of theorems about his/her system.

   Verification is a difficult process, largely because even the simplest system has several dozen theorems, each of which has to be proven. Even a traditional mathematical proof is a complex affair. Given the demands of complexity, almost all formal systems use an automated theorem proving tool of some form. These tools can prove simple theorems, verify the semantics of theorems, and provide assistance for verifying more complicated proofs.

3. Implementation: Once the model has been specified and verified, it is implemented by converting the specification into code. As the difference between software and hardware design grows narrower, formal methods for developing embedded systems have been developed. LARCH, for example, has a VHDL implementation. Similarly, hardware systems such as the VIPER and AAMP5 processors have been developed using formal approaches.

Formal methods offer additional benefits outside of provability, and these benefits do deserve some mention.

- Discipline: By virtue of their rigor, formal systems require an engineer to think out his design in a more thorough fashion. In particular, a formal proof of correctness is going to require a rigorous specification of goals, not just operation. This thorough approach can help identify faulty reasoning far earlier than in traditional design.

The discipline involved in formal specification has proved useful even on already existing systems. Engineers using the PVS system, for example, reported identifying several microcode errors in one of their microprocessor designs.

- Precision: Traditionally, disciplines have moved into jargons and formal notation as the weaknesses of natural language descriptions become more glaringly obvious. There is no reason that systems engineering should differ, and there are several formal methods which are used almost exclusively for notation.

For engineers designing safety-critical systems, the benefits of formal methods lie in their clarity. Unlike many other design approaches, the formal verification requires very clearly defined goals and approaches. In a safety critical system, ambiguity can be extremely dangerous, and one of the primary benefits of the formal approach is the elimination of ambiguity.

The purpose of this course is to learn how to specify behavior of systems and to experience the design of a system where you can prove that the behavior is correct. Students will learn how to formally specify requirements and to prove (or disprove) them on the behavior. The behavior of systems will be represented by such formalisms as

- finite state machines;
- process algebras;
- Petri nets;
- temporal logics.

With a practical assignment students will experience how to apply the techniques in practice.

The first part of this course focuses on the study of the semantics of a variety of programming language constructs. We will study structural operational semantics as a way to formalize the intended execution and implementation of languages, axiomatic semantics, useful in developing as well as verifying programs, and denotational semantics, whose deep mathematical underpinnings make it the most versatile of all.

Then the special emphasis will be put on parallel and distributed systems modeling, specification and analysis. We consider two basic approaches to concurrent systems specification and analysis: process algebras and Petri nets.

Process algebra is a mathematical framework in which system behavior is expressed in the form of algebraic terms, enhancing the available techniques for manipulation. Fundamental to process algebra is a parallel operator, to break down systems into their concurrent components. A set of equations is imposed to derive whether two terms are behaviorally equivalent. In this framework, non-trivial properties of systems can be established in an elegant fashion. For example, it may be possible to equate an implementation to the specification of its required input/output relation. In recent years a variety of automated tools have been developed to facilitate the derivation of such properties.

Applications of process algebra exist in diverse fields such as safety critical systems, network protocols, and biology. In the educational vein, process algebra has been recognized to teach skills to deal with complex concurrent systems, by representing and reasoning about such systems in a mathematically clear and precise manner.

Petri nets is another popular formalism for modeling, analyzing and verifying reactive and distributed systems. Their strength are their simple but precise semantics, their clear graphical notation, and many methods and algorithms for analysis and verification.

The course introduces Petri nets and their theory by the help of examples from different application domains. The focus, however, will be on traditional Petri net theory, in particular on Place/Transition-Systems and on concepts such as place and transition invariants, deadlocks and traps, and the coverability tree. The course also covers different versions and variants of Petri nets as well as different modeling and analysis techniques for particular application areas. Thus

we consider an urgent topic of modeling and analysis of workflow processes in more details.

The forth module covers a prominent verification technique that has emerged in the last thirty years - model checking. This approach is based on systematical check whether a model of a given system satisfies a property such as deadlock freedom, invariants, or request-response. This automated technique for verification and debugging has developed into a mature and widely-used industrial approach with many applications in software and hardware. It is used (and further developed) by companies and institutes such as IBM, Intel, NASA, Cadence, Microsoft, and Siemens, to mention a few, and has culminated in a series of mostly freely downloadable software tools that allow the automated verification of, for instance, C#-programs or combinational hardware circuits.

Subtle errors, for instance due to multi-threading, that remain undiscovered using simulation or peer reviewing can potentially be revealed using model checking. Model checking is thus an effective technique to expose potential design errors and improve software and hardware reliability.

This course provides an introduction to the theory of model checking and its theoretical complexity. We introduce transition systems, safety, liveness and fairness properties, as well as omega-regular automata. We then cover the temporal logics LTL, CTL and CTL*, compare them, and treat their model-checking algorithms. Techniques to combat the state- space explosion problem are at the heart of the success of model checking.

We will show that model checking is based on well-known paradigms from automata theory, graph algorithms, logic, and data structures. Its complexity is analyzed using standard techniques from complexity theory.

Training Objectives:

During the course, the students will:

- Study the basic principles of using formal methods for specification and analysis of software systems;
- Study basic notions and modes of formal semantics for sequential and concurrent programs.
- Study formalism, such as process algebras and Petri nets, and methods for modeling and analysis of concurrent and distributed systems.
- Study methods and algorithms for model checking of concurrent systems;
- Master methods and tools of software specification, analysis and verification;
- Acquire practical skills in design, specification and analysis of model distributed systems examples.

At the end of the course, students should be able to: understand the language of studied formalisms; model various classes of systems using these formalisms; apply specific analytical techniques; prove properties of discrete systems using process algebras, Petri nets and appropriate specification formalisms.

II. . Topic-Wise^Curricula Plan

No	Topic Name	Course Hours, Total	Audience Hours		Self-Study
			Lectures	Practical Studies	
Module 1 (80 hrs.)					
1	Formal methods as a basis for software reliability.	10	2	2	6
2.	Finite state machines (FSMs): basic definitions, operational semantics. Categories of FSMs. Extended FSMs. Modeling concurrent systems with communicating FSMs.	20	4	4	12
3.	Petri nets: basic notions, definitions and classification. Modeling distributed systems with Petri nets.	30	6	6	18
4.	Petri nets analysis. Checking structural and behavioral properties.	20	4	4	12
	Module 1, totally:	80	16	16	48
Module 2 (80 hrs.)					
5.	High-level Petri nets. Colored Petri nets and CPNTools.	20	4	4	12
6.	Workflow modeling and verification based on Petri nets formalism	10	2	2	6
7.	Modeling distributed and concurrent system with process algebras. Algebra CCS: syntax, semantics, modeling technique.	30	6	6	18
8.	The notion and properties of bisimilarity relation. Verifying reactive concurrent systems with CCS.	20	4	4	12
	Module 2, totally:	80	16	16	48
Module 3 (100 hrs.)					
9.	Elements of predicate logic and theory of computation.	20	4	4	12
10.	Temporal logics LTL and CTL for specification of behavioral properties of reactive systems.	20	4	4	12
11.	Model checking algorithm for verification of CTL formulae.	30	6	6	18
12.	Automata-based approach for verification of LTL formulae.	30	6	6	18
	Module 3, totally:	100	20	20	60

Module 4 (100 hrs.)

13.	Specifying distributed systems with Promela. Spin model checker.	40	8	8	24
14.	Semantics of sequential programs. Operational and denotational semantics.	30	6	6	16
15.	Floyd method for verification of sequential programs. Hoare axiomatic semantics for sequential and parallel programs.	30	6	6	16
	Module 4, totally:	100	20	20	60
	TOTAL:	360	72	72	216

III. .Basic_book(S)).and/or reader(s)

Books:

1. Карпов Ю.Г. MODEL CHECKING. Верификация параллельных и распределенных программ и систем. - СПб.: БХВ-Петербург, 2010. - 560 с.
2. Ломазова И.А. Сети Петри и анализ поведенческих свойств распределенных систем. - Ярославль: ЯрГУ, 2002. 164 с.
3. Миронов А.М. Теория процессов. М.: МГУ. Доступна на http://intsvs.msu.ru/staff/mironov/processes.pdf.
4. Ben-Ari M. Principles of the Spin Model Checker. - Springer-Verlag, 2008. - 216 p.
5. Jensen K. and Kristensen L. M. Coloured Petri Nets Modelling and Validation of Concurrent Systems, Springer-Verlag, 2009.
6. Nielson H. R. and Nielson F. Semantics with Applications: An Appetizer. Springer-Verlag, 2007-274 p.
7. Schneider K. Verification of Reactive Systems. - Springer-Verlag, 2004. - 216 p.
8. C. Girault, R. Valk. Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications. Springer-Verlag, 2002.
9. D. Peled: Software Reliability Methods, Springer-Verlag 2001.
10. Грис Д. Наука программирования. - М.: Мир, 1984. - 416 с.
11. Michael R. A. Huth, Mark D. Ryan. Logic in Computer Science - modelling and reasoning about systems. - Cambridge University Press, 2004, 427 pages.
12. Singh A. Elements of Computation Theory. Springer-Verlag, 2009. - 422 p.
13. Хопкрофт Дж., Мотвани Р., Ульман Дж. Введение в теорию автоматов, языков и вычислений: Пер. с англ. - М.: Издательский дом "Вильямс", 2008. 528 с.
14. Glynn Winskel, "The Formal Semantics of Programming Languages: An Introduction", MIT Pres, 1993.
15. R.A. Milner. Calculus of communicating systems. Lecture Notes in Computer Science, v.92, Springer, 1980.
16. Fokkink W. Modelling distributed systems (Texts in Theoretical Computer Science. An EATCS Series), Springer-Verlag New York, Inc., Secaucus, NJ, 2007. 156 pp.
17. Roscoe, A. W. The Theory and Practice of Concurrency. Prentice Hall, 1997. - 605 p. http://web.comlab.ox.ac.uk/oucl/work/bill.roscoe/publications/68b.pdf

18. Glenn Brunes. Dystributed system analysis with CCS. Prentice HallEurope, 1997. - 168 p.
19. C. Girault, R. Valk. Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications. Springer-Verlag, 2002.
20. ван дер Аалст В., ван Хей К. Управление потоками работ: модели, методы и системы. - М.: Физматлит, 2007. - 316 с.

Internet References:

1. Formal Methods Wiki http://formalmethods.wikia.com/wiki/Formal Methods Wiki
2. Formal Methods Education Resources http://www.cs.indiana.edu/formal-methods-education/
3. Marcelo Fiore. Course materials *Denotational Semantics* (University of Cambridge). http://www.cl.cam.ac.uk/teaching/0910/DenotSem/
4. Wolfgang Schreiner. Course materials *Formal Semantics of Programming Languages* (RICS) http://moodle.risc.uni-linz.ac.at/course/view.php?id=30
5. Matthew Parkinson. Course materials *Software Verification* (University of Cambridge). http://www.cl.cam.ac.uk/teaching/0910/L19/
6. Rajeev Alur, Tom Henzinger. Invariant verification. Chapter II in manuscript "Computer-aided verification". http://mtc.epfl.ch/courses/CAV2006/Notes/2.pdf
7. The Petri Nets World http://www.informatik.uni-hamburg.de/TGI/PetriNets/
8. Internet resource: Workflow management coalition http://www.wfmc.org/
9. Internet resource: Workflow And Reengineering International Association http://www.waria.com/
10. Carl Adam Petri and Wolfgang Reisig. Petri net. *Scholarpedia*, 3(4):6477 (2008). http://www.scholarpedia.org/article/Petri net

IV. Education .controlforms/Assessments:

- Current control: attendance record, seminar-based knowledge control, home assignment control;
- Intermediate control: written test by the end of Module 1, exam by the end of Module 2, home assignment by the end of Module 3;
- Final control: exam by the end of Module 4;
- The final course grade is formed from the following elements:

1) practice activities (reports, discussions, business cases);
2) home assignment;
3) written test;
4) exam at the end of Module 2;
5) exam at the end of Module 4.

The credit grade C_2 (10-point scale) at the end of the Module 2 is calculated according to the following formulas:

$$CG_2 = 0{,}3\ P_2 + 0{,}7\ T,$$
$$C_2 = 0{,}4\ CG_2 + 0{,}6\ E_2,$$

where CG_2 is the cumulative grade at the end of Module 2, P_2 - practice activity in the course of Modules 1 and 2, T is the written test grade, E_2 - exam at the end of Module 2 (all in 10-point scale).

The overall course grade G (10-point scale) is calculated as follos:

$$CG_4 = 0{,}25\ P_4 + 0{,}5\ H + 0{,}25\ CG_2,$$

$$G = 0{,}4\ CG_4 + 0{,}6\ E_4,$$

where CG_4 is the cumulative grade at the end of Module 4, P_4 - practice activity in the course of Modules 3 and 4, H is the homework grade, E_4 - exam at the end of Module 4 (all in 10-point scale).

The overall course grade G (10-point scale) is rounded up to an integer number of points.

Summary Table : Correspondence of ten-point to five-point system's marks

Ten-point scale [10]	Five-point scale [5]
1 - unsatisfactory 2 - very bad 3 - bad	Unsatisfactory - 2
4 - satisfactory 5 - quite satisfactory	Satisfactory - 3
6 - good 7 - very good	Good - 4
8 - nearly excellent 9 - excellent 10 - brilliant	Excellent - 5

V *Program_ Contents*

Topic 1: Formal methods as a basis for software reliability.

- *Topic outline:*
  - Why formal methods.
  - Formal methods and software/hardware reliability.
  - Formal methods: historical overview.
  - How logic helps computer scientists.
  - Formal methods vs. simulation and testing.
  - Course overview.
- *Main references/books/reading:*
  1. D. Peled: Software Reliability Methods, Springer-Verlag 2001.
  2. Карпов Ю.Г. MODEL CHECKING. Верификация параллельных и распределенных программ и систем. - СПб.: БХВ-Петербург, 2010. - 560 с.
- *Additional references/books/reading:*
  1. Грис Д. Наука программирования. - М.: Мир, 1984. - 416 с.
  2. Formal methods. In: Wikipedia, http://en.wikipedia.org/wiki/Formal methods
  3. Michael R. A. Huth, Mark D. Ryan. Logic in Computer Science - modelling and reasoning about systems. - Cambridge University Press, 2004, 427 pages.
  4. J. Rutten, M. Kwiatkowska, G. Norman and D. Parker: Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems, Volume 23 of CRM Monograph Series. American Mathematical Society, P. Panangaden and F. van Breugel (eds.), March 2004.

5. Jonathan P., Bowen and Mike Hinchey “Ten Commandments of Formal Methods ... Ten Years Later”, , IEEE Computer, 39(1):40-48, January 2006.

Topic 2. Finite state machines (FSMs): basic definitions, operational semantics. Categories of FSMs. Extended FSMs. Modeling concurrent systems with communicating FSMs.

- *Topic outline:*
  - Finite state machines (FSMs): informal introduction, formal definitions, case study.
  - State transition diagrams.
  - Deterministic and nondeterministic FSMs.
  - Extended FSMs.
  - Communicating mechanisms for concurrent systems. Specifying distributed systems with interacting automata.
  - Proving protocol correctness with communicating FSMs.
- *Main references/books/reading:*
  1. Хопкрофт Дж., Мотвани Р., Ульман Дж. Введение в теорию автоматов, языков и вычислений: Пер. с англ. - М.: Издательский дом "Вильямс", 2008. 528 с.
  2. Карпов Ю.Г. Теория автоматов. - СПб., Питер, 2003. - 208 с.
- *Additional references/books/reading:*
  1. Yuri Gurevich, *Sequential Abstract State Machines Capture Sequential Algorithms,* ACM Transactions on Computational Logic, vl. 1, no. 1 (July 2000), pages 77-111. http://research.microsoft.com/~gurevich/Opera/141.pdf
  2. Дехтярь М.И. Лекции по дискретной математике. / М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007.
  3. Boerger E., Staerk R. Abstract state machines. A method for high-level system design and analysis. - Springer, 2003. 448 p.
  4. Wagner, F., "Modeling Software with Finite State Machines: A Practical Approach", Auerbach Publications. - CRC Press, 2006. 302 p.

Topic 3. Petri nets: basic notions, definitions and classification. Modeling distributed systems with Petri nets.

- *Topic outline:*
  - Motivation and informal introduction. Net formalisms for modeling distributed systems. Examples from different areas.
  - Place/Transition systems: basic concepts. Places, transition, linear algebraic representation.
  - Firing rule, interleaving semantics, occurrence graph, unboundedness.
  - Variants of Petri nets: condition/event systems, contact-free nets, high-level Petri nets, colored Petri nets, nested Petri nets.
  - Modeling basic control constructs with Petri nets: sequencing, nondeterministic choice, concurrency.
  - Modeling causality relations and resource dependencies with Petri nets.

- *Main references/books/reading:*

  1. C. Girault, R. Valk. Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications. Springer-Verlag, 2002.
  2. Ломазова И.А. Сети Петри и анализ поведенческих свойств распределенных систем. - Ярославль: ЯрГУ, 2002. 164 с.
  3. Carl Adam Petri and Wolfgang Reisig. Petri net. *Scholarpedia*, 3(4):6477 (2008). http://www.scholarpedia.org/article/Petri net

- *Additional references/books/reading:*

  1. Jensen K. and Kristensen L. M. Coloured Petri Nets Modelling and Validation of Concurrent Systems, Springer-Verlag, 2009.
  2. Вирбицкайте И.Б. Сети Петри: модификации и расширения. Новосибирск: Изд-во НГУ, 2005, 123 с.
  3. В.Е.Котов. Сети Петри. М.: Наука, 1984.
  4. Ломазова И.А. Вложенные сети Петри: моделирование и анализ распределенных систем с объектной структурой. - М.: Научный мир, 2004. 208 с.
  5. Питерсон Дж. Теория сетей Петри и моделирование систем. М.: Мир, 1984.
  6. The Petri Nets World http://www.informatik.uni-hamburg.de/TGI/PetriNets/
  7. Wolfgang Reisig. Petrinetze. Modellierungstechnik, Analysemethoden, Fallstudien. Vieweg+Teubner, 2010.

Topic 4. Petri nets analysis. Checking structural and behavioral properties.

- *Topic outline:*

  - Interleaving and concurrent semantics for Petri nets. Sequential and concurrent runs.
  - Coverability tree.
  - Propositional state properties of P/T nets: incidence matrix, state equation, place invariants.
  - Positive place invariants and boundedness; transition invariants and deadlocks; siphons and traps.
  - Analysis of behavioral problems for Petri Nets: Safeness; Boundedness; Conservation; Liveness; Reachability and coverability.
  - Analysis techniques for State Machines, Marked Graphs, Extended Free Choice Nets.

- *Main references/books/reading:*

  1. C. Girault, R. Valk. Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications. Springer-Verlag, 2002.
  2. Ломазова И.А. Сети Петри и анализ поведенческих свойств распределенных систем. - Ярославль: ЯрГУ, 2002. 164 с.
  3. Jorg Desel, Wolfgang Reisig, Grzegorz Rozenberg (Eds.)Lectures on Concurrency and Petri Nets, Advances in Petri Nets, Lecture Notes in Computer Science, vol. 3098, Springer-Verlag, 2004.

- *Additional references/books/reading:*

  1. Jensen K. and Kristensen L. M. Coloured Petri Nets Modelling and Validation of Concurrent Systems, Springer-Verlag, 2009.
  2. Ломазова И.А. Вложенные сети Петри: моделирование и анализ распределенных

систем с объектной структурой. - М.: Научный мир, 2004. 208 с.

3. Вирбицкайте И.Б. Сети Петри: модификации и расширения. Новосибирск: Изд-во НГУ, 2005, 123 с.
4. В.Е.Котов. Сети Петри. М.: Наука, 1984.
5. Питерсон Дж. Теория сетей Петри и моделирование систем. М.: Мир, 1984.
6. The Petri Nets World http://www.informatik.uni-hamburg.de/TGI/PetriNets/
7. Wolfgang Reisig. Petrinetze. Modellierungstechnik, Analysemethoden, Fallstudien. Vieweg+Teubner, 2010.

Topic 5. High-level Petri nets. Colored Petri nets and CPNTools.

- *Topic outline:*
  - Expressibility of Petri nets. Extending Petri nets with reset and inhibitor arcs.
  - Introducing colored tokens and types.
  - Hierarchical modeling.
  - Modeling multi-agent systems with nested Petri nets.
  - Modeling case studies: producer/consumer system, sequential and parallel buffers, crosstalk algorithm, mutual exclusion, dining philosophers.
- *Main references/books/reading:*
  1. C. Girault, R. Valk. Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications. Springer-Verlag, 2002.
  2. Jensen K. and Kristensen L. M. Coloured Petri Nets Modelling and Validation of Concurrent Systems, Springer-Verlag, 2009.
  3. Reisig, Wolfgang. Elements of distributed algorithms :modeling and analysis with Petri Nets. Berlin : Springer, 1998.
  4. Ломазова И.А. Сети Петри и анализ поведенческих свойств распределенных систем. - Ярославль: ЯрГУ, 2002. 164 с.
- *Additional references/books/reading:*
  1. Вирбицкайте И.Б. Сети Петри: модификации и расширения. Новосибирск: Изд-во НГУ, 2005, 123 с.
  2. В.Е.Котов. Сети Петри. М.: Наука, 1984.
  3. Ломазова И.А. Вложенные сети Петри: моделирование и анализ распределенных систем с объектной структурой. - М.: Научный мир, 2004. 208 с.
  4. Питерсон Дж. Теория сетей Петри и моделирование систем. М.: Мир, 1984.
  5. The Petri Nets World http://www.informatik.uni-hamburg.de/TGI/PetriNets/
  6. Wolfgang Reisig. Petrinetze. Modellierungstechnik, Analysemethoden, Fallstudien. Vieweg+Teubner, 2010.

Topic 6. Workflow modeling and verification based on Petri nets formalism.

- *Topic outline:*
  - Workflow concepts: the case, the task, the process, routing, enactment.
  - Mapping workflow concepts onto Petri nets. Case studies.

- Workflow nets: definition and structural properties.
- Analysis technique for workflow nets: reachability analysis, structural analysis.
- Soundness (proper termination) for workflow nets.
- Well-structured workflow nets. Soundness and safeness for well-structured nets.
- Free-choice workflow nets and their properties.

♦ *Main references/books/reading:*

1. ван дер Аалст В., ван Хей К. Управление потоками работ: модели, методы и системы. - М.: Физматлит, 2007. - 316 с.

♦ *Additional references/books/reading:*

1. Internet resource: Workflow management coalition http://www.wfmc.org/
2. Internet resource: Workflow And Reengineering International Association http://www.waria.com/

Topic 7. Modeling distributed and concurrent system with process algebras. Algebra CCS: syntax, semantics, modeling technique.

♦ *Topic outline:*

- Reactive systems: main notions and examples.
- Flow diagrams of distributed systems. Ports and interactions.
- Interleaving semantics of concurrent systems. Labeled transition systems. Concurrency and nondeterminism.
- The Calculus of Communicating Systems (CCS) of R.Milner informally.
- Formal definition of CCS; semantics of CCS; transition diagrams; examples.
- CCS case studies.

♦ *Main references/books/reading:*

1. R.A. Milner. Calculus of communicating systems. Lecture Notes in Computer Science, v.92, Springer, 1980.
2. Fokkink Wan. Introduction to Process Algebra. - Springer-Verlag, 2007. - 169 p.
3. Roscoe, A. W. The Theory and Practice of Concurrency. Prentice Hall, 1997. - 605 p. http://web.comlab.ox.ac.uk/oucl/work/bill.roscoe/publications/68b.pdf

♦ *Additional references/books/reading:*

1. Fokkink W. Modelling distributed systems (Texts in Theoretical Computer Science. An EATCS Series), Springer-Verlag New York, Inc., Secaucus, NJ, 2007. 156 pp.
2. Хоар А.Ч. Взаимодействующие последовательные процессы. М.: Мир, 1989.
3. Glenn Brunes. Distributed system analysis with CCS. Prentice Hall Europe, 1997. - 168 p.
4. Миронов А.М. Теория процессов. М.: МГУ. http://intsys.msu.ru/staff/mironov/processes.pdf.
5. Glynn Winskel, Mogens Nielsen. Models for Concurrency. http://www.daimi.au.dk/PB/463/PB-463.pdf

Topic 8. The notion and properties of bisimilarity relation. Verifying reactive concurrent systems with CCS.

- *Topic outline:*
  - Trace equivalence; strong bisimilarity; bisimulation games; properties of strong bisimilarity.
  - Weak bisimilarity; weak bisimulation games; properties of weak bisimilarity; example (a tiny communication protocol).
  - Analysis of CCS behavior; syntax of Hennessy-Milner logic; semantics of Hennessy-Miler logic; examples.
  - Correspondence between strong bisimilarity and Hennessy-Milner logic.
  - Value passing CCS.
  - The language of Communicating Sequential Processes (CSP): brief overwiew.
- *Main references/books/reading:*

4. R.A. Milner. Calculus of communicating systems. Lecture Notes in Computer Science, v.92, Springer, 1980.
5. Glenn Brunes. Dystributed system analysis with CCS. Prentice HallEurope, 1997. - 168 p.
6. Миронов А.М. Теория процессов. М.: МГУ. http://intsys.msu.ru/staff/mironov/processes.pdf.

- *Additional references/books/reading:*

6. Fokkink Wan. Introduction to Process Algebra. - Springer-Verlag, 2007. - 169 p.
7. Fokkink W. Modelling distributed systems (Texts in Theoretical Computer Science. An EATCS Series), Springer-Verlag New York, Inc., Secaucus, NJ, 2007. 156 pp.
8. Хоар А.Ч. Взаимодействующие последовательные процессы. М.: Мир, 1989.
9. Roscoe, A. W. The Theory and Practice of Concurrency. Prentice Hall, 1997. - 605 p. http://web.comlab.ox.ac.uk/oucl/work/bill.roscoe/publications/68b.pdf

Topic 9. Elements of predicate logic and theory of computation.

- *Topic outline:*
  - The language of predicate logic: syntax and semantics.
  - Logical consequence and equivalence. Equivalent transformations for predicate logic formulae.
  - The natural deductive system for predicate logic: axioms and deductive rules. Soundness and completeness of natural deductive axiomatic.
  - Decidable and undecidable problems. Examples of decidable and undecidable problems. Deducibility problem for predicate logic. The notion of reducibility. Rice theorem.
  - Computational complexity. Decision problems as formal languages. Time complexity. Complexity classes. Reductions. NP-hard and NP-complete problems.
- *Main references/books/reading:*

1. Singh A. Elements of Computation Theory. Springer-Verlag, 2009. - 422 p.
2. Michael R. A. Huth, Mark D. Ryan. Logic in Computer Science - modelling and reasoning about systems. - Cambridge University Press, 2004, 427 pages.
3. Колмогоров А.Н., Драгалин А.Г. Математическая логика. - М.: КомКнига, 2006.

240 с.

- *Additional references/books/reading:*

1. Непейвода Н.Н. Прикладная логика. - Новосибирск: Изд-во Новосиб. Ун-та, 2000. - 521 с.
2. Хопкрофт Дж., Мотвани Р., Ульман Дж. Введение в теорию автоматов, языков и вычислений: Пер. с англ. - М.: Издательский дом "Вильямс", 2008. 528 с.
3. Булос Дж., Джеффри Р. Вычислимость и логика. М., Мир, 1994.
4. Дехтярь М.И. Лекции по дискретной математике. / М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007.
5. Грис Д. Наука программирования. - М.: Мир, 1984. - 416 с.

Topic 10. Temporal logics LTL and CTL.

- *Topic outline:*
  - Model and temporal logics: main consepts.
  - Linear Temporal Logic LTL: syntax, semantics, main properties and case studies.
  - Linear time properties: safety, liveness, decomposition.
  - Fairness: unconditional, strong and weak fairness.
  - Computational Tree Logic CTL: syntax, semantics, equational laws.
  - Comparing LTL and CTL.
- *Main references/books/reading:*

1. Карпов Ю.Г. MODEL CHECKING. Верификация параллельных и распределенных программ и систем. - СПб.: БХВ-Петербург, 2010. - 560 с.
2. Manna Z., Pnueli A. The temporal logic of reactive and concurrent systems. - Springer-Verlag, 1991. 427 p.

- *Additional references/books/reading:*

1. Schneider K. Verification of Reactive Systems. - Springer-Verlag, 2004. - 216 p.
2. Кларк Э.М., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. - М.: МЦНМО, 2002. - 416 с.
3. Кузьмин Е.В. Верификация моделей программ. - Ярославль: ЯрГУ, 2008. - 76 с.

Topic 11. Model checking algorithm for verification of CTL formulae.

- *Topic outline:*
  - Kripke structures.
  - Semantics of CTL on computational trees.
  - CTL model checking: recursive descent, backward reachability, complexity.
  - Fairness, counterexamples/witnesses.
  - CTL+ and CTL* .
  - Fair CTL semantics, model checking.
- *Main references/books/reading:*

1. Карпов Ю.Г. MODEL CHECKING. Верификация параллельных и распределенных программ и систем. - СПб.: БХВ-Петербург, 2010. - 560 с.
2. Schneider K. Verification of Reactive Systems. - Springer-Verlag, 2004. - 216 p.

♦ *Additional references/books/reading:*

1. Кларк Э.М., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. - М.: МЦНМО, 2002. - 416 с.
2. Кузьмин Е.В. Верификация моделей программ. - Ярославль: ЯрГУ, 2008. - 76 с.

Topic 12. Automata-based approach for verification of LTL formulae.

♦ *Topic outline:*

- Automata on finite words.
- Verifying regular safety properties. Product construction, counterexamples.
- Automata on infinite words. Generalized Buchi automata, ю-regular languages.
- Verifying ю-regular properties: nested depth first search.

♦ *Main references/books/reading:*

1. Карпов Ю.Г. MODEL CHECKING. Верификация параллельных и распределенных программ и систем. - СПб.: БХВ-Петербург, 2010. - 560 с.
2. Schneider K. Verification of Reactive Systems. - Springer-Verlag, 2004. - 216 p.

♦ *Additional references/books/reading:*

1. Кларк Э.М., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. - М.: МЦНМО, 2002. - 416 с.
2. Кузьмин Е.В. Верификация моделей программ. - Ярославль: ЯрГУ, 2008. - 76 с.

Topic 13. Specifying distributed systems with Promela. Spin model checker.

♦ *Topic outline:*

- Sequential Programming in PROMELA specification language: data types, operators and expressions, control statements.
- Verification of sequential programs, assertions, guided simulation.
- Interactive simulation of concurrent programs.
- Synchronization and nondeterminism in concurrent programs.
- Deadlock verification.
- Verification with temporal logic LTL.
- Expressing and verifying safety properties.
- Expressing and verifying liveness properties.
- Case studies.

♦ *Main references/books/reading:*

1. Ben-Ari M. Principles of the Spin Model Checker. - Springer-Verlag, 2008. - 216 p.

♦ *Additional references/books/reading:*

1. Карпов Ю.Г. MODEL CHECKING. Верификация параллельных и распределенных программ и систем. - СПб.: БХВ-Петербург, 2010. - 560 с.

2. Schneider K. Verification of Reactive Systems. - Springer-Verlag, 2004. - 216 p.
3. Кларк Э.М., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. - М.: МЦНМО, 2002. - 416 с.
4. Кузьмин Е.В. Верификация моделей программ. - Ярославль: ЯрГУ, 2008. - 76 с.

Topic 14. Semantics of sequential programs. Operational and denotational semantics.

- *Topic outline:*
  - Sequential programs as state transformers.
  - The imperative model language WHILE.
  - Operational semantics of WHILE. Reduction rules. Properties of operational semantics.
  - Denotational semantics of WHILE.
  - The least fixpoint operator properties.
  - Equivalence of operational and denotational semantics.
- *Main references/books/reading:*
  1. Nielson H. R. and Nielson F. Semantics with Applications: An Appetizer. Springer- Verlag, 2007-274 p.
  2. Glynn Winskel, "The Formal Semantics of Programming Languages: An Introduction", MIT Pres, 1993.
- *Additional references/books/reading:*
  1. Flemming Nielson, Hanne Riis Nielson, Chris Hankin Principles of program analysis Springer, 2005, 450 pp.
  2. Marcelo Fiore. Course materials *Denotational Semantics* (University of Cambridge). http://www.cl.cam.ac.uk/teaching/0910/DenotSem/
  3. Wolfgang Schreiner. Course materials *Formal Semantics of Programming Languages* (RICS) http://moodle.risc.uni-linz.ac.at/course/view.php?id=30

Topic 15. Floyd method for verification of sequential programs. Hoare axiomatic semantics for sequential and parallel programs.

- *Topic outline:*
  - Partial and total correctness assertions.
  - Floyd method for proving partial program correctness. The notion of invariant.
  - The axiomatic approach for proving program correctness
  - Hoare's assertion language: syntax and semantics.
  - Partial correctness properties.
  - Hoare's logic. Soundness and relative completeness of Hoare's logic.
  - Weakest preconditions and their properties.
  - Proving total program correctness. Soundness and relative completeness of total correctness.
  - Equivalence of axiomatic and denotational/operational semantics.
  - Hoare's logic for parallel programs. Semantics of parallel constructions. Rules for partial correctness assertions.

- *Main references/books/reading:*

1. Nielson H. R. and Nielson F. Semantics with Applications: An Appetizer. Springer- Verlag, 2007- 274 p.
2. Грис Д. Наука программирования. - М.: Мир, 1984. - 416 с.

- *Additional references/books/reading:*

1. Rajeev Alur, Tom Henzinger. Invariant verification. Chapter II in manuscript "Computer-aided verification". http://mtc.epfl.ch/courses/CAV2006/Notes/2.pdf
2. Matthew Parkinson. Course materials *Software Verification* (University of Cambridge). http://www.cl.cam.ac.uk/teaching/0910/L19/

VI._Assignment_topicsfor_ various education_ controjformsf

- Home assignment:

The home task deals with constructing a formal model and verifying it and can be done individually of in small (2-3 students) groups. Given a concrete distributed system (communication protocol, a system of interacting agents, resource producing/consuming system etc.) student should accomplish the following items:

- Develop a Petri net model of a given distributed system.
- Describe the main behavioral properties of the constructed model.
- Classify the behavioral properties and chose appropriate methods and/or tools for specifying and verifying these properties.
- Verify the behavior of the constructed system.

- Written test

The written test is a computer testing assessment based on the topics covered in the course.

VII,. Topics for course , resultsquality, assessment

♦ Exam

The final exam is based on the course topics:

- Decidability and complexity of formal languages.
- Operational, denotational and axiomatic semantics of sequential programs.
- The least fixpoint semantics of loop statement.
- Verification of sequential programs with partial and total correctness assertions.
- Interleaving semantics of concurrent programs.
- Labeled transition systems.
- Formal models of concurrent and distributed systems.
- Theory of process algebras.
- Branching time semantics of concurrent processes.
- Trace and bisimulation equivalence of concurrent programs. Strong and weak bisimulation.
- Hennessy-Milner logic for process algebra CCS.
- Process algebra CSP.
- Petri net theory.
- Interleaving and concurrent semantics for Petri nets.
- Structural properties of Petri nets.
- Classification of Petri nets.
- Expressibility of Petri nets.
- Proving Petri nets properties with reachability and coverability trees.
- Modeling workflow processes with
- Proving soundness for workflow nets.
- Structured workflow nets and their properties.
- Temporal logics for specification of concurrent systems behavior.
- Syntax, semantics and equational laws of Linear Temporal Logic LTL.
- Syntax, semantics and equational laws of Computational Tree Logic CTL.
- Comparing LTL and CTL expressibility.
- Automata on infinite words and ю-regular languages.
- Model checking of LTL and CTL formulae.

The author of the program: I.A. Lomazova