INTRODUCTION TO COMPUTER SCIENCE

General competences

Description of study/examination achievements

Type of examination: Written examination (120 minutes)

The examination is carried out in the form of a written examination lasting 120 minutes. Knowledge questions test familiarity with computer science and programming concepts, while small programming tasks test the ability to solve problems and implement distributed applications using customized algorithms.

Intended learning outcomes

After successfully completing this module, participants will understand the essential concepts of computer science at a basic, practice-oriented but scientific level and will be able to handle them responsibly.

Concepts of this type include: algorithms, syntax and semantics, as well as efficiency in terms of memory consumption or time.

Participants will then be able to solve manageable algorithmic problems in Java or a similar object-oriented language and program simple distributed and concurrent applications. They will understand the concepts and models underlying these programming languages and will therefore be able to learn other assignment and object-oriented programming languages independently.

Contents

The following content is covered by way of example in the module IN0001:

- Introduction

++ Basic concepts: problem - algorithm - program

++ Imperative program constructs

- Syntax and semantics

++ Syntax of programming languages: regular expressions and context-free grammars

++ Semantics of programs: control flow diagrams

- Basic data structures I:

++ Numbers, strings, fields

++ Sorting by insertion

- Recursion

++ Binary search

++ Types of recursion

- Basic data structures II:

++ Objects, classes, methods

++ Lists, stacks and queues

- Object-oriented programming

++ Inheritance

++ Abstract classes and interfaces

++ Polymorphism

- Programming on a large scale (outlook and embedding in economic, social and ethical contexts)

- Concurrent programming and threads

teaching and learning method

Lecture, combined with your own experimental development of the examples on the computer and access to further literature to clarify technical details.

INTRODUCTION TO COMPUTER ARCHITECTURE

General competences

Description of study/examination achievements

The examination takes the form of a 120-minute written exam. Using simple example tasks on machine-oriented assembler programming, microprogramming and circuit design, students are expected to demonstrate their mastery of the practical concepts of computer architecture. Students are also expected to demonstrate their mastery of the basic theoretical concepts of computer architecture by answering questions. Information sheets are provided to help with the exam; no other aids are permitted.

Intended learning outcomes

After taking part in this module, students will be able to understand computer systems as layered, abstract machines. They will have gained an initial insight into the field of computer architecture and will have mastered the following individual skills:

They will have learned the essential concepts of machine-level programming, microprogramming and circuit design and will be able to apply them. They will have understood the machine instruction cycle based on the processes in the hardware at the register transfer level and will be able to classify computer architectures. They will have learned the principles of modern computer architectures.

Contents

- Function and structure of computer systems from a technical perspective: Von Neumann computers, machine instruction cycle, hardware-software interface

- The Instruction Set Architecture (ISA): Functionality and machine-level assembler programming, calling conventions

- Implementation of machine instructions through microprogramming

- Circuits, switching mechanisms, switching mechanism design using a formal language using the example of VHDL

- Introduction to computer architecture: Microprocessor architectures and system architectures, parallel and distributed systems, memory systems and I/O

teaching and learning method

Using a slide presentation with animations, the lecture introduces the basic concepts and methods of computer architecture and explains them using examples. An accompanying central exercise and accompanying exercises use suitable tasks to deepen the understanding of the content of the lecture and show the application of the various methods using manageable tasks. Homework allows students to delve deeper into the topics through self-study. Solutions to the tasks are discussed in the central exercise and the exercise groups. Presenting your own solution in the accompanying exercise improves communication skills.

DISCRETE STRUCTURES

General competences

Description of Achievement and Assessment Methods

Students are assessed by means of a written 120 - 180 minutes exam consisting of a list of exercises.

Some exercises test if the student can correctly use the mathematical vocabulary about sets, relations, logic, graphs and other mathematical objects introduced in the lectures. Further exercises test if the student is able to select the right logical, combinatorial, graph theoretical, or algebraic technique for the solution of a specific problem, and can apply it correctly.

Intended Learning Outcomes

On successful completion of the module, students will be able to

- understand the elementary vocabulary of discrete mathematics and use logic, algebraic und algorithmic calculi,

- solve combinatoric problems,

- model and solve problems using graph theory, and

- do a quantitive analysis of the efficiency of algorithms.

Content

The lecture introduces elementary concepts and important areas of discrete mathematics that are relevant for informatics students. It covers the following five topics:

1) Basic concepts of sets, relations and functions:

- sets: basic operations, equivalence laws, KV-diagram, countable and uncountable sets, Cantor's Theorem

- relations: join, transitive hull, relational algebra

- functions: basic properties, composition, inverse

2) Fundamentals of Propositional Logic and First-Order Logic:

- Propositional Logic:

- syntax and semantics

- truth tables and their connection to KV-diagrams

- equivalence laws

- CNF, DNF, normalization procedure, equisatisfiability

- SAT-procedure: DPLL, resolution, proof of correctness
- modelling with propositional logic
- Predicate Logic:
- syntax and semantics
- equivalence laws
- modelling with predicate logic

3) Basics of combinatorics:
- counting principles
- drawing of balls from urns: variations, permutations, combinations
- binomial coefficients: symmetry, identities of Pascal and Vandermonde
- distribution problems
- Stirling-numbers of the first and second kind
- ordered and unordered partition functions
- application: load distribution

4) Basics of graph theory:
- basic definitions
- trees
- Euler and Hamilton circuits: Euler's theorem, theorems of Dirac and Ore
- planar graphs: Euler's polyhedron formula, Kuratowski's theorem
- matchings: marriage theorem, augmenting paths
- matchings with preferences: Gale-Shapley's theorem

5) Algebraic basics:
- basic definitions: algebra, group, ring, field
- groups:
- order: Lagrange's theorem, generator, group exponent
- cyclic groups
- basics of number theory: largest common divisor, extended euclidean algorithm, Euler's phi function

- multiplicative groups of integers modulo n

- RSA

Teaching and Learning Methods

The module consists of lectures and exercises. During the lectures students are asked to solve small exercises online. Students also receive weekly assignments, whose solution is discussed in the exercises.

FUNDAMENTALS OF PROGRAMMING (EXERCISES & LABORATORY)

General competences

Description of Achievement and Assessment Methods

Type of Assessment: exercise work

On 7 to 14 exercise sheets questions or programming tasks will be posed, which have to be solved and handed in by the participants in written or electronic form. By that paticipants approve that they are able to do programming in the small by means of an object-oriented programming language such as Java and that they have understood fundamental concepts of Informatics and are able to apply these in order to provide original solutions or programs.

In order to identify the individual contributions of the participants they must be able to defend their solutions interactively. Before the beginning of classes, it shall be announced how the single exercise sheets contribute to the final grade.

Intended Learning Outcomes

After successful completion of the module, participants are acquainted with the programming language Java or a similar object-oriented programming languae and master programming in the small. They are able to realize programs on their own and to apply the fundamental concepts of Informatics as taught in module IN0001, on a basic practical but scientific level.

Content

Accompanying the module IN0001, assignments may excercise and apply for problem solving concepts, such as:

- basic data structures

- recursion

- objects, classes and methods

- lists, queues, and trees

- advanced concepts of object-oriented programming

- concurrency

Teaching and Learning Methods

Approximately a quarter of the module consists of the processing of exercises for the accompanying module IN0001. These exercises deepen the understanding of fundamental concepts of computer science.

During the remainder of the time, the participants develop small sample applications under guidance to develop their programming skills in an object-oriented programming language.

INTRODUCTION TO SOFTWARE ENGINEERING

General competences

Description of Achievement and Assessment Methods

Type of assessment: written exam

The exam takes the form of a 90 minutes written test. The examination consists of describing the main concepts and methods of each phase of the software engineering process. The students have to apply their knowledge to solve small problems. By means of modelling problems, the students have to show their ability to adequately analyze and evaluate given requirements.

Intended Learning Outcomes

After successful completion of this module, students are familiar with the basic concepts and methods of the different phases of a project, e.g. modeling the problem, reuse of classes and components, and delivery of the software. They have the ability to select and apply suitable concepts and methods for concrete problems.

The students know the most important software engineering terms and workflows and are able to analyze and evaluate given problems. In addition, students can solve concrete problems in software engineering, e.g. with the help of design patterns.

Content

Software engineering is the the establishment and systematic use of engineering principles, methods, and tools for the division of work, the development and application of extensive, complex software systems. It deals with the production and development of software, the organization and modelling of data structures and objects, and the operation of software systems. Topics of the lecture include, among others:

- Modeling with UML

- Process models in software development (linear, iterative, agile)

- Requirements elicitation and analysis (functional model, dynamic model, and object model)

- System design (specification, software architecture, architectural patterns, and design goals)

- Object design and implementation (reuse, design patterns, and interface specification)

- Testing (component test, integration test, and system test)

- Configuration management, build management, and release management

- Software maintenance and evolution

- Project organization and communication

Teaching and Learning Methods

By means of a slide presentation with animations, the interactive lecture introduces the basic concepts and methods of software engineering and explains them using examples. Small exercises, e.g. quizzes, modelling, and programming tasks, with individual feedback help students to identify whether they have understood the basic concepts and methods.

Accompanying excercises deepen the understanding of the concepts explained in the lecture by means of suitable group exercises and show the application of the different methods with the help of manageable problems in the different phases of software engineering. Homework enables students to deepen their knowledge in self-study. The presentation of the own solution in the accompanying exercise improves communication skills, which are essential in software engineering. Individual feedback on homework allows students to measure learning progress and improve their skills.

FUNDAMENTALS OF ALGORITHMS AND DATA STRUCTURES

General competences

Description of Achievement and Assessment Methods

The exam takes the form of a 90 minutes written test. In the written exam, based on the questions posed, the students are intended to demonstrate that they have fundamental knowledge in the area of algorithms and data structures. They are able to apply their knowledge successfully in order to solve given problems. In addition, by answering the questions, the students are expected to show that they have profound knowledge of the fundamental algorithmic methods and data structures covered in the module. The students prove that they are able to recognize and analyze basic algorithmic problems and to find efficient solutions within a limited scope of time.

Intended Learning Outcomes

The participants master the basic algorithms and data structures mentioned above. They are able to independently analyze their complexity and apply the corresponding analysis concepts to related algorithmic problems. Furthermore, the participants are able to use the algorithms and data structures handled, if necessary to modify them and to compare different solutions in their quality.

Content

First, the module studies the basics of efficiency and complexity analyses. Basic terminology, complexity measures, the Landau symbols and different machine models are introduced. Then, fundamental data structures and algorithmic problems are studied.

- Data structures for sequences: Dynamic arrays, lists, stacks and queues are investigated. The complexity of each operation is analyzed.

- Hashing: Hashing with chaining, universal hashing as well as various probing methods are examined. Perfect hashing and hash-based algorithms, e.g. for set intersection problems, may also be explored.

- Sorting: First, simple algorithms such as InsertionSort, SelectionSort and BubbleSort are reviewed. Then, advanced algorithms such as MergeSort, HeapSort and QuickSort are investigated. Furthermore, sorting-based algorithms, the lower bound for comparison-based sorting, selection, RadixSort and external sorting may be covered.

- Priority queues: Binary heaps and binomial heaps are presented in the module.

- Search trees: Binary search trees, AVL trees and (a,b)-trees are investigated.

- Graph algorithms: Various graph representations, traversal techniques using DFS/BFS, the computation of 2-connected components and strongly connected components, topological sorting, the computation of the shortest paths and minimum spanning trees are covered. Approaches for solving the Traveling Salesman Problem (TSP) may be studied.

The module may also cover data compression schemes (Huffman, Lempel-Ziv) and simple pattern matching algorithms.

Teaching and Learning Methods

The module consists of lectures and exercises. The content of the lectures is conveyed in presentations of the scientific material. By solving homework assignments, the students are encouraged to work intensively on the respective topics. The solutions of the assignments are discussed in the exercises. The homework assignments are graded so that students get an individual feedback on their learning success.

FUNCTIONAL PROGRAMMING AND VERIFICATION

General competences

Description of Achievement and Assessment Methods

The exam takes the form of a 120 minutes written test. Small programming tasks allow to assess whether the students master a functional programming language and are able to realize small implementation problems. By inferring simple invariants they demonstrate that they have understood the principles of program verification and are able to apply these.

The successful completion of homework asignments may contribute to the grade as a bonus. The exact details for this are announced timely at the begin of the lecture.

Intended Learning Outcomes

After successful completion of the module, participants understand the key concepts of functional programming languages. They are able to solve well presented tasks in a functional programming language. Therefore, they are able to acquire programming skills on their own also in further functional programming languages. They also are familiar with the most important techniques for the verification of imperative and functional programming language and can apply them to simple programs.

Content

Among others, the module IN0003 is concerned with the following topics:

- Correctness of imperative programs

++ Verification according to Floyd or Hoare

++ Termination

++ Procedures

- Basic concepts of functional programming

++ Values, variables, functions

++ Data++structures, pattern matching

++ Higher order functions

++ Polymorphic types

++ Programming in the large: Structures and Functors

++ Correctness of functional programs

+++ Semantics of functional programs

+++ Verification of functional programs

Teaching and Learning Methods

By means of a presentation, either by slides or whiteboard, the lecture transports the concepts of verification and the programming language and illustrates them by examples.

Accompanying assignments for individual study deepen the understanding of the concepts explained in the lecture, and train students to apply these to the verification of small programs and to master programming in the given programming language.

LINEAR ALGEBRA FOR INFORMATICS

General competences

Description of Achievement and Assessment Methods

The module examination is based on a written exam (90 minutes). Students have to show their knowledge of basic concepts of linear algebra and its applications. They have to discuss solutions for vector and matrix calculations in limited time.

Intended Learning Outcomes

After having attended this module, the students are familiar with the basic notions and methods of linear algebra. They are proficient in vector and matrix calculus as well as in the theory and application of linear mappings including their representation by matrices, and they are aware of determinants and eigenvalues as an essential characteristics of matrices. The students can handle the objects of liner algebra in computational and algorithmic ways, and can apply this to problems in computer science.

Content

- Complex numbers

- Vector and matrix calculus

- Systems of linear equations

- Vector spaces

- Linear combinations and bases

- Dimension

- Linear maps and representation matrices

- Determinants

- Eigenvalues

- Scalar products

- Symmetric matrices

Teaching and Learning Methods

The module is offered as lectures with accompanying practice sessions. In the lectures, the contents will be presented in a talk with demonstrative examples, as well as through discussion with the students. The lectures should motivate the students to carry out their own analysis of the themes presented and to independently study the relevant literature. Corresponding to each lecture, practice sessions will be offered, in which exercise sheets and solutions will be available. In this way, students can deepen their understanding of the methods and concepts taught in the lectures and independently check their progress. At the beginning of the module, the practice sessions will be offered under guidance, but during the term the sessions will become more independent, and intensify learning individually as well as in small groups.

BASIC PRACTICAL COURSE: COMPUTER ARCHITECTURE

General competences

Description of Achievement and Assessment Methods

The examination is carried out in the form of a project work in small groups of 3 students. The project work consists of an implementation as well as an elaboration of approx. 10 pages and a presentation of 15 minutes, which are weighted to one third each. By working on an extensive project task, it is demonstrated that the students are able to identify a problem together in a limited time, to solve it with an efficient implementation, and to analyze and qualitatively evaluate this solution. The communicative competence and the ability to analyze, interpret and evaluate one's own problem solution is tested by elaboration and presentation.

Intended Learning Outcomes

After participating in the module, the students are able, in the context of modern computing systems (modern x86-64 processors, GPUs) to analyze the performance characteristics of programs, to use these results to develop targeted optimizations and thus to program complex hardware in a time- and energy-efficient manner and to evaluate their own implementations.

Content

- Performance analysis techniques

- Benchmarking techniques

- Profiling tools, e.g. perf, gprof, Callgrind

- Use of hardware performance counters

- Basics of C programming

- Data structures, functions, strings, pointer arithmetic, I/O

- Makefiles, debugging techniques, GDB, assertions

- Basics of compiler optimizations

- Framework: Application Binary Interface, defined

behavior

- Influence of program annotations (pointer aliasing, alignment,

etc.) on performance

- Basic models for parallel programming, Flynn's classification, etc.

classification

- Optimization of instruction parallelism

- Specialization in the context of the project work in one of the following

areas:

- Exploitation of vector units of modern processors
- SIMD instructions in machine code
- SIMD-Intrinsics in higher programming languages
- Multi-threading with Pthreads or OpenMP
- Synchronization, atomic operations
- Cache optimizations
- Programming models for heterogeneous systems (e.g. GPUs, FPGAs)

Teaching and Learning Methods

In the first half of the semester, the necessary basics are taught in weekly practicals through classroom exercises and homework. The knowledge acquired in this phase has to be applied in group projects in the second half of the semester. Each group, consisting of 3 participants, is assigned a project to be completed in a given time. In addition to the implementation of the project, a paper has to be written and a presentation has to be given.

FUNDAMENTALS OF DATABASES

General competences

Description of Achievement and Assessment Methods

The academic assessment will be done by a 90 minutes written exam. Assignments checking knowledge verify the familiarity with the main concepts of relational database systems. Transfer assignments and small scenarios check the ability to apply and evaluate these concepts systematically and in a qualified manner.

Intended Learning Outcomes

Students are able to apply the essential concepts of relational database systems and can use and evaluate them systematically and in a qualified manner.

The students have the expertise to systematically use a database system starting from the conceptual design to the implementation design to the physical design. They are able to formulate even complex queries in SQL and have a basic understanding of logical and physical optimization based on relational algebra. Furthermore they know how to safe-guard a database application with respect to recovery, concurrency control and authorization.

Content

SQL, data integrity, theory of relational database design, physical data organisation (storage structures, index structures), query processing, transaction management, main features of error handling (recovery, backup) and multi-user synchronisation, security aspects (authorization), XML data modeling (optional); in the exercise the content is practiced along concrete examples

Teaching and Learning Methods

Lecture, exercises, problems for individual study, web interface to the data base system HyPer for actively testing SQL queries and self-study of query plans.

BASIC PRINCIPLES: OPERATING SYSTEMS AND SYSTEM SOFTWARE

General competences

Description of Achievement and Assessment Methods

In the 90 minutes written exam students have to show their understanding of the subject, like resource management and the usage of systems software. They have to prove to be able to identify a given problem and find solutions within limited time. Additionally, they shall apply the mechanisms presented and demonstrate their skill in systems programming by solving small assignments.

Intended Learning Outcomes

After visiting this module, students are able to understand the basics, problems and solutions of operating systems and current developments. In addition, they understand the components like process and memory management and they are able to analyze and evaluate different strategies and techniques. They learn to apply the acquired basic knowledge to new developments in the area of operating systems as well as system software.

Content

Basic concepts: Operating systems; concurrency; parallel programming; low-level programming (processes, memory, communication, resource management; models (abstract, formal) for concurrency, e.g. petri nets; mutual exclusion, synchronization, deadlocks; compiler/linker/loader with library integration, transition to (adequate) hardware basic, machine-oriented programming and C; I/O especially as preparation for networking)

Teaching and Learning Methods

By means of a slide presentation, the lecture introduces the basic concepts and methods of operating systems and explains them using examples.

Accompanying exercises deepen the understanding of the concepts explained in the lecture by means of suitable group exercises and show the application of the different methods with the help of manageable problems in the different aspects of operating system decomposition.

Additional programming assignments enable students to deepen their knowledge in self-study. Feedback and help in programming exercise sessions allow students to measure learning progress and improve their skills.

IT SECURITY

General competences

Description of Achievement and Assessment Methods

The examination performance is provided in the form of a 90-minute written exam. Knowledge questions test familiarity with basic concepts and methodical procedures for securing components and systems as well as familiarity with basic protocols. Smaller tasks test the ability to apply security concepts in a targeted and correct manner, or to recognize security eficiencies.

Intended Learning Outcomes

After successful participation in this module, participants understand the basic concepts, methods and protocols for protecting data and systems from manipulation and misuse at a fundamental, practice-oriented, scientific level. They understand the causes of security problems in today's systems, can reproduce the relationships between protection mechanisms and the problems they address and apply them to case studies. They also master basic methods of operational IT security management.

Content

- Basic concepts of IT security,

- Selected security problems of today's systems and selected attacks,

- Basics and practical methods of applied cryptography

- Basic concepts and protocols for identification/authentication, access control and system security

- Fundamentals of network security: firewall concepts and practice-relevant protocols for network security

- Methods of operational IT risk and security management (ISM)

Teaching and Learning Methods

Lecture supplemented by exercise with practical parts to gain a detailed understanding of conceptual and technical issues by means of concrete tasks, including simple practical programming tasks.

ANALYSIS FOR INFORMATICS

General competences

Description of Achievement and Assessment Methods

The module examination is based on a written exam (90 minutes). Students have to show their knowledge of basic methods of real analysis and of the concepts of convergence and approximation. They are able to provide adequate solutions to differential and integral calculus problems and simple differential equations under time pressure.

Intended Learning Outcomes

Having attended this module, the students are able to understand the elementary notions and methods of real analysis as well as of convergence and approximation. They are moreover able to apply the methods of differential calculus of functions of a single and of several variables, as well as of integral calculus of a function of a single variable, and they understand important function classes and simple examples of differential equations.

Content

Basics on real numbers:

Ordering of the real numbers

Infimum, supremum

Rational numbers are dense in the reals

Triangle inequality, Cauchy Schwarz inequality

Sequences:

Convergence in C and R

Improper convergence (convergence to plus/minus infinity)

Properties of limits

asymptotic equality of sequences

monotone sequences

Series:

Convergence and absolute convergence of series

Important examples: harmonic series, geometric series, exponential series,

alternating series

Criteria for convergence (among others comparison test, ratio test)

Rearrangement, Cauchy product

Continuity:

Intermediate value theorem

Minima and maxima of continuous functions

Compact sets

Inverse functions

Important classes of functions:

Polynomials

Rational functions

Exponential function and logarithm

Trigonometric functions

Differentiability of functions of a single variable:

Landau symbols

Differentiation rules (product rule, quotient rule, chain rule,

derivative of the inverse function)

Mean value theorem

Higher derivatives

Taylor formula

Power series

L'Hospital's rule

Curve sketching

Integration in one variable:

Fundamental theorem of calculus

Antiderivative

Integration by parts, substitution rule

Improper integrals

Integrals depending on a parameter

Differential calculus of functions of several variables:

partial derivatives, gradient, Jacobi and Hesse matrix

Necessary and sufficient criteria for local extrema

Plane curves

Elementary introduction to differential equations:

Classification, examples

Initial value problems for separable equations

Linear differential equations

Teaching and Learning Methods

The module is offered as lectures with accompanying practice sessions. In the lectures, the contents will be presented in a talk with demonstrative examples, as well as through discussion with the students. The lectures should motivate the students to carry out their own analysis of the themes presented and to independently study the relevant literature. Corresponding to each lecture, practice sessions will be offered, in which exercise sheets and solutions will be available. In this way, students can deepen their understanding of the methods and concepts taught in the lectures and independently check their progress. At the beginning of the module, the practice sessions will be offered under guidance, but during the term the sessions will become more independent, and intensify learning individually as well as in small groups.

INTRODUCTION TO COMPUTER NETWORKING AND DISTRIBUTED SYSTEMS

General competences

Description of Achievement and Assessment Methods

The exam takes the form of a 90 minutes written test. Comprehension questions and calculation tasks allow to assess acquaintance with the technologies and methods of computer networks and distributed systems, and the understanding obtained by implementation of protocol mechanisms. Calculation tasks also allow to assess the ability to determine the performance of selected computer networks and distributed applications.

Intended Learning Outcomes

After successful completion of the module, participants understand the key concepts of technologies and methods of computer networks and distributed systems and are able to use key layered network architecture protocols to explain what protocol mechanisms are used in each layer and how they work. They understand the architecture of distributed applications like the World Wide Web based on Internet protocols, and the architecture of computer networks. Participants can determine the performance of selected networks and distributed applications, and can implement specific protocol mechanisms.

Content

- Computer networking

++ Overview: Computer networking and the Internet

+++ components (router, switches, clients, server)

+++ design (topology, routing, packets vs. virtual circuits)

+++ layered system structure (OSI and Internet)

+++ historical background

++ concepts used by multiple layers (covered within the appropriate layers):

+++ Addressing

+++ Error detection

+++ Coding and modulation

+++ Media access protocols

+++ Flow control

+++ Connection management

+++ Packet vs. virtual circuit switching

++ Layers:

+++ Application layer: application protocols and applications

++++ Tasks and interface

++++ Examples: HTTP, DNS, SMTP (Mail), Peer-to-Peer protocols

+++ Transport layer

++++ Tasks and interface

++++ Examples: TCP and UDP

+++ Network layer

++++ Tasks and interface

++++ Routing: link state vs. distance vector protocols

++++ Addressing: IP Addresses

++++ Examples: IP, Routing in the Internet

+++ Link layer

++++ Tasks and interface

++++ Examples: Ethernet, Wireless LAN

+++ Physical layer

++++ Tasks and interface

++++ Examples

- Distributed systems:

++ Middleware, e.g. RPC

++ Web Services

- General tasks:

++ Network management

++ IT security

+++ Basics of cryptography

+++ Authentication, privacy, integrity

+++ Protocols with security mechanisms, e.g.: IPsec, PGP, Kerberos, SSL, SSH, ...

+++ Firewalls, intrusion detection

Content of the Exercises:

The exercises cover comprehension questions and calculation tasks and target determination of performance of protocols and mechanisms of specific layers (Physical Layer, Data Link Layer, Network Layer, Transport Layer). Programming exercises address implementation of specific protocol mechanisms.

Teaching and Learning Methods

The interactive lecture with slide presentations, animations, demonstrations and life programming presents the basic knowledge of computer networks and distributed systems and explains them using examples. Quizzes help students to recognize whether they have understood the basic concepts and essential contexts. Homework enables students to deepen their knowledge in self-study. Accompanying exercises deepen the understanding of the contents of the lecture by means of suitable tasks and show the application of the various methods on the basis of manageable problems. The presentation of the own solution in the accompanying exercise improves the communication skills and allows to compare the own learning progress with that of other students. Programming tasks allow computer-aided deepening and application of conceptual knowledge to practical problems.

INTRODUCTION TO THEORY OF COMPUTATION

General competences

Description of Achievement and Assessment Methods

The exam takes the form of a 180 minutes written test. Knowledge questions allow to assess acquaintance with concepts of Theoretical Informatics, algorithmic questions assess the ability to apply known algorithms to concrete problems or to design small new algorithms, and deductive questions assess the ability to reason logically about the concepts of the course.

Intended Learning Outcomes

After successfully completing this module, the students understand the core concepts of the theory of computation on a basic but scientific level. They know what regular expressions, contextfree grammars, the Chomsky hierarchy, finite automata and Turing machines are. They can define formal languages with the appropriate grammars or machines. They can prove that a given language cannot be defined with a given class of grammars or machines. They can prove that certain grammars and machines are equivalent and they can transform them into each other algorithmically. They can explain the basic concepts of complexity theory and can reduce decision problems algorithmically to each other under given complexity limitations.

Content

Formal languages, grammars, Chomsky hierarchy.

Regular languages: DFA, NFA with and without ε-transitions, regular expressions and translations between them; systems of language equations; closure under boolean operations; Arden's lemma; pumping lemma; decision problems; minimization; Myhill-Nerode theorem.

CFLs: PDAs and translation between CFGs and PDAs; proof that DPDAs are weaker than PDAs; closure properties; CYK algorithm; pumping lemma; Chomsky and Greibach normal forms.

Context-sensitive languages and LBAs.

Computability: computability, decidability, semi-decidability, recursive-enumerability and their relationships; existence of non-computable problems; Turing machines, accepted languages, type-0 languages: equivalence of Turig machines, While-programs and Goto-programs; primitive and μ-recursive functions; reductions between problems; the

Halting problem; universal Turing machines; Rice's theorem; Rice-Shapiro theorem; undecidability of the Post Correspondence Problem and important problems on CFGs.

Complexity theory: time and space complexity classes; polynomial-time reductions; the classes P and NP; NP-completeness; Cook's theorem; important NP-complete problems and reductions between them.

All proofs are covered.

Teaching and Learning Methods

The module consists of lectures and exercises. In the lectures, the material is presented by the teacher, in dialogue with the students. During the exercises, the students work on given exercises either individually or in small groups with help from the supervisors. Exercises and homework are primarily pen and paper based but can also involve computer-based components.

DISCRETE PROBABILITY THEORY

General competences

Description of Achievement and Assessment Methods

Students will be assessed by a 120 minutes written exam, which consists of three types of exercises. Comprehension exercises test if the student understands the basic concepts and theorems of the lecture. They require students to apply these concepts to examples. Algorithmic exercises test if the student knows and is able to apply the presented rules to selected inputs. Modelling exercises test the ability of the student to use the mathematical tools from the lectures to model and solve concrete problems.

Intended Learning Outcomes

After successful completion of the module

- Participants are familiar with important concepts of discrete and continuous probability spaces and stochastic processes and can in large part deduce them themselves

- master calculation rules for the determination and estimation of probabilities, expected values and variances,

- are able to map real problems to abstract probability spaces and

- can easily apply simple statistical tests.

Content

-Fundamentals of (discrete) probability theory

++ probability spaces, events, principle of inclusion/exclusion, Boolean inequality, conditional probability, multiplication theorem, total probability theorem, Bayesian theorem, independence

++ random variables, expected value, variance, linearity of expected value, conditional random variables and their expected value, variance, moments and central moments, several random variables and their common density & distribution, independence from random variables, moments of composite random variables, indicator variables

++ discrete distributions: Bernoulli distribution, binomial distribution, geometric distribution, coupon collector problem, Poisson distribution, relationships among distributions

++ methods for estimating probabilities, inequalities of Markov and Chebyshev, Chernoff barriers

++ Law of large numbers

++ Probability-generating functions and their application to distributions, moment-generating functions with various applications

-continuous probability spaces

++ continuous random variables, Kolmogorov axioms, sigma algebras, Lebesgue integrals, computing with continuous random variables, simulation of random variables

++ continuous distributions: Equal distribution, normal distribution and linear transformation, exponential distribution and waiting processes, relationship with discrete distributions

++ several continuous random variables, boundary distributions and independence, sums of random variables

++ Moment generating functions for continuous random variables

++ Central limit theorem

-Inductive Statistics

++ estimation variables, maximum likelihood principle, confidence intervals, hypothesis testing, statistical test development and application

-Stochastic processes

++ processes with discrete time, Markov chains, transition probabilities, arrival probabilities, transition times, return times, fundamental theorem for ergodic Markov chains

Teaching and Learning Methods

The module consists of lectures and exercises. During the lectures students are asked to solve small exercises online. Students also receive weekly assignments, whose solution is discussed in the exercises.

ADVANCED SEMINAR COURSE

General competences

Description of Achievement and Assessment Methods

Type of Assessment: Research elaboration

Before the start of the Seminar (usually at the end of the previous semester) each participant receives an advanced topic on which she prepares a scientific essay by using scientific literature which has possibly been found on her own. The length of the scientific essay varies according to the topic.

The results of the work will be presented to the other participants of the seminar in oral form, supported by visual media such as projector or slides. Evaluated is also how the student responds to questions, suggestions and discussion points to her work and presentation, and how she takes part in the discussion on the work and presentations of other participants. By that students demonstrate their expertise for critical analysis of presented scientific contents.

Before the start of the seminar the respective lecturer will announce, how the various ingredients are weighted for the calculation of the module grade.

Intended Learning Outcomes

Participants have the necessary methodological and interdisciplinary skills to independently write a scientific essay about an advanced topic in computer science, as well as to present and discuss its content. The students can work with scientific literature (i.e. search, categorize, prioritize, cite, ...). They master the required presentation and discussion techniques.

Content

- Independent assessment of a advanced scientific theme

- Preparation of a term paper with a section on related work

- Presentation and discussion of scientific results

This module is offered by all faculty members. They select suitable topics from their research area and support the students in learning the technical and scientific skills.

Teaching and Learning Methods

Participants independently assess an advanced scientific topic. The accompanying elaboration summarizes the essential concepts of the topic and provides an overview of the sources.

In addition, they present and discuss their results with the other participants.

Accompanying support provides assistance in the scientific development and feedback on the design of the elaboration as well as the presentation.

NUMERICAL PROGRAMMING

General competences

Description of Achievement and Assessment Methods

Type of Assessment: written exam

The exam takes the form of a 90 minutes written test. In the exam students should prove to be able to identify a given problem and find solutions within limited time. The examination will completely cover the content of the lectures. The answers will require own formulations. Exam tasks allow to assess the participants' understanding of fundamental principles of numerical programming. This includes the evaluation of the participants' knowledge concerning the most important basic numerical algorithms that are relevant for applications from computer science. Small code snippets allow to assess the students' capability to apply and implement these algorithms.

Export

Export

General data (module handbook)

Module Level

Bachelor

Abbrevation

Subtitle

Duration

one semester

Occurrence

winter/summer semester

Language

German

Related Programs

Work load

Total Hours

180

Contact Hours

75

Self-study Hours

105

Study and examination performance

Description of Achievement and Assessment Methods

Type of Assessment: written exam

The exam takes the form of a 90 minutes written test. In the exam students should prove to be able to identify a given problem and find solutions within limited time. The examination will completely cover the content of the lectures. The answers will require own formulations. Exam tasks allow to assess the participants' understanding of fundamental principles of numerical programming. This includes the evaluation of the participants' knowledge concerning the most important basic numerical algorithms that are relevant for applications from computer science. Small code snippets allow to assess the students' capability to apply and implement these algorithms.

Exam retake next semester

J

Exam retake at the end of semester

N

Description

Prerequisites (recommended)

MA0901 Linear Algebra for Informatics, MA0902 Analysis for Informatics

Intended Learning Outcomes

At the end of the module, participants understand the fundamental principles of numerical programming, know the most important basic numerical algorithms (e.g. Gaussian elimination, QR-algorithm, Neville-algorithm, Jacobi und Gauß-Seidel-algorithm, Newton's method) that are relevant for applications from computer science, and are capable to apply and implement these algorithms.

Content

- Floating point arithmetic (rounding error analysis, condition and stability)

- Interpolation (polynomial interpolation, splines, trigonometric interpolation, fast Fourier transform)

- Quadrature (simple and composite rules, extrapolation, Gaussian quadrature)

- Solving linear systems (Gaussian elimination, LR-factorization, pivoting, least squares)

- Ordinary differential equations (finite differences, Euler and Runge Kutta methods, discretization error, consistency, stability and convergence, multi-step methods)

- Iterative methods (relaxation methods: Jacobi and Gauss-Seidel, minimization methods: steepest descent, fix point iteration, nonlinear equations: from bisection to Newton)

- The symmetric eigenvalue problem (motivation, condition, vector iteration, QR iteration, reduction algorithms)

Teaching and Learning Methods

This module comprises lectures and accompanying exercises. The contents of the lectures will be taught by talks and presentations.

Students will be encouraged to study literature and to get involved with the topics in depth. In the exercises, concrete problems will be solved - partially in teamwork - and selected examples will be discussed.

BACHELOR PRACTICAL COURSE

General competences

Description of Achievement and Assessment Methods

Type of assessment: project work

Different phases of a software project will be handled by the participants in teams. Individual teams may deal with only one or more selected phases of the project. By their work, the participants demonstrate that they can deal with a given project contract or subcontract within a given time and using appropriate tools in teams. During design and implementation, they demonstrate that they are able to apply methods and models of engineering. They can assess the risks and problems in the various phases of a software project and are able to deal with these difficulties. A written report is prepared and possibly presented orally in order to evaluate the communicative competence in documentation and presentation of results. Before the start of the practical course the respective lecturer will announced how the individual components are weighted in the calculation of the module grade.

Intended Learning Outcomes

Participants are able to design and implement small software systems in a team. They know how to apply engineering methods and models from different areas of informatics in order to realice the various phases of the project. They can judge the risks and typical problems encountered in software projects and know methods to cope with these. They are able to report about their project work in written as well as oral form.

Content

- Implementation of a software project or subproject in teams

- Methods and models of engineering for software development in various areas of informatics

- Techniques for documentation and presentation of results or intermediate results in software development

This module is offered by different chairs. The chairs select the topic for the projects based on their research and education preferences (e.g., databases, information systems, networks, groupware, graphics, robotics, image recognition).

Teaching and Learning Methods

Participants exercise systematic software engineering for a small system in small teams with a precise task description with tight time constraints (design, implementation, test). Intermediate results of the team work have to be presented. Design, project plans and implementation have to be documented.

BACHELOR’S THESIS

General competences

Description of Achievement and Assessment Methods

Type of Assessment: Research Paper

The Bachelor's Thesis (100% of the module grade) is a written scientific report with a duration of 4 months, where the students apply research methods to elaborate a specific scientific topic in informatics and report the results. The Bachelor's Thesis shall verify that students are able to apply the knowledge and methodological skills acquired during the studies to a specific topic in this area.

It should be between 30 and 60 pages long. The Bachelor's Thesis includes a presentation of the results. With the presentation, students demonstrate that they are able to communicate scientific issues and their findings to third parties in a comprehensible manner. The presentation is not included in the assessment.

Intended Learning Outcomes

At the end of the module "Bachelor's Thesis" students are able to elaborate a scientific topic in informatics in an autonomous and systematic way. The students are able to become acquainted with a scientific topic. They have learned to understand a specific scientific question and apply scientific methods to answer the question. The students are able to discuss their results in a critical way.

Content

The Bachelor's Thesis focuses on a research topic in informatics. The Thesis is always supervised by an expert professor at TUM, often in cooperation with industry or a research institute. The topic of the thesis is designed in a way that it can be treated extensively and parallel to other modules within four months.

Teaching and Learning Methods

The creation of the thesis requires the students to deal soundly with a scientific subject. To do this, they apply the knowledge and methodological skills acquired during the studies and create an elaborated scientific report within the set time frame.

EFFICIENT ALGORITHMS AND DATA STRUCTURES

General competences

Description of study/examination achievements

The examination takes the form of a 150-minute written exam. In this, students demonstrate, based on the tasks set, that they have mastered the conceptual and mathematical foundations of algorithm analysis. Furthermore, students show that they have fundamental and advanced knowledge in the area of efficient data structures and algorithms. They demonstrate that they can recognize and analyze typical algorithmic problems and find ways to solve them in a limited amount of time.

Intended learning outcomes

After completing the module, students are able to analyze and evaluate the runtime and storage space requirements of algorithms. In addition, they have a basic understanding of how numerous fundamental algorithms and data structures work. This understanding enables them to independently develop algorithms and data structures for new problems.

Contents

The module first covers the basics of algorithm analysis. Then fundamental data structures and basic algorithmic problems are covered. In the area of the basics of algorithm analysis, the module studies various machine models, complexity measures and the solving of recursion equations. In the area of fundamental data structures, the module introduces various search trees, hash methods, priority queues and union-find data structures. In the area of basic algorithms, the module focuses on the development of numerous maxflow and mincut algorithms as well as algorithms for the matching problem.

The module first covers the basics of algorithm analysis. Then fundamental data structures and basic algorithmic problems are covered. In the area of the basics of algorithm analysis, the module studies various machine models, complexity measures and the solving of recursion equations. In the area of fundamental data structures, the module introduces various search trees, hash methods, priority queues and union-find data structures. In the area of basic algorithms, the module focuses on the development of numerous maxflow and mincut algorithms as well as algorithms for the matching problem.

teaching and learning method

The module consists of a lecture and an accompanying exercise. The content of the lecture is conveyed in the lecture and through presentations. Students are encouraged to engage with the content of the topics in particular by solving exercise sheets. The solutions to the exercises are discussed in the exercise session. In addition, the students receive individual feedback on their learning success by correcting the exercise sheets.

EFFICIENT ALGORITHMS AND DATA STRUCTURES II

General competences

Description of study/examination achievements

The examination takes the form of a 150-minute written exam. In this, students demonstrate, based on the tasks set, that they have fundamental and advanced knowledge of algorithm analysis and can successfully apply this knowledge to solving problems. In addition, when solving the tasks set, students demonstrate that they have mastered the data structures and advanced algorithmic methods of combinatorial optimization covered in the module. Students demonstrate that they can recognize and analyze complex algorithmic problems in a limited amount of time and find ways to solve them efficiently.

Intended learning outcomes

After completing the module, students have extensive knowledge of advanced algorithmic methods, particularly in the field of linear optimization. In addition, they know the importance of approximation algorithms for solving NP-complete problems. They know various techniques for obtaining approximate solutions to problems in the field of combinatorial optimization and can independently apply these techniques to new problems that arise in a scientific and/or professional application.

Contents

Linear optimization - modeling - simplex method - Seidel's algorithm - ellipsoid method - Karmarkar approximation algorithms - greedy methods - local search - rounding methods - primal/dual methods

Linear optimization - modeling - simplex method - Seidel's algorithm - ellipsoid method - Karmarkar approximation algorithms - greedy methods - local search - rounding methods - primal/dual methods

teaching and learning method

The module consists of a lecture and an accompanying exercise. The content of the lecture is conveyed in the lecture and through presentations. Students are encouraged to engage with the content of the topics in particular by solving exercise sheets. The solutions to the exercises are discussed in the exercise session. In addition, the students receive individual feedback on their learning success by correcting the exercise sheets.

COMPLEXITY THEORY

General competences

Description of study/examination achievements

The examination takes the form of a 120-minute written exam. In this, students demonstrate, based on the tasks set, that they have fundamental and advanced knowledge in the area of complexity theory. Students demonstrate that they can recognize and analyze complexity theory problems in a limited amount of time and find ways to find an efficient solution.

Intended learning outcomes

The participants of the module know the central methods of complexity theory. They know about computation models, complexity classes, reductions, completeness and are familiar with advanced concepts such as diagonalization, the polynomial hierarchy, space complexity, alternation, Boolean circuits, randomization and interactive proof systems. Furthermore, they can apply the corresponding methods and concepts to analyze new problems in their complexity.

Contents

The module first deals thoroughly with Turing machines. Time and space complexity are studied. Furthermore, circuits are examined as a computational model. The complexity classes L, NL, P, NP, PSPACE, EXP, NEXP, PH are introduced. Then, completeness and fundamental structural relationships between complexity classes are derived. The module also deals with the concept of alternation, Boolean circuits, randomization and interactive proof systems.

The module first deals thoroughly with Turing machines. Time and space complexity are studied. Furthermore, circuits are examined as a computational model. The complexity classes L, NL, P, NP, PSPACE, EXP, NEXP, PH are introduced. Then, completeness and fundamental structural relationships between complexity classes are derived. The module also deals with the concept of alternation, Boolean circuits, randomization and interactive proof systems.

teaching and learning method

The module consists of a lecture and an accompanying exercise session. The content of the lecture is conveyed in lectures and presentations. Students are encouraged to engage with the content of the topics, particularly by solving exercise sheets. The solutions to the exercises are discussed in the exercise session. In addition, students receive individual feedback on their learning success by correcting the exercise sheets.

PARALLEL ALGORITHMS

General competences

Description of study/examination achievements

The examination takes the form of a 120-minute written exam. In this, students demonstrate, based on the tasks set, that they are familiar with fundamental concepts of parallel computing, machine models and the basics of parallel complexity theory. Students also demonstrate that they have the ability to design and evaluate parallel algorithms.

Intended learning outcomes

The participants of the module master the fundamental concepts of parallel computing. They know machine models, master the basics of parallel complexity theory and can design and evaluate parallel algorithms.

Contents

The module studies parallel machine models in depth. It develops elementary parallel algorithms. It also introduces advanced algorithms and methods for routing and sorting.

The module studies parallel machine models in depth. It develops elementary parallel algorithms. It also introduces advanced algorithms and methods for routing and sorting.

teaching and learning method

The module consists of a lecture and an accompanying exercise session. The content of the lecture is conveyed in lectures and presentations. Students are encouraged to engage with the content of the topics, particularly by solving exercise sheets. The solutions to the exercises are discussed in the exercise session. In addition, students receive individual feedback on their learning success by correcting the exercise sheets.

ADVANCED NETWORK AND GRAPH ALGORITHMS

General competences

Description of study/examination achievements

The examination takes the form of a 120-minute written exam. In this, students demonstrate, based on the tasks set, that they are familiar with fundamental concepts of network and graph algorithms. Students demonstrate that they can recognize and analyze corresponding algorithmic problems within a limited time and find ways to an efficient solution.

Intended learning outcomes

Participants of the module are able to evaluate specialized network problems in terms of their complexity and solve them by applying advanced algorithms to graphs. This includes in particular the analysis and evaluation of typical properties such as robustness or susceptibility to errors of complex networks. Participants are also able to develop algorithms to optimize various network parameters.

Contents

The module provides an in-depth look at the graph algorithms known from the lectures "Fundamentals: Algorithms and Data Structures" and "Efficient Algorithms and Data Structures 1/2" with regard to new algorithmic methods, analysis and areas of application. The module focuses on the development of centrality indices and associated algorithms as well as the investigation of density in (sub)graphs. Advanced algorithms for connection problems and for the assignment problem based on the Hungarian method are developed. Other optional topics in the module are graph coloring, clustering, network statistics, network comparison, algebraic methods, spectral analysis and robustness investigations.

The module provides an in-depth look at the graph algorithms known from the lectures "Fundamentals: Algorithms and Data Structures" and "Efficient Algorithms and Data Structures 1/2" with regard to new algorithmic methods, analysis and areas of application. The module focuses on the development of centrality indices and associated algorithms as well as the investigation of density in (sub)graphs. Advanced algorithms for connection problems and for the assignment problem based on the Hungarian method are developed. Other optional topics in the module are graph coloring, clustering, network statistics, network comparison, algebraic methods, spectral analysis and robustness investigations.

teaching and learning method

The module consists of a lecture and an accompanying exercise session. The content of the lecture is conveyed in lectures and presentations. Students are encouraged to engage with the content of the topics, particularly by solving exercise sheets. The solutions to the exercises are discussed in the exercise session. In addition, students receive individual feedback on their learning success by correcting the exercise sheets.

RANDOMIZED ALGORITHMS

General competences

Description of study/examination achievements

The examination takes the form of a 120-minute written exam. In this, students demonstrate, based on the tasks set, that they have fundamental and advanced knowledge in the area of randomized algorithms. Students demonstrate that they can recognize and analyze corresponding algorithmic problems within a limited time and find ways to find an efficient solution.

Intended learning outcomes

The participants of the module have a thorough knowledge of various basic principles of randomized algorithms, associated computational models, the corresponding mathematical analyses (estimates according to Markov, Chebyshev, Chernoff) and several paradigmatic applications.

Contents

The module first develops basic results in the area of randomized algorithms. The randomized quicksort algorithm as well as randomized algorithms for min-cut, binary planar partition and the verification of matrix multiplication are derived. The module covers the concepts of Las Vegas and Monte Carlo algorithms and defines various randomized complexity classes. It covers game theory techniques such as Yao's minimax principle and applies this to game tree evaluation. The module introduces moments and develops elementary inequalities such as the Markov inequality and the Chebyshev inequality. These are used to analyze the coupon collector problem as well as the problems of median calculation and stable marriage in detail. The module also develops Chernoff bounds and uses them to analyze routing and wiring problems. The probabilistic method is presented and applied to numerous problems. Finally, the module studies randomized data structures such as treaps as well as universal and perfect hashing.

The module first develops basic results in the area of randomized algorithms. The randomized quicksort algorithm as well as randomized algorithms for min-cut, binary planar partition and the verification of matrix multiplication are derived. The module covers the concepts of Las Vegas and Monte Carlo algorithms and defines various randomized complexity classes. It covers game theory techniques such as Yao's minimax principle and applies this to game tree evaluation. The module introduces moments and develops elementary inequalities such as the Markov inequality and the Chebyshev inequality. These are used to analyze the coupon collector problem as well as the problems of median calculation and stable marriage in detail. The module also develops Chernoff bounds and uses them to analyze routing and wiring problems. The probabilistic method is presented and applied to numerous problems. Finally, the module studies randomized data structures such as treaps as well as universal and perfect hashing.

teaching and learning method

The module consists of a lecture and an accompanying exercise session. The content of the lecture is conveyed in lectures and presentations. Students are encouraged to engage with the content of the topics, particularly by solving exercise sheets. The solutions to the

exercises are discussed in the exercise session. In addition, students receive individual feedback on their learning success by correcting the exercise sheets.

ALGORITHMIC GAME THEORY

General competences

Description of study/examination achievements

The examination takes the form of a 120-minute written exam. This is intended to demonstrate that, within a limited time, - a game theory problem is recognized, - connections to questions dealt with in the module are made, and - ways to a solution can be found. In addition, in accordance with APSO Paragraph 6, Section 5(2), there will be a voluntary mid-term assignment in the form of 10-20 exercises to be completed online. This mid-term assignment will be offset against the exam grade at a ratio of 80:20 (exam:mid-term assignment) and will only be used to improve the grade of a passed exam. No aids are permitted in the exam other than a self-made information sheet.

Intended learning outcomes

After successfully completing the module, students will be able to - understand the fundamentals of algorithmic game theory, - analyze different representations of n-player games, - calculate and compare different solution concepts, - examine these solution concepts from a complexity theory perspective, and - analyze and outline simple algorithms for determining solutions to game theory problems.

Contents

Algorithmic game theory is a young research area at the intersection of theoretical computer science, mathematics and economics that deals with optimal strategic behavior in interactive situations. This lecture pays particular attention to the algorithmic aspects of game-theoretic solution concepts such as Nash equilibria and the design of economic mechanisms.

Algorithmic game theory is a young research area at the intersection of theoretical computer science, mathematics and economics that deals with optimal strategic behavior in interactive situations. This lecture pays particular attention to the algorithmic aspects of game-theoretic solution concepts such as Nash equilibria and the design of economic mechanisms.

teaching and learning method

The module consists of a lecture and an accompanying exercise. The content of the lecture is conveyed in lectures and presentations. The students are encouraged to deal with the content of the lecture independently by regularly providing exercise sheets. Solutions to the exercises are discussed in the exercise session.

COMPUTATIONAL SOCIAL CHOICE

General competences

Description of study/examination achievements

The examination takes the form of a 120-minute written exam. This consists of several tasks in which the following skills are tested: * Understanding collective decision-making problems and understanding and classifying new concepts * Proving or refuting axiomatic properties of preference bundling procedures with the help of mathematical arguments or counterexamples * Applying algorithms from the lecture to specific questions in the area of collective decision-making No aids are permitted in the exam other than one or more self-made information sheets. The permitted number of information sheets will be announced in good time before the exam. The module grade can be improved by 0.3 by correctly completing exercise sheets or online quizzes over the course of the semester. This grade bonus is only applicable if the exam is passed and is not transferable to subsequent semesters. No repeat exam is offered.

Intended learning outcomes

After successfully completing the module, students will be able to * understand the fundamentals of collective decision-making, * investigate axiomatic properties of preference bundling procedures, * calculate and compare results of different preference bundling procedures and * analyze algorithmic properties of these procedures.

Contents

"Social Choice Theory" deals with methods for collective decision-making. In addition to classic applications such as voting procedures, these methods have been used in various areas of computer science in recent years. The focus of this lecture is on the analysis and comparison of procedures based on the majority relation. In particular, algorithmic aspects of these procedures are considered. Overview of topics: Preferences, voting procedures, choice theory (rationalizability, consistency), May's theorem, Arrow's impossibility theorem, point method, Fishburn's classification of Condorcet procedures, McGarvey's theorem, top cycle, uncovered set, Slater set, Banks set, minimal covering set, tournament equilibrium set, Kemeny-Young method, computational complexity of voting procedures.

"Social Choice Theory" deals with methods for collective decision-making. In addition to classic applications such as voting procedures, these methods have been used in various areas of computer science in recent years. The focus of this lecture is on the analysis and comparison of procedures based on the majority relation. In particular, algorithmic aspects of these procedures are considered. Overview of topics: Preferences, voting procedures, choice theory (rationalizability, consistency), May's theorem, Arrow's impossibility theorem, point method, Fishburn's classification of Condorcet procedures, McGarvey's theorem, top cycle, uncovered set, Slater set, Banks set, minimal covering set, tournament equilibrium set, Kemeny-Young method, computational complexity of voting procedures.

teaching and learning method

The module consists of a lecture and an accompanying exercise. The content of the lecture is conveyed in lectures and presentations. The students are encouraged to deal with the

content of the lecture independently by regularly providing exercise sheets. Solutions to the exercises are discussed in the exercise session.

ADVANCED ALGORITHMICS

General competences

Description of study/examination achievements

The examination takes the form of a 120-minute written exam. In this, students demonstrate, based on the tasks set, that they have fundamental and advanced knowledge in the area of efficient algorithms. The students demonstrate that they can recognize and analyze corresponding algorithmic problems and find ways to an efficient solution within a limited time.

Intended learning outcomes

The participants of the module have in-depth knowledge of various basic principles in the development and analysis of algorithms (divide-and-conquer, greedy methods, dynamic programming, randomization) as well as numerous paradigmatic application problems.

Contents

The module covers basic and classical topics from the area of efficient algorithms. Central techniques of algorithm design and analysis are studied. Specifically, the module covers the methods divide-and-conquer, dynamic programming, randomization, greedy methods and amortized analysis. These techniques are used to solve basic algorithmic problems. The agenda of the module includes the following topics. Divide-and-Conquer: Introduction -- deterministic quicksort; geometric divide-and-conquer; closest pair problem; intersection of line segments; fast Fourier transform (FFT). Introduction to randomized algorithms: Las Vegas and Monte Carlo algorithms; prime number test; RSA cryptographic methods; verification of matrix multiplications. Data structures: Treaps - randomized search trees; universal and perfect hashing; construction of suffix trees. Amortized analysis: Dynamic tables; Fibonacci heaps. Greedy algorithms: Scheduling of time intervals (interval scheduling); planning with the goal of minimizing delays; shortest paths in graphs. Dynamic programming: matrix chain product; construction of optimal search trees; segmentation of data points; edit distance; approximate string matching; sequence alignment. Graph problems: max-flow problem; calculation of minimum cuts. Complexity: PSPACE - a class of problems above NP; relaxed complexity measures. Selected topics: stable marriage; local search; calculation of the median; routing in the hypercube; the probabilistic method.

The module covers basic and classical topics from the area of efficient algorithms. Central techniques of algorithm design and analysis are studied. Specifically, the module covers the methods divide-and-conquer, dynamic programming, randomization, greedy methods and amortized analysis. These techniques are used to solve basic algorithmic problems. The agenda of the module includes the following topics. Divide-and-Conquer: Introduction -- deterministic quicksort; geometric divide-and-conquer; closest pair problem; intersection of line segments; fast Fourier transform (FFT). Introduction to randomized algorithms: Las Vegas and Monte Carlo algorithms; prime number test; RSA cryptographic methods; verification of matrix multiplications. Data structures: Treaps - randomized search trees; universal and perfect hashing; construction of suffix trees. Amortized analysis: Dynamic tables; Fibonacci

heaps. Greedy algorithms: Scheduling of time intervals (interval scheduling); planning with the goal of minimizing delays; shortest paths in graphs. Dynamic programming: matrix chain product; construction of optimal search trees; segmentation of data points; edit distance; approximate string matching; sequence alignment. Graph problems: max-flow problem; calculation of minimum cuts. Complexity: PSPACE - a class of problems above NP; relaxed complexity measures. Selected topics: stable marriage; local search; calculation of the median; routing in the hypercube; the probabilistic method.

teaching and learning method

The module consists of a lecture and an accompanying exercise session. The content of the lecture is conveyed in lectures and presentations. Students are encouraged to engage with the content of the topics, particularly by solving exercise sheets. The solutions to the exercises are discussed in the exercise session. In addition, students receive individual feedback on their learning success by correcting the exercise sheets.

ONLINE AND APPROXIMATION ALGORITHMS

General competences

Description of study/examination achievements

The examination takes the form of a 120-minute written exam. In this, students demonstrate, based on the tasks set, that they have fundamental and advanced knowledge in the area of online and approximation algorithms. Students demonstrate that they can recognize and analyze corresponding algorithmic problems within a limited time and find ways to find an efficient solution.

Intended learning outcomes

The participants of the module know the results of the basic online problems studied, especially in the area of resource management, data structuring and scheduling, as well as the studied NP-hard graph, satisfiability and packing problems. Furthermore, they master the analysis concept of amortization and the algorithm design technique based on LP formulations. They can apply the learned results and concepts to solve new algorithmic problems.

Contents

The module first covers the basics of online and approximation algorithms. The concept of performance guarantees is introduced. In the area of online algorithms, the module studies classical scheduling and paging problems. The concept of amortized analysis is introduced and used to analyze self-organizing lists. The module then examines randomized online algorithms with the associated adversary concept. Data compression algorithms based on self-organizing lists, online algorithms for robot navigation, bipartite matching, energy-efficient scheduling, the k-server problem and financial games are developed. In the area of approximation algorithms, algorithms for max-cut, the traveling salesman problem and load balancing are derived. Furthermore, polynomial approximation schemes are developed, with a focus on the knapsack problem and the load balancing problem. The module also covers the concepts of LP relaxation and randomized rounding. Finally, approximations for set cover and shortest superstring are developed.

The module first covers the basics of online and approximation algorithms. The concept of performance guarantees is introduced. In the area of online algorithms, the module studies classical scheduling and paging problems. The concept of amortized analysis is introduced and used to analyze self-organizing lists. The module then examines randomized online algorithms with the associated adversary concept. Data compression algorithms based on self-organizing lists, online algorithms for robot navigation, bipartite matching, energy-efficient scheduling, the k-server problem and financial games are developed. In the area of approximation algorithms, algorithms for max-cut, the traveling salesman problem and load balancing are derived. Furthermore, polynomial approximation schemes are developed, with a focus on the knapsack problem and the load balancing problem. The module also covers the concepts of LP relaxation and randomized rounding. Finally, approximations for set cover and shortest superstring are developed.

teaching and learning method

The module consists of a lecture and an accompanying exercise session. The content of the lecture is conveyed in lectures and presentations. Students are encouraged to engage with the content of the topics, particularly by solving exercise sheets. The solutions to the exercises are discussed in the exercise session. In addition, students receive individual feedback on their learning success by correcting the exercise sheets.

OPERATIONS RESEARCH

General competences

Description of study/examination achievements

The module examination consists of a 120-minute written exam in which the students demonstrate their ability to apply, analyze and evaluate basic methods from the areas of decision analysis, linear optimization and integer linear optimization by solving problems. On the one hand, practical tasks are modeled as mathematical optimization problems. On the other hand, by answering theoretical questions, the participants show that they have understood the functionality and mathematical properties of various algorithms. Answering the probl Intended learning outcomes

After completing the module, participants will be familiar with basic methods from the areas of decision analysis, linear optimization, and integer linear optimization. They will be able to model operational planning and decision problems such as transport problems or network flow problems and understand algorithms to solve these problems effectively. Students will know the essential properties of these algorithms and relevant business applications.

Contents

Decision analysis (decision trees, MCDM, investment calculation), linear optimization, modeling of operational optimization problems (transportation problem, allocation problem, goal programming), simplex (normal form, basic variables and basic solutions, primal simplex algorithm, simplex tableau, revised simplex), sensitivity analysis and duality theory, integer programming (branch-and-bound, complexity theory, modeling of integer problems), network planning and network flow problems

Decision analysis (decision trees, MCDM, investment calculation), linear optimization, modeling of operational optimization problems (transportation problem, allocation problem, goal programming), simplex (normal form, basic variables and basic solutions, primal simplex algorithm, simplex tableau, revised simplex), sensitivity analysis and duality theory, integer programming (branch-and-bound, complexity theory, modeling of integer problems), network planning and network flow problems

teaching and learning method

The teaching format consists of a lecture and a content-coordinated exercise. In the lecture, the lecturer presents the content and interactively presents application examples from the practice of operational planning. The students are thus familiarized with business planning and decision-making problems and learn to distinguish the applications from corresponding optimization methods. In the exercise, the students work on exercises in supervised individual and group work and evaluate the corresponding quantitative methods. In this way, the students learn to evaluate operational planning and decision-making problems and to constructively criticize their own work.

ANALYSIS OF THREE-DIMENSIONAL SHAPES

General competences

Description of study/examination achievements

At the end of the semester the understanding of the covered topics will be evaluated in an oral exam of 30 minutes. Content from the lecture as well as from the exercises will be part of the exam. Questions allow to assess acquaintance with the foundations of Differential geometry and discretizations of continuous objects and operators. Small programming tasks assess the ability to conceive appropriate algorithmic solutions to analyze and compare three-dimensional shapes.

Intended learning outcomes

At the end of the module, students are familiar with continuous and discrete representations of three-dimensional shapes. Participants understand methods to analyze and compare three-dimensional shapes given in different representations and are able to implement these methods.

Contents

It is a classical problem in Machine Vision to represent, analyze and compare three-dimensional shapes. In the last years this field has known a fast development leading to a number of very powerful algorithms with a solid mathematical foundation. In this course we will present some of these, discussing both, the mathematics involved and the practical issues for the implementation. Topics we plan to cover include: - Foundations of Differential Geometry of surfaces (tangent spaces, shape operators, metric, geodesics and their discrete versions) - Detection of intrinsic symmetries - Matching pairs or a collection of shapes - The Gromov-Hausdorff distance and its variants - Spectral methods (ie Laplace-Beltrami operators and their eigenspaces) - Conformal geometry applied to shape analysis - Shape matching based on continuum mechanics - Pointwise feature descriptors - Machine learning applied to 3D shape analysis

It is a classical problem in Machine Vision to represent, analyze and compare three-dimensional shapes. In the last years this field has known a fast development leading to a number of very powerful algorithms with a solid mathematical foundation. In this course we will present some of these, discussing both, the mathematics involved and the practical issues for the implementation. Topics we plan to cover include: - Foundations of Differential Geometry of surfaces (tangent spaces, shape operators, metric, geodesics and their discrete versions) - Detection of intrinsic symmetries - Matching pairs or a collection of shapes - The Gromov-Hausdorff distance and its variants - Spectral methods (ie Laplace-Beltrami operators and their eigenspaces) - Conformal geometry applied to shape analysis - Shape matching based on continuum mechanics - Pointwise feature descriptors - Machine learning applied to 3D shape analysis

teaching and learning method

Lecture, exercise course (theory and programming)

BASIC MATHEMATICAL METHODS FOR IMAGING AND VISUALIZATION

General competences

Description of study/examination achievements

Type of examination: written exam The examination takes the form of a 75-minute written exam in which students demonstrate, based on the tasks set, that they have knowledge of basic mathematical methods and can successfully apply these to solving simple, abstract mathematical problems. Furthermore, when solving tasks related to specific applications in image processing and computer vision, students demonstrate that they can formulate application problems mathematically, analyze their mathematical properties, and solve them using suitable methods.

Intended learning outcomes

After successfully completing this module, participants will understand the basic mathematical techniques and methods. They will then be able to formulate real-world tasks in the field of imaging and visualization and select, optimize and evaluate methods for problem solving. They can also apply these techniques and methods to other engineering disciplines such as artificial intelligence, machine learning, computer graphics, robotics, etc.

Contents

Basic, frequently used techniques are presented in the lecture and demonstrated using applications from image processing and computer vision. The same mathematical methods are also used in other engineering disciplines such as artificial intelligence, machine learning, computer graphics, robotics, etc. The following content is covered as examples: - Linear algebra ++ Vector spaces and bases ++ Linear mappings and matrices ++ Linear systems of equations, solving linear systems of equations ++ Method of least squares ++ Eigenvalue problems and singular value decomposition - Analysis ++ Metric spaces and topology ++ Convergence, compactness ++ Continuity and differentiability in multidimensionality, Taylor expansion - Optimization ++ Existence and uniqueness of minimizers, identification of minimizers ++ Gradient descent, conjugate gradient ++ Newton's method, fixed point iterations - Probability theory ++ Probability spaces, random variables ++ Expected value and conditional expectation ++ Estimators, expectation maximization method The exercises give participants the opportunity to gain a deeper understanding and practical experience in implementing or applying the methods to solve real problems.

Basic, frequently used techniques are presented in the lecture and demonstrated using applications from image processing and computer vision. The same mathematical methods are also used in other engineering disciplines such as artificial intelligence, machine learning, computer graphics, robotics, etc. The following content is covered as examples: - Linear algebra ++ Vector spaces and bases ++ Linear mappings and matrices ++ Linear systems of equations, solving linear systems of equations ++ Method of least squares ++ Eigenvalue problems and singular value decomposition - Analysis ++ Metric spaces and topology ++ Convergence, compactness ++ Continuity and differentiability in multidimensionality, Taylor expansion - Optimization ++ Existence and uniqueness of minimizers, identification of minimizers ++ Gradient descent, conjugate gradient ++ Newton's method, fixed point iterations - Probability theory ++ Probability spaces, random variables ++ Expected value and conditional expectation ++ Estimators, expectation maximization method The exercises give

participants the opportunity to gain a deeper understanding and practical experience in implementing or applying the methods to solve real problems.

teaching and learning method

The module consists of a lecture and an accompanying exercise. The content of the lecture is conveyed in a lecture and presentation with blackboard writing. Students are encouraged to engage with the topics and their applications in particular by solving exercise sheets. The solutions to the exercises are discussed in the exercise.

REAL-TIME COMPUTER GRAPHICS

General competences

Description of study/examination achievements

The examination takes the form of a 75-minute written exam. This is designed to demonstrate that students are familiar with the basic terms and fundamental concepts and methods in the field of interactive computer graphics. This is tested using knowledge questions and simple transfer tasks.

Intended learning outcomes

Participants acquire in-depth knowledge in the field of game development with a focus on real-time graphics, physical motion simulation and collision detection. After attending the module, participants know the essential components of game engines and understand how they interact. They understand the basic principles of the rasterization-based rendering pipeline, are familiar with simple methods for simulating particle movements and know the basic methods for detecting collisions between dynamic bodies. Students demonstrate the ability to apply the concepts and methods learned and to expand them in a specific application.

Contents

The module accompanies the module IN0039 Real-time computer graphics practical course. The lecture discusses basic algorithms of real-time graphics programming and the simulation of the motion dynamics of rigid bodies. Topics include: polygon models, lighting simulation and shading, texture mapping, rasterization-based graphics pipeline, motion dynamics, particle systems, collision detection and resolution.

The module accompanies the module IN0039 Real-time computer graphics practical course. The lecture discusses basic algorithms of real-time graphics programming and the simulation of the motion dynamics of rigid bodies. Topics include: polygon models, lighting simulation and shading, texture mapping, rasterization-based graphics pipeline, motion dynamics, particle systems, collision detection and resolution.

teaching and learning method

The module consists of a lecture in which the lecturer explains the specific content, points out relevant literature and demonstrates examples on the board or on the computer to provide a deeper understanding of the concepts and methods taught.

INTERNSHIP: REAL-TIME COMPUTER GRAPHICS

General competences

Description of study/examination achievements

Weekly exercise sheets mainly contain programming tasks, which participants submit electronically. Small programming tasks are used to check whether students have understood the basic concepts of graphics programming and can implement basic graphics algorithms in C++ and with current graphics APIs. In order to be able to demonstrate their own contribution to the solutions, participants must be able to present their solutions orally at any time. A total of 8 exercise sheets are issued. For passing the 75-minute exam, a grade bonus of 0.3 (if 31%-60% of the points from the exercises are achieved), 0.6 (if 61%-80% of the points from the exercises are achieved), or 1.0 (if > 80% of the points from the exercises are achieved) can be received.

Intended learning outcomes

Participants will master the basic principles of the C++ programming language and the Direct3D graphics interface. They will be able to develop programs independently, applying key concepts of programming and interactive 3D graphics at a basic, practice-oriented, but scientific level. After attending the module, participants will know the key functionality of Direct3D and will be able to use it to program interactive graphics applications in C++.

Contents

The module accompanies module IN0038. The students develop and program small sample applications under supervision. The students learn the basic concepts of graphics interfaces and graphics programming on graphics cards with Direct3D and C++ as well as the basics of shader programming. The students program in small teams using common version control systems.

The module accompanies module IN0038. The students develop and program small sample applications under supervision. The students learn the basic concepts of graphics interfaces and graphics programming on graphics cards with Direct3D and C++ as well as the basics of shader programming. The students program in small teams using common version control systems.

teaching and learning method

The internship is held in weekly tutor groups. The tutor discusses the solutions to the tasks on the last exercise sheet and gives tips on the tasks on the current exercise sheet. The students then work in small groups on the computer to find the solutions to the current exercises. Since not all tasks can be completed in the tutor group, the students should solve the remaining tasks as homework.

TRACKING AND DETECTION IN COMPUTER VISION

General competences

Description of study/examination achievements

Type of Assessment: written exam (105 minutes) Mid-term exam is planned after 6 lectures and it is written exam. The final exam takes place at the end of the lectures and it also takes the form of written test. Questions allow to assess acquaintance with concepts thought at the lectures and exercises, small calculation tasks and pseudo codes of the algorithmic solutions. Programming tasks are not part of the examinations. For the midterm exam, nothing but your pens and your calculators are allowed. For the final exam, nothing but your pens, your calculators and one DIN A4 page (handwritten, front and back side) with notes are allowed. Bonus points can be earned from the homework projects and the intermediate exam. The final exam brings a maximum of 100 points. You need to have 50 points in order to pass it. Bonus points are irrelevant for the 50-points-hurdle and are added afterwards to your final exam score.

Intended learning outcomes

In the end, the students will have a thorough description of the most important tracking and detection techniques. They should be able to understand and implement those solutions and apply them in reasonably complex problems. The concepts described in this course will be accompanied with brief explanations of the necessary mathematical tools. The participants of this course will be given an important basis to follow the vast and growing Computer Vision literature and use the acquired knowledge to solve new practical problems.

Contents

Computer vision, as a relatively young research area, has emerged as a key discipline in computer science. This is not only evident by a growing and highly competitive research community with a high impact factor in computer science, but also by the emergence of numerous vision companies turning research ideas into a myriad of commercial applications. Besides well-known studies of 3D geometry and camera models, object tracking and detection in images and videos becomes one of the principal research directions of modern Computer Vision. The main objective of this course is to provide students with a gradual introduction to modern tracking, detection and recognition techniques developed in the last years. The course will provide in-depth knowledge of image features, their detection and description, matching techniques, key-point recognition, basic and advanced tracking algorithms based on image features and image intensities, basics of probabilistic and machine learning methods for tracking and object detection. Note that every year the course content is refreshed with new the most promising and potentially the most influential works in the field. The following topics will be handled - Introduction (overview of the course) - Convolution and filtering: -- Basic of image formation -- Convolution and correlation -- Non-linear filtering -- Gaussian Filtering -- Image Derivatives -- Edge Detection - Local invariant feature detectors: -- Harris corner -- Harris Laplace/Affine -- Hessian, Hessian-Laplace/Affine -- FAST - Feature descriptors: -- Difference of Gaussians and SIFT -- Integral images and SURF -- Histogram of Oriented Gradients (HOG) - Keypoint recognition: -- Randomized trees -- FERNS -- Keypoint signatures - Face detection -- Haar features -- Ada-boost -- Viola-Jones Face Detection - Camera models and projections -- Model based tracking -- Pose estimation from 2D-3D coresspondencies (DLT, PnP) -- Rotation

parametrization - Non-linear optimization -- Robust estimators -- RANSAC - Template tracking methods: -- Lucas-Kanade, -- Compositional Alg. -- Inverse Compositional -- ESM -- Linear Predictor - Mean-shift tracking -- mean-shift for pdf estimation -- mean-shift for segmentation -- mean-shift for object tracking -- multi-scale - Template matching approaches -- basic correlation methods (SAD, NCC etc.) -- DOT(Dominant Orientation Template) -- LineMod (LINEarizing the memory multiMODal template matching) - Kalman and particle filtering -- basics Kalman filer -- basics Particle filer -- applications to visual tracking -- applications to camera tracking - Tracking with Dictionary Learning

Computer vision, as a relatively young research area, has emerged as a key discipline in computer science. This is not only evident by a growing and highly competitive research community with a high impact factor in computer science, but also by the emergence of numerous vision companies turning research ideas into a myriad of commercial applications. Besides well-known studies of 3D geometry and camera models, object tracking and detection in images and videos becomes one of the principal research directions of modern Computer Vision. The main objective of this course is to provide students with a gradual introduction to modern tracking, detection and recognition techniques developed in the last years. The course will provide in-depth knowledge of image features, their detection and description, matching techniques, key-point recognition, basic and advanced tracking algorithms based on image features and image intensities, basics of probabilistic and machine learning methods for tracking and object detection. Note that every year the course content is refreshed with new the most promising and potentially the most influential works in the field. The following topics will be handled - Introduction (overview of the course) - Convolution and filtering: -- Basic of image formation -- Convolution and correlation -- Non-linear filtering -- Gaussian Filtering -- Image Derivatives -- Edge Detection - Local invariant feature detectors: -- Harris corner -- Harris Laplace/Affine -- Hessian, Hessian-Laplace/Affine -- FAST - Feature descriptors: -- Difference of Gaussians and SIFT -- Integral images and SURF -- Histogram of Oriented Gradients (HOG) - Keypoint recognition: -- Randomized trees -- FERNS -- Keypoint signatures - Face detection -- Haar features -- Ada-boost -- Viola-Jones Face Detection - Camera models and projections -- Model based tracking -- Pose estimation from 2D-3D coresspondencies (DLT, PnP) -- Rotation parametrization - Non-linear optimization -- Robust estimators -- RANSAC - Template tracking methods: -- Lucas-Kanade, -- Compositional Alg. -- Inverse Compositional -- ESM -- Linear Predictor - Mean-shift tracking -- mean-shift for pdf estimation -- mean-shift for segmentation -- mean-shift for object tracking -- multi-scale - Template matching approaches -- basic correlation methods (SAD, NCC etc.) -- DOT(Dominant Orientation Template) -- LineMod (LINEarizing the memory multiMODal template matching) - Kalman and particle filtering -- basics Kalman filer -- basics Particle filer -- applications to visual tracking -- applications to camera tracking - Tracking with Dictionary Learning

teaching and learning method

Lecture, combined with experimental programming assessment targeting practical implementations of the methods explained at the lectures. The practical programming assessments are first explained at the exercises and then given in the form of home works for students to do them in groups.

USE AND IMPLEMENTATION OF DATABASE SYSTEMS

General competences

Description of study/examination achievements

The examination takes the form of a written exam; the examination lasts 90 minutes. Knowledge questions test familiarity with components of modern database systems; programming tasks test the ability to implement and critically assess advanced algorithms and data structures of the database components; small scenarios with specific architectures and applications, which must be implemented using the methods learned, test the ability to develop specific partial solutions. Knowledge questions test knowledge and characteristics of the various areas of application of database systems. Typical small, specific application scenarios, which must be implemented using the methods learned, test the ability to develop specific partial solutions.

Intended learning outcomes

Participants have a detailed knowledge of the components of modern database systems and the different areas of application of modern database systems. They can implement and critically assess the underlying algorithms and data structures and outline and critically evaluate specific application options in realistic scenarios. They are able to further develop these for different boundary conditions.

Contents

Implementation of database systems - Transaction management - Error handling (recovery) - Multi-user synchronization - Physical data organization - Query processing (logical and physical optimization of queries, cost models) Use of database systems - Distributed databases - Business applications (OLTP, OLAP) - XML and database systems - Performance evaluation - Web services

Implementation of database systems - Transaction management - Error handling (recovery) - Multi-user synchronization - Physical data organization - Query processing (logical and physical optimization of queries, cost models) Use of database systems - Distributed databases - Business applications (OLTP, OLAP) - XML and database systems - Performance evaluation - Web services

teaching and learning method

The module is offered as a lecture with an accompanying exercise. In the lecture, the content is conveyed through clear examples and through discussion with the students. The lecture is also intended to motivate students to independently engage with the topics and to study the literature. In each exercise, worksheets are offered that match the lecture content, which the students can use to independently check and to deepen the methods and concepts learned. After this is initially done with instructions, this is increasingly deepened independently over the course of the semester, individually and sometimes in small groups. Web interfaces for self-study and active testing of SQL, XQuery and Datalog are integrated into the lecture and exercise.

DOCUMENT ENGINEERING AND THE WORLD WIDE WEB

General competences

Description of study/examination achievements

The examination takes the form of a 75-minute written exam. The exam questions test the extent to which the semantic processing of modern digital documents has been mastered. Other questions relate to the ability to make connections to other areas of computer science or to classify the opportunities and risks of this technology.

Intended learning outcomes

Participants are able to semantically evaluate and process modern digital documents in a variety of ways, in the field of tension between text, data and programs. This enables them to accurately assess the modern possibilities of electronic publishing on the web. They have a comprehensive understanding of the basics of web publishing, i.e. concepts, models, architectures, technologies, established procedures and tools relating to the Extensible Markup Language (XML). They correctly assess the state of the art with its strengths and deficits, they can independently work out related technologies and classify them, and they get a realistic picture of future developments. They can establish cross-connections with other computer science disciplines, for example with software engineering via modeling and process models, with formal languages and compiler construction via language definitions and tool development, or with databases via modeling and query languages. They recognize generally valid paradigms of computer science in the context of electronic publishing.

Contents

After an introduction to the topic with a description of the background and classification, the module covers the following topic blocks - Modeling and coding of structured documents Document models (structured documents, semi-structured data) with implementations of XML Information Set and XDM; coding of structured documents with XML, XML namespaces and JSON; coding of text with Unicode. - Structure specifications and their formal foundations Schema languages such as XML DTD, XML Schema, Relax NG, Schematron. - Program-controlled processing of structured documents Addressing of XML fragments with XPath; Transformation of XML documents and preparation for presentation with XSLT, XSL-FO, XHTML; Querying XML documents with XQuery - Architectures for publication applications on the web Data collection and user interfaces with XForms, description of processes with SCXML, architecture XRX

After an introduction to the topic with a description of the background and classification, the module covers the following topic blocks - Modeling and coding of structured documents Document models (structured documents, semi-structured data) with implementations of XML Information Set and XDM; coding of structured documents with XML, XML namespaces and JSON; coding of text with Unicode. - Structure specifications and their formal foundations Schema languages such as XML DTD, XML Schema, Relax NG, Schematron. - Program-controlled processing of structured documents Addressing of XML fragments with XPath; Transformation of XML documents and preparation for presentation with XSLT, XSL-FO, XHTML; Querying XML documents with XQuery - Architectures for publication applications on the web Data collection and user interfaces with XForms, description of processes with SCXML, architecture XRX

teaching and learning method

Lecture with activating elements and integrated practice phases to convey the necessary background information about digital documents and their processing. Self-study tasks and student activities (e.g. designing and leading a practice phase) to deepen practical skills.

DATABASE SYSTEMS ON MODERN CPU ARCHITECTURES

General competences

Description of study/examination achievements

The examination is carried out in the form of a written exam without aids, in which; the examination lasts 90 minutes. Knowledge questions test familiarity with the interaction of database systems/algorithms with modern computer architectures. Transfer tasks test familiarity with the development of internals of database systems for modern computer architectures. Small scenarios test the ability to modify the internals of the database system for the profitable use of modern computer architectures.

Intended learning outcomes

The students - understand the interaction of database systems/algorithms and modern computer architectures (here: CPU, cache, primary memory) and - learn how to develop or modify the internals of database systems in order to profitably use the properties of these computer architectures.

Contents

- Architecture of modern CPUs and the memory hierarchy - Cache memory, TLB, associative memory - Branch prediction in CPUs - Sequential and randomized memory access - Memory access patterns in query evaluation in RDBMS - Tuple storage in relational DBMS (RDBMS) - Vertically fragmented RDBMS and query evaluation - Cache-aware database algorithms (e.g. joins, sorting) - Main memory database systems - Compression to increase data throughput

- Architecture of modern CPUs and the memory hierarchy - Cache memory, TLB, associative memory - Branch prediction in CPUs - Sequential and randomized memory access - Memory access patterns in query evaluation in RDBMS - Tuple storage in relational DBMS (RDBMS) - Vertically fragmented RDBMS and query evaluation - Cache-aware database algorithms (e.g. joins, sorting) - Main memory database systems - Compression to increase data throughput

teaching and learning method

The module is offered as a lecture with an accompanying exercise. In the lecture, the content is conveyed through clear examples and through discussion with the students. The lecture is also intended to motivate students to independently engage with the content of the topics and to study the literature. In each exercise, worksheets are offered that match the lecture content, which the students can use to independently check and to deepen the methods and concepts learned. The work on these is done independently in small groups. These small groups also work on programming tasks, later small projects, which are submitted, commented on and evaluated and discussed in the exercise.

CLOUD INFORMATION SYSTEMS

General competences

Description of study/examination achievements

The assessment is by means of a written exam of 90 minutes assessing knowledge regarding the fundamental concepts of cloud information systems.

Intended learning outcomes

Upon successful completion of this module, students are able to: - understand and analyze public cloud services. - define the requirements and challenges when architecting, implementing and managing cloud-native information systems. - design cost-efficient and scalable cloud information systems using current cloud technologies.

Contents

- Introduction to foundational technologies (virtualization, containers, orchestration) - Architecture of cloud-native information systems (client/server, middleware, microservices) - Cost optimization in public clouds (cost and pricing models) - Economic cloud architecture comparison - Multi-cloud deployments - Implications of SaaS (DevOps, security) - Case studies of public clouds - Case studies of cloud-native information system architectures and their building blocks (eg, cloud object stores, key/value stores, OLTP DBMS, OLAP DBMS)

- Introduction to foundational technologies (virtualization, containers, orchestration) - Architecture of cloud-native information systems (client/server, middleware, microservices) - Cost optimization in public clouds (cost and pricing models) - Economic cloud architecture comparison - Multi-cloud deployments - Implications of SaaS (DevOps, security) - Case studies of public clouds - Case studies of cloud-native information system architectures and their building blocks (eg, cloud object stores, key/value stores, OLTP DBMS, OLAP DBMS)

teaching and learning method

By means of a presentation, either by slides or whiteboard, the lecture presents fundamental concepts of public clouds and the architecture of scalable and cost-efficient cloud information systems. Accompanying assignments for individual study deepen the understanding of the concepts explained in the lecture.

CLOUD-BASED DATA PROCESSING

General competences

Description of study/examination achievements

The exam takes the form of a 90 minute written test or an oral exam in case of low number of participants. Assignments that check whether the student can identify the key requirements for a data processing in the cloud and design a scalable system that can meet them for a given scenario.

Intended learning outcomes

Upon successful completion of this module, students are able to: * define the requirements and challenges when architecting, building and managing a large-scale data processing service in the cloud. * use the theoretical foundations of distributed algorithms to construct the building blocks for a scalable data system design in relation to distributed storage, coordination and computation. * understand and analyze the different trade-offs when designing scalable data processing systems that need to run in the public cloud. * design, implement and evaluate a three-tier system using current cloud technologies. * identify the scalability bottlenecks and vulnerabilities of a complex computer system.

Contents

Introduction * Intro to Data-centers * Latest trends in Cloud Computing Fundamentals for managing distributed data * Data replication and quorums * Data partitioning * Fault-tolerance and unreliable components * Distributed system model * Consensus protocols and coordination services * Consistency models and distributed key value stores Design of cloud-based data processing systems * Distributed transactions * Cloud-native OLTP databases * Cloud-native data warehouses * Dataflow computing (ie, derived data) * Cloud-scale data streaming systems * Query-as-a-Service (QaaS) (serverless data processing) * Resource management and scheduling Other related topics: * Novel data storage formats (eg, data lakes, data-mash, etc.) * Security and Privacy for data processing in the cloud * Accelerators and impact of new hardware technology

Introduction * Intro to Data-centers * Latest trends in Cloud Computing Fundamentals for managing distributed data * Data replication and quorums * Data partitioning * Fault-tolerance and unreliable components * Distributed system model * Consensus protocols and coordination services * Consistency models and distributed key value stores Design of cloud-based data processing systems * Distributed transactions * Cloud-native OLTP databases * Cloud-native data warehouses * Dataflow computing (ie, derived data) * Cloud-scale data streaming systems * Query-as-a-Service (QaaS) (serverless data processing) * Resource management and scheduling Other related topics: * Novel data storage formats (eg, data lakes, data-mash, etc.) * Security and Privacy for data processing in the cloud * Accelerators and impact of new hardware technology

teaching and learning method

Lectures, tutorials, problems for individual study: The module consists of lectures and accompanying tutorials. The contents of the course will be presented primarily in the form of lectures and discussions of real-world system designs. Solutions to exercises will be discussed in the tutorials.

CODE GENERATION FOR DATA PROCESSING

General competences

Description of study/examination achievements

The module examination consists of a written exam (90 minutes) in which students demonstrate that they have understood how code generation works in the context of data processing systems and that they can develop such systems while taking application-specific features into account. The examination primarily requires independently formulated answers and, if necessary, the completion of code segments.

Intended learning outcomes

After successfully completing the module, students will be able to implement an efficient execution environment for data processing programs based on code generation. Students will be able to design suitable program representations for high-level languages, database queries and machine code, translate these into high-performance machine code, and use suitable optimizations and security measures (sandboxing).

Contents

- Basic execution models of programs (interpretation, use of bytecode, machine code generation, etc.) - Various program representations (source code, intermediate representations (IRs), various forms of bytecode) - Classical techniques of code generation - SSA and optimization techniques using LLVM-IR as an example - Machine code generation: instruction selection and register allocation - Execution of programs in virtual machines (e.g. WebAssembly, BPF, JavaScript) - Sandboxing and optimizations for JIT compilation - Execution of database queries (e.g. SQL, Data Frame API) - Execution models and code representations - Execution of machine code/binary translation (e.g. RISC-V) - Special features of machine code translation

- Basic execution models of programs (interpretation, use of bytecode, machine code generation, etc.) - Various program representations (source code, intermediate representations (IRs), various forms of bytecode) - Classical techniques of code generation - SSA and optimization techniques using LLVM-IR as an example - Machine code generation: instruction selection and register allocation - Execution of programs in virtual machines (e.g. WebAssembly, BPF, JavaScript) - Sandboxing and optimizations for JIT compilation - Execution of database queries (e.g. SQL, Data Frame API) - Execution models and code representations - Execution of machine code/binary translation (e.g. RISC-V) - Special features of machine code translation

teaching and learning method

In the lecture, the content is conveyed using concrete examples and discussions with the students. The lecture is also intended to motivate the students to independently engage with the topics and to study the literature. In the accompanying exercises, the students are expected to implement the concepts taught in order to deepen their understanding of the techniques covered.

DATA PROCESSING ON MODERN HARDWARE

General competences

Description of study/examination achievements

The exam takes the form of a 90 minute written test or an oral exam in case of low number of participants. Assignments that assess whether the student can apply the various optimization techniques taught in class on a given data structure/workload set-up or whether the student can correctly identify and explain the performance behavior of a system running on a modern hardware configuration.

Intended learning outcomes

Upon successful completion of this module, students are able to: * implement data processing engines that run faster on modern machines using a variety of techniques: - optimize data structures and algorithms for the memory hierarchy - effectively adjust the program structure of the execution engine and the data pipelines to better exploit modern hardware features (AVX, SGX, - apply custom-parallelization techniques to most common data-processing algorithms - efficiently synchronize data structures * learn how to best leverage new hardware technologies in the context of (distributed) data processing engines: - accelerators (eg, FPGAs, smart NICs, etc.) - low-latency high-bandwidth networks - modern storage and memory (non-volatile, memory extensions, far memory, CXL) * reason about software abstractions and programming models for modern hardware, especially in the context of data-intensive applications. * apply the knowledge in real data management systems: - understand what happens in the background - do performance analysis and debugging of hot spots and bottlenecks - optimize algorithms and data structures to run well both in isolation and alongside other programs (eg, multi-tenancy).

Contents

* Cache hierarchy awareness and how it influences the design of modern data storage formats and indexes. * Memory-, threading- and execution models for data processing on modern machines * Latest practices for query execution, compilation and optimization * Exploiting parallelism in multi-core machines for in-memory data management systems, with particular focus on crafting for non-uniformity in memory access latencies * Building efficient distributed data processing at scale through the use of modern network and storage technologies

* Cache hierarchy awareness and how it influences the design of modern data storage formats and indexes. * Memory-, threading- and execution models for data processing on modern machines * Latest practices for query execution, compilation and optimization * Exploiting parallelism in multi-core machines for in-memory data management systems, with particular focus on crafting for non-uniformity in memory access latencies * Building efficient distributed data processing at scale through the use of modern network and storage technologies

teaching and learning method

Lectures, tutorials, problems for individual study: The module consists of lectures and accompanying tutorials. Solutions to exercises will be discussed in the tutorials. The students

will have an opportunity to apply the knowledge acquired in a group/or individual project at the end of the course.

DATA STRUCTURE ENGINEERING

General competences

Description of study/examination achievements

The goal of the course is to teach skills for understanding and developing efficient data structures. We assess that in a written 90-minute exam that tests the acquired knowledge about modern hardware and data structures discussed in the lecture, as well as the ability to apply this knowledge based on usage scenarios.

Intended learning outcomes

Students can design efficient and compact data structures. They are capable of implementing custom, domain-specific data structure variants and of synchronizing them for multi-core processors in a scalable fashion.

Contents

Data structure engineering is the process of designing and implementing in-memory and out-of-memory data structures to support the efficient storage, organization, and retrieval of data. On modern hardware a low asymptotic complexity alone does not guarantee high performance; one must also take features of today's processors and storage technologies into account. The course first introduces the necessary hardware background, before studying different data structures such as hash tables, search trees, and tries and how to optimize them for modern hardware.

Data structure engineering is the process of designing and implementing in-memory and out-of-memory data structures to support the efficient storage, organization, and retrieval of data. On modern hardware a low asymptotic complexity alone does not guarantee high performance; one must also take features of today's processors and storage technologies into account. The course first introduces the necessary hardware background, before studying different data structures such as hash tables, search trees, and tries and how to optimize them for modern hardware.

teaching and learning method

By means of a presentation, either by slides or whiteboard, the lecture presents existing high-performance data structures, and teaches basic principles and techniques of data structure engineering. Accompanying assignments for individual study deepen the understanding of the concepts explained in the lecture, train students to implement custom data structures.

EVENT PROCESSING

General competences

Description of study/examination achievements

There is a final exam, which verifies the knowledge of theory. Exam duration: 90 minutes.

Intended learning outcomes

Processing paradigms. Apply event processing formalisms, patterns and languages to use cases. Analyze capabilities of emerging standards and products on event processing. Know the background and theory of event processing.

Contents

Introduction to Event Processing (EP) applications. After the module the student knows the common building blocks EP systems and can apply them. The outline of the module is as follows. First, an introduction to EP is given, What EP support is required? What are the characteristics and requirements of EP? Then more theoretical background is given: Event processing terminology, theories of events & philosophical perspectives. Throughout the module multiple distributed, local, and event processing languages are presented which are then used in the course project. Additional in the lecture the following things are presented and discussed: event processing patterns, event generation, event detection, event filtering and event correlation algorithms. To model event-based systems formalisms are presented, eg, event calculus, event algebra or Petrie nets. Additionally covered are event processing paradigms, models for event stream processing, event processing languages, Publish/Subscribe, Tuple spaces, Rule-based event processing, Trigger processing, Continuous query processing. Additional keywords which are presented are EP Languages and data models, window semantics, matching, filtering, Rete matching, staged-event driven architecture and emerging standards & products as case studies. The content is supported by examples and case studies. Also the content is practically used within the course project.

Introduction to Event Processing (EP) applications. After the module the student knows the common building blocks EP systems and can apply them. The outline of the module is as follows. First, an introduction to EP is given, What EP support is required? What are the characteristics and requirements of EP? Then more theoretical background is given: Event processing terminology, theories of events & philosophical perspectives. Throughout the module multiple distributed, local, and event processing languages are presented which are then used in the course project. Additional in the lecture the following things are presented and discussed: event processing patterns, event generation, event detection, event filtering and event correlation algorithms. To model event-based systems formalisms are presented, eg, event calculus, event algebra or Petrie nets. Additionally covered are event processing paradigms, models for event stream processing, event processing languages, Publish/Subscribe, Tuple spaces, Rule-based event processing, Trigger processing, Continuous query processing. Additional keywords which are presented are EP Languages and data models, window semantics, matching, filtering, Rete matching, staged-event driven architecture and emerging standards & products as case studies. The content is supported by examples and case studies. Also the content is practically used within the course project.

teaching and learning method

The module is organized as a combination of lectures slides, invited talks, case studies and required reading material. During the course an event processing system presented in the case studies is used and implemented. At the end of the lecture the implementation is presented and discussed in a discussion advantages and disadvantages of the system. The course project is conducted in teams of two-three students.

HIGH PERFORMANCE QUERY PROCESSING

General competences

Description of study/examination achievements

The goal of the course is to teach skills for understanding and developing high-performance query engines capable of exploiting modern hardware. We assess this in a written 90-minute graded exam that tests the acquired knowledge about modern hardware, data structures, and query processing algorithms discussed in the lecture, as well as the ability to apply this knowledge. No aids or devices are allowed during the exam.

Intended learning outcomes

Students know how modern database engines execute queries. They are capable of implementing salable and efficient algorithms for executing analytical database queries efficiently on multi-core processors and with modern storage technologies.

Contents

Query processing refers to the series of steps that a database system takes to execute an analytical query given a query plan. High performance query processing is about executing analytical queries with performance close to what the hardware allows. To achieve high performance, it is not enough to know about the algorithms that database systems use; one must also take features of today's processors and storage technologies into account. This course covers database query processing paradigms, algorithms, SIMD parallelization, multi-core parallelization, distributed query processing, and out-of-memory processing.

Query processing refers to the series of steps that a database system takes to execute an analytical query given a query plan. High performance query processing is about executing analytical queries with performance close to what the hardware allows. To achieve high performance, it is not enough to know about the algorithms that database systems use; one must also take features of today's processors and storage technologies into account. This course covers database query processing paradigms, algorithms, SIMD parallelization, multi-core parallelization, distributed query processing, and out-of-memory processing.

teaching and learning method

By means of a presentation, either by slides or whiteboard, the lecture presents query processing paradigms, and teaches the engineering required to implement high performance query processing engines. Accompanying assignments for individual study deepen the understanding of the concepts explained in the lecture and train students to implement parts of a concurrent high performance query engine.

Query OPTIMIZATION

General competences

Description of study/examination achievements

The examination takes the form of a written exam; the examination lasts 90 minutes. Knowledge questions test familiarity with the essential optimization techniques of query optimization. Transfer tasks test familiarity with the problem-related development of optimization techniques for query optimization. Small scenarios test the ability to analyze the complexity of specific queries or to evaluate queries in terms of their potential for improvement.

Intended learning outcomes

Participants master the essential optimization techniques of query optimization, can develop them in a problem-related manner, and analyze the complexity of specific queries or evaluate the queries for the purpose of improvement.

Contents

- Query processing in relational database systems - basic techniques of query optimization - determination of the optimal join order - dynamic programming techniques - cost models for query processing - optimization taking physical properties into account

- Query processing in relational database systems - basic techniques of query optimization - determination of the optimal join order - dynamic programming techniques - cost models for query processing - optimization taking physical properties into account

teaching and learning method

The module is offered as a lecture with accompanying exercises. In the lecture, the content is conveyed through clear examples and through discussion with the students. The lecture is also intended to motivate students to independently engage with the topics and to study the literature. In the exercises, worksheets are offered that match the lecture content, which the students can use to independently check and deepen the methods and concepts they have learned. The work is done independently in small groups. These small groups also work on programming tasks, and later small projects, which are submitted, commented on and evaluated.

TRANSACTION SYSTEMS

General competences

Description of study/examination achievements

The exam will be in the form of a written exam; the exam lasts 90 minutes. Knowledge questions test familiarity with the interaction of transaction management and recovery. Transfer tasks and small scenarios test the ability to develop transactions securely (maintaining ACID properties) and yet efficiently (high concurrency). Grade bonus: There will be optional homework. If you achieve 75% or more in the homework, you will receive a bonus of 0.3 on the exam grade (if your exam grade is between 1.3 and 4.0).

Intended learning outcomes

Students - understand the interaction of transaction management and recovery - can develop transactions securely yet efficiently so that ACID transactions with high concurrency can be achieved

Contents

- Concepts of correctness - Algorithms for multi-user synchronization (concurrency control) - Multiversion concurrency control - Concurrency control for objects - Concurrency control for relational databases - Concurrency control for index structures - Implementation of multi-user synchronization - Transaction error handling (recovery) - Crash recovery - Media recovery - Application recovery - Distributed multi-user synchronization - Distributed transaction recovery

- Concepts of correctness - Algorithms for multi-user synchronization (concurrency control) - Multiversion concurrency control - Concurrency control for objects - Concurrency control for relational databases - Concurrency control for index structures - Implementation of multi-user synchronization - Transaction error handling (recovery) - Crash recovery - Media recovery - Application recovery - Distributed multi-user synchronization - Distributed transaction recovery

teaching and learning method

The module is offered as a lecture with an accompanying practice session. In the lecture, the content is conveyed through clear examples and through discussion with the students. The lecture should also motivate the students to independently engage with the content of the topics and to study the literature. Exercise sheets are offered that match the lecture content, which the students should use to independently check and to deepen the methods and concepts learned. The work on these is done independently in small groups. Concrete example scenarios with material from the lecture are also presented, which give rise to questions that the students work on theoretically or in small projects, which are submitted, commented on and evaluated and discussed in the practice session.

IT AND MANAGEMENT CONSULTING

General competences

Description of study/examination achievements

The examination is mainly carried out by working on case studies in groups and presenting them. The results of the group work are documented and evaluated in the form of a presentation of the results of the case study. In addition, one of the prepared case studies is presented to all students (15-25 minutes) and included in the final assessment. To successfully work on the case studies, students must be able to design appropriate solutions to typical problems in the practical implementation of IT consulting. This requires, above all, the ability to assess the fundamental strategic decisions in IT management and their implications.

Intended learning outcomes

After participating in the module, students are able to analyze IT consulting processes in terms of content and their interrelationships. They can evaluate the fundamental strategic decisions in IT management and their implications. Participants understand typical problems in the practical implementation of IT consulting, such as defining an IT strategy or successful IT organization, and can develop appropriate solutions and justify them to others.

Contents

The module gives students an overview of the basic structures and current problems of IT consulting. Concepts such as IT strategy, IT governance, IT organization and processes are presented. The module also introduces students to IT controlling, IT cost and IT value management. All topics are clearly explained using practical case studies.

The module gives students an overview of the basic structures and current problems of IT consulting. Concepts such as IT strategy, IT governance, IT organization and processes are presented. The module also introduces students to IT controlling, IT cost and IT value management. All topics are clearly explained using practical case studies.

teaching and learning method

The teaching format consists of a seminar in which external experts from IT consulting interactively present content and parts of the relevant literature and present practical application examples. The students are thus familiarized with the basic methods of IT consulting and learn to distinguish between their applications. In addition, the participants work in groups to address current problems in the practical implementation of IT consulting and find solutions using case studies. Based on this, presentations are prepared and carried out.

MODELING OF DISTRIBUTED SYSTEMS

General competences

Description of study/examination achievements

90-minute exam that tests the expected learning outcomes and acquired skills. Students demonstrate that they can create formal specifications of concurrent systems and, using examples, that they have knowledge and understanding of essential specification techniques and programming paradigms for concurrent systems. They demonstrate that they can create proofs or refutations of mathematical relationships in these areas and can prepare formal specifications of concurrent systems.

Intended learning outcomes

Participants know the essential basic concepts of methods for specification, modularization, abstraction, refinement and verification of concurrent, distributed, cooperating systems. They know the importance of both the definition of requirements (specification) and the proof of correctness (verification) of implementations. They master fundamental techniques and methods and are able to use a variety of existing modeling and description techniques in a targeted manner and to differentiate them from one another.

Contents

Almost all computers and software systems are distributed and networked or run in some form of network. A large number of these systems are not only distributed and networked among each other, but are also connected to physical and technical processes and therefore have a wide variety of user interfaces. Topics such as concurrency, networking, interaction and coordination of systems are therefore central to the systematic development of modern software systems. The modeling and systematic treatment of such systems brings with it a multitude of questions, phenomena and difficulties that go well beyond the treatment of sequential, non-interactive systems. The lecture presents basic concepts and models for distributed systems as well as ways of describing, specifying, structuring, analyzing and ultimately verifying them. The content of the lecture specifically deals with questions and approaches to modeling distributed systems, the common models and their theories, structures and relationships. The following content is taught: - Concurrent, distributed, cooperating systems: basic concepts and essential properties - Systems as state machines - Functional description of system components - Structural and distribution view - Processes as sequences of distributed systems - Refinement of systems - Message-synchronous systems

Almost all computers and software systems are distributed and networked or run in some form of network. A large number of these systems are not only distributed and networked among each other, but are also connected to physical and technical processes and therefore have a wide variety of user interfaces. Topics such as concurrency, networking, interaction and coordination of systems are therefore central to the systematic development of modern software systems. The modeling and systematic treatment of such systems brings with it a multitude of questions, phenomena and difficulties that go well beyond the treatment of sequential, non-interactive systems. The lecture presents basic concepts and models for distributed systems as well as ways of describing, specifying, structuring, analyzing and ultimately verifying them. The content of the lecture specifically deals with questions and approaches to modeling distributed systems, the common models and their theories,

structures and relationships. The following content is taught: - Concurrent, distributed, cooperating systems: basic concepts and essential properties - Systems as state machines - Functional description of system components - Structural and distribution view - Processes as sequences of distributed systems - Refinement of systems - Message-synchronous systems

teaching and learning method

Lecture, exercise, self-study tasks

PATTERNS IN SOFTWARE ENGINEERING

General competences

Description of study/examination achievements

Type of examination: Written examination The examination is carried out in the form of a written examination lasting 75 minutes. Knowledge questions determine knowledge of pattern schemas, reusability, delegation, inheritance and the applicability of patterns to specific problems. Problem questions determine the ability to solve specific problems by applying patterns and refactorings, to map them to UML models and to implement them in Java. In this way, students demonstrate that they have understood the principles of pattern-oriented modeling and implementation and can apply them.

Intended learning outcomes

After successfully completing this module, students will understand the main categories of patterns, particularly design patterns, architectural patterns, test patterns and organizational patterns. They will understand that patterns are used to describe reusable knowledge in software development that can be used in particular in requirements analysis, system and object design and project management. They will be able to identify the applicability of patterns to given problems, create a corresponding UML model that includes this pattern and translate the model into Java source code.

Contents

The following content is covered as an example in the module IN2081: Patterns in software engineering: -Patterns as a technique for structuring reusable expert knowledge - Categories of patterns: ++ Architectural patterns ++ Design patterns ++ Process patterns ++ Test patterns ++ Organizational patterns ++ Antipatterns - Classification and combination of patterns ++ Pattern-oriented software development - Model transformations - ++ UML model transformations - ++ Forward development and reconstruction of models from code - ++ Code smells - ++ Source code transformations - Use of patterns ++ in analysis ++ in system design ++ in detailed design ++ Identification of patterns in legacy software systems

The following content is covered as an example in the module IN2081: Patterns in software engineering: -Patterns as a technique for structuring reusable expert knowledge - Categories of patterns: ++ Architectural patterns ++ Design patterns ++ Process patterns ++ Test patterns ++ Organizational patterns ++ Antipatterns - Classification and combination of patterns ++ Pattern-oriented software development - Model transformations - ++ UML model transformations - ++ Forward development and reconstruction of models from code - ++ Code smells - ++ Source code transformations - Use of patterns ++ in analysis ++ in system design ++ in detailed design ++ Identification of patterns in legacy software systems

teaching and learning method

Blended learning: Combining knowledge transfer and problem solving: Each pattern is explained and then practiced using specific problems during the lecture. There is also homework for self-study.

PROJECT ORGANIZATION AND MANAGEMENT IN SOFTWARE ENGINEERING

General competences

Description of study/examination achievements

Type of examination: Written examination The examination is carried out in the form of a written examination lasting 90 minutes. Knowledge questions determine the students' knowledge of typical management problems in software projects. Problem questions determine the ability to complete management tasks, such as sketching out a software plan, a configuration management plan, a branching model or a release plan. In this way, students demonstrate that they have understood the principles of software project management and organization and can apply them.

Intended learning outcomes

After successfully completing this module, students will understand the essential concepts of software project management. They will be able to perform typical tasks such as creating a software project management plan, initiating and implementing a software project, and adapting a software lifecycle model. They will also be familiar with the key issues of risk management, scheduling, quality management, and build and delivery management, and will be able to apply these to smaller problems.

Contents

The following content is covered as an example in the IN2083 module: - Project organization ++ Life cycle model of a project ++ Organizations (line, matrix and project-based) ++ Roles and responsibilities, management anti-patterns ++ Global software projects ++ Maturity levels of organizations (CMM, CMMI) - Project planning ++ Software life cycle models ++ Linear models (waterfall, V-model) ++ Iterative models (spiral model, unified process, V-model XT) ++ Agile models (Scrum, Kanban) ++ Estimation ++ Project scheduling - Contracts ++ Types of contracts ++ Life cycle model for contracts ++ Legal issues - Change management ++ Configuration management ++ Continuous integration ++ Continuous delivery - Quality management ++ User management ++ Tests, V&V ++ Risk management ++ IEEE standards

The following content is covered as an example in the IN2083 module: - Project organization ++ Life cycle model of a project ++ Organizations (line, matrix and project-based) ++ Roles and responsibilities, management anti-patterns ++ Global software projects ++ Maturity levels of organizations (CMM, CMMI) - Project planning ++ Software life cycle models ++ Linear models (waterfall, V-model) ++ Iterative models (spiral model, unified process, V-model XT) ++ Agile models (Scrum, Kanban) ++ Estimation ++ Project scheduling - Contracts ++ Types of contracts ++ Life cycle model for contracts ++ Legal issues - Change management ++ Configuration management ++ Continuous integration ++ Continuous delivery - Quality management ++ User management ++ Tests, V&V ++ Risk management ++ IEEE standards

teaching and learning method

Lecture, tutorial exercise, interactive exercises during the lecture, self-study tasks

ADVANCED TOPICS IN SOFTWARE TESTING

General competences

Description of study/examination achievements

The examination takes the form of a 75-minute written exam. This is intended to demonstrate that students can critically reflect on the various activities of the test and apply them in an exemplary manner, for example by abstractly defining test cases for exemplary systems.

Intended learning outcomes

After participating in the module, students are able to name the various activities and artifacts of software testing and evaluate them in terms of their strengths and weaknesses. In given contexts, they can construct and analyze test cases for systems both conceptually and by example and also concretely and justify why certain test cases are useful. They can understand, justify and criticize various test-related management decisions regarding process and tool support.

Contents

Basics - Introduction - Terms and motivation - Basic test process - Prioritization of tests - Psychology of testing Testing in the software life cycle - Waterfall / V / W / Incremental / Spiral model - Module / component testing - Integration testing - System testing - Acceptance testing - Maintenance of tests - Testing of prototypes Static testing - Manual test methods - Static analysis Dynamic testing - Black box method - White box method - Intuitive test case determination Test management - Risk management - Economic efficiency of tests - Reuse - Error management - Test planning / monitoring / control - Metrics - Organization of test teams / qualifications - Requirements for configuration management - Norms and standards Test tools / test automation - Types - Selection - Introduction of tools - "Home built" vs. commercial - Presentation of tools This module can also be completed in combination with the module IN2114 "Automotive Software - Methods and Technologies".

Basics - Introduction - Terms and motivation - Basic test process - Prioritization of tests - Psychology of testing Testing in the software life cycle - Waterfall / V / W / Incremental / Spiral model - Module / component testing - Integration testing - System testing - Acceptance testing - Maintenance of tests - Testing of prototypes Static testing - Manual test methods - Static analysis Dynamic testing - Black box method - White box method - Intuitive test case determination Test management - Risk management - Economic efficiency of tests - Reuse - Error management - Test planning / monitoring / control - Metrics - Organization of test teams / qualifications - Requirements for configuration management - Norms and standards Test tools / test automation - Types - Selection - Introduction of tools - "Home built" vs. commercial - Presentation of tools This module can also be completed in combination with the module IN2114 "Automotive Software - Methods and Technologies".

teaching and learning method

The module consists of a lecture and an accompanying exercise. The content of the lecture is conveyed in lectures and presentations. Students are encouraged to study the literature and to engage with the topics in terms of content. In the exercises, specific questions are answered together and selected examples are worked on.

AUTOMOTIVE SOFTWARE - METHODS AND TECHNOLOGIES

General competences

Description of study/examination achievements

A 75-minute exam that tests the expected learning outcomes and acquired skills. The aim is to demonstrate knowledge and understanding of the different types of automotive software and their characteristics, as well as the consequences for development and testing. The aim is also to demonstrate knowledge and understanding of relevant development processes and tools, as well as the ability to critically classify them.

Intended learning outcomes

After participating in the module, students are able to develop simple embedded systems in the automotive domain. They can differentiate between the various application areas of software in automobiles and understand the differences and consequences for development. They know the key components of relevant development processes and typical tools and can evaluate them.

Contents

The strategic importance of software in vehicles is increasing rapidly. The lecture will introduce three different software domains in vehicles: infotainment, body/comfort, driver assistance and safety electronics. The requirements and special features as well as the theoretical basics for each domain will be explained in a lecture. After the three software domains, three cross-sectional topics are addressed centrally: Development process, architecture and quality assurance - Introduction and overview "Software and software engineering in the automotive context" - "Embedded systems" application to body and comfort electronics - "Real-time and control technology" application in safety electronics and chassis control - "Embedded Java" application in infotainment - "Sensors / actuators" application to driver assistance systems - "Development process I" requirements, logical architecture - "Development process II" from the logical architecture to the integrated system - Model-based development: embedded middleware - Tools and execution models: ASCET, Rose RT, MATLAB - Testing and quality assurance - Software in the product life cycle - Challenges of the future In the exercise, 3 special programming tasks must be solved. The lecture can be attended in parallel with IN2084 "Processes and methods in testing software".

The strategic importance of software in vehicles is increasing rapidly. The lecture will introduce three different software domains in vehicles: infotainment, body/comfort, driver assistance and safety electronics. The requirements and special features as well as the theoretical basics for each domain will be explained in a lecture. After the three software domains, three cross-sectional topics are addressed centrally: Development process, architecture and quality assurance - Introduction and overview "Software and software engineering in the automotive context" - "Embedded systems" application to body and comfort electronics - "Real-time and control technology" application in safety electronics and chassis control - "Embedded Java" application in infotainment - "Sensors / actuators" application to driver assistance systems - "Development process I" requirements, logical architecture - "Development process II" from the logical architecture to the integrated system - Model-based development: embedded middleware - Tools and execution models: ASCET, Rose RT, MATLAB - Testing and quality assurance - Software in the product life cycle -

Challenges of the future In the exercise, 3 special programming tasks must be solved. The lecture can be attended in parallel with IN2084 "Processes and methods in testing software".

teaching and learning method

Lecture; practical exercises on the development of embedded systems in the field of automotive software.

ADVANCED CONCEPTS OF PROGRAMMING LANGUAGES

General competences

Description of Achievement and Assessment Methods

The assessment is by means of a written exam of 90 minutes. Individual assignments assess in how far students are able to reproduce the complex semantical behaviors of small example programs. Their knowledge and practical skills concerning programming constructs is further assessed by assignments, which ask to simulate programming language constructs of one kind by programming language constructs of another kind. It is permitted to prepare a double-sided A4 sheet of paper for the written exam and bring it to the exam.

Intended Learning Outcomes

After successful completion of this module, students are familiar with an assortment of programming language constructs from popular programming languages. They understand the semantics of these constructs as well as the implementation consequences, that they inflict on the implementation as well as the runtime behavior of compiler and runtime system. They are able to analyze and compare different language based approaches, to discuss their relative merits and potential workarounds in case certain language features are missing. By means of this knowledge, they are able to extend existing compilers to handle the aforementioned constructs as well as able to re-encode concepts from one language by means of another language.

Content

This is not a beginners programming course – Instead, this lecture focusses on the semantics of various programming language concepts, and their implementation from the point of view of a compiler engineer. Topics may include, among others: - Relaxed Memory Models - Wait-/Lock-free Programming - Locks, Monitors & Condition Variables - Transactional Memory - Gotos, Co-Routines and Continuations - Single/Multi-Dispatching - (Multiple-) Inheritance - Delegation and Prototype Based Programming - Aspect Oriented Programming - Meta programming

This is not a beginners programming course – Instead, this lecture focusses on the semantics of various programming language concepts, and their implementation from the point of view of a compiler engineer. Topics may include, among others: - Relaxed Memory Models - Wait-/Lock-free Programming - Locks, Monitors & Condition Variables - Transactional Memory - Gotos, Co-Routines and Continuations - Single/Multi-Dispatching - (Multiple-) Inheritance - Delegation and Prototype Based Programming - Aspect Oriented Programming - Meta programming

Teaching and Learning Methods

By means of pre-recorded lesson videos of around 15 minutes each, students can prepare the lecture content at their own pace. In the classroom, students can open discussion on unclear parts of the lesson videos. Additionally, illustrating examples and live programming enhance and deepen the student’s insights into the topics. Selected problems that are then solved by the joined effort of the audience and the lecturer further illustrate the lessons with hands-on experiences. In the additionally offered exercise course (2h), accompanying assignments for individual study deepen the understanding of the concepts explained in the

lecture, train students to apply the learnt concepts in implementations and develop the skill to to simulate the effect of missing language features by others.

AUTOMATA AND FORMAL LANGUAGES

General competences

Description of study/examination achievements

Students are assessed by means of a written 120 minute exam consisting of a list of exercises. Some exercises test if the students are able to construct finite automata for different languages, directly or with the help of composition operations. Other exercises test if the student knows and can apply and combine the algorithms to move between logical and automata-theoretic descriptions. Other exercises test if students can select the right automata-theoretic technique to solve problems related to verification and pattern-matching.

Intended learning outcomes

On successful completion of the module, students will be able to - use finite automata as a data structure for representation of finite and infinite sets; - understand and determine the computational complexity of different operations for different classes of automata; - move to and fro logical and automata-theoretic descriptions; - apply automata to problems in pattern matching and formal verification.

Contents

The module is divided into two parts. The first part deepens and expands the study of finite automata initiated in IN0011 (Introduction to theoretical computer science), while the second introduces automata on infinite words. In both parts automata are seen as a data structure for the manipulation of (possibly infinite) sets and relationships. The module shows how to implement Boolean operations and joins for different automata classes (nondeterministic and deterministic automata, binary decision diagrams, Büchi automata). It also introduces the connection between automata and logic. The algorithms are applied to a variety of problems, ranging from pattern-matching to program verification and solution of Diophantine equations.

The module is divided into two parts. The first part deepens and expands the study of finite automata initiated in IN0011 (Introduction to theoretical computer science), while the second introduces automata on infinite words. In both parts automata are seen as a data structure for the manipulation of (possibly infinite) sets and relationships. The module shows how to implement Boolean operations and joins for different automata classes (nondeterministic and deterministic automata, binary decision diagrams, Büchi automata). It also introduces the connection between automata and logic. The algorithms are applied to a variety of problems, ranging from pattern-matching to program verification and solution of Diophantine equations.

teaching and learning method

The module consists of lectures and tutorials. During the lectures students are asked to solve small exercises online. Students also received weekly assignments, whose solution is discussed in the tutorials.

COMPILER CONSTRUCTION I

General competences

Description of study/examination achievements

The examination takes the form of a 90-minute written exam. The individual tasks test the knowledge and mastery of the techniques required for the different components of a compiler. The tasks relating to the frontend ensure that the basic concepts of finite automata and context-free grammars are mastered. This could require students to apply the algorithms learned to example automata or grammars, or to derive simple properties or assertions in general. The tasks relating to the middleend test the extent to which students have understood the concepts learned in this area, such as types and type systems, and can apply them to small program examples. The tasks relating to the backend, i.e. code generation, test the extent to which students are able to generate code for a simple target machine using the translation schemes and algorithms learned for small example programs. Students are allowed to make a double-sided A4 sheet of paper and bring it to the exam. A successful 15-minute presentation of a selected advanced topic from the lecture can be included in the exam assessment as a grade bonus of 0.3. The exact rules for this will be announced in good time at the start of the module.

Intended learning outcomes

After successfully completing this module, students will have mastered the basic concepts and techniques of compiler construction. They will understand practically relevant procedures that are used in the three components of a compiler: the frontend, the middleend and the backend. With their help, they will be able to write a compiler that generates code for a simple programming language and a simple target architecture.

Contents

A compiler is an essential component of system software, whose task is to translate programs from a higher programming language - such as C or Java - into sequences of machine instructions on a real computer. Compilers are relatively complex programs; ideas and methods from many areas are used in their construction. Topics of the lecture include: - Lexical analysis; - Syntactic analysis; - Type checking for imperative programming languages; - Type inference for functional programming languages; - Code generation for an abstract machine.

A compiler is an essential component of system software, whose task is to translate programs from a higher programming language - such as C or Java - into sequences of machine instructions on a real computer. Compilers are relatively complex programs; ideas and methods from many areas are used in their construction. Topics of the lecture include: - Lexical analysis; - Syntactic analysis; - Type checking for imperative programming languages; - Type inference for functional programming languages; - Code generation for an abstract machine.

teaching and learning method

The lecture introduces the basic concepts and techniques of compiler construction and explains them using examples. In the accompanying exercises, the understanding of the content of the lecture is deepened using suitable tasks and the application of the various

techniques to manageable problems from the various components of a compiler is practiced. - The lecture is held in flipped classroom mode. - Lecture videos are created by the students in advance. - There will be a weekly on-site meeting (inverted classroom sessions) for mini quizzes, questions and answers, tutorials, demonstrations and other practice examples.

FUNCTIONAL DATA STRUCTURES

General competences

Description of study/examination achievements

The exam takes the form of a written test of 120 minutes or an oral exam of 30 minutes. The exam will assess students' fundamental understanding of functional data structures. This will involve both programming tasks and proofs and will test the students' ability to use that data structures covered in the lectures and to design new data structures and algorithms based on the methods covered in the module.

Intended learning outcomes

After the successful completion of this module, students will have an in-depth understanding of a range of functional data structures and are able to prove properties about them with the help of a computer-based proof assistant. Students will be able to - select suitable implementations of functional data structures for common programming tasks, - design and implement new functional data structures that are based on the principles covered in the module, - prove the correctness and complexity of the data structures on paper and with the help of a proof assistant.

Contents

The module introduces students to the design and analysis of data structures for functional programming languages. It assumes that the students are familiar with functional programming and with running time analysis of algorithms. The module covers a range of functional data structures with an emphasis on their precise analysis. Proofs of both functional correctness and complexity will be a core part of the module Proofs are carried out with the help of a computer-based proof assistant. An introduction to the proof assistant is part of the module. A selection of the following non-exhaustive list of data structures and algorithms will be covered: - Sorting - Search trees: Unbalanced trees, AVL trees, Red-Black trees, 2-3 trees, Weight balanced trees, Splay trees, Tries - Other trees: Quad trees, Finger trees - Huffman coding - Priority queues: Braun trees, Leftist heaps, Skew binomial heaps, Skew heaps, Pairing heaps, Fibonacci heaps - Dijkstra's algorithm - String matching

The module introduces students to the design and analysis of data structures for functional programming languages. It assumes that the students are familiar with functional programming and with running time analysis of algorithms. The module covers a range of functional data structures with an emphasis on their precise analysis. Proofs of both functional correctness and complexity will be a core part of the module Proofs are carried out with the help of a computer-based proof assistant. An introduction to the proof assistant is part of the module. A selection of the following non-exhaustive list of data structures and algorithms will be covered: - Sorting - Search trees: Unbalanced trees, AVL trees, Red-Black trees, 2-3 trees, Weight balanced trees, Splay trees, Tries - Other trees: Quad trees, Finger trees - Huffman coding - Priority queues: Braun trees, Leftist heaps, Skew binomial heaps, Skew heaps, Pairing heaps, Fibonacci heaps - Dijkstra's algorithm - String matching

teaching and learning method

The module consists of lectures and tutorials. In the lectures, the material is presented by the teacher, in dialogue with the students. During the tutorials, the students work on given

exercises either individually or in small groups with help from the tutors. Exercises can be computer-based as well as paper and pencil based.

GAMES ON GRAPHS

General competences

Description of study/examination achievements

Type of examination: 90-minute written examination or 20-minute oral examination The examination takes the form of an oral examination (individual interview) or written examination (written examination). The examination format depends on the number of participants. In the examination itself, the students give definitions and solution methods. Using small problem examples, the students demonstrate that they can also apply the solution methods independently. They also answer comprehension tasks on the concepts covered in the lecture.

Intended learning outcomes

After successfully completing this module, students will know the basic concepts of game theory. They will be able to apply the relevant algorithmic solution methods and thus solve smaller game instances by hand. They will also understand the relevant mathematical tools for general solution of the games discussed.

Contents

The lecture introduces the theory of games on (finite) graphs (recursive games with a finite state space). Using different variants of reachability games (with/without chance, alternating/simultaneous decisions), the basic definitions and procedures from the field of mathematical game theory are presented. Building on this, more complex quantitative (e.g. Markov decision processes, discounted-payoff games, mean-payoff games, Shapely's stochastic games) and qualitative (e.g. parity, Rabin, Muller, Streett) variants are discussed. Further topics include techniques for reducing memory requirements (e.g. BDDs), corresponding software tools (e.g. PRISM), games with partial observations, games on recursive graphs, references to logic and set theory (e.g. Gale-Stewart games and determinism according to Martin) or the synthesis of control programs.

The lecture introduces the theory of games on (finite) graphs (recursive games with a finite state space). Using different variants of reachability games (with/without chance, alternating/simultaneous decisions), the basic definitions and procedures from the field of mathematical game theory are presented. Building on this, more complex quantitative (e.g. Markov decision processes, discounted-payoff games, mean-payoff games, Shapely's stochastic games) and qualitative (e.g. parity, Rabin, Muller, Streett) variants are discussed. Further topics include techniques for reducing memory requirements (e.g. BDDs), corresponding software tools (e.g. PRISM), games with partial observations, games on recursive graphs, references to logic and set theory (e.g. Gale-Stewart games and determinism according to Martin) or the synthesis of control programs.

teaching and learning method

Lecture, exercise, self-study tasks

LOGIC

General competences

Description of study/examination achievements

In the written 120 minute exam, questions are designed to assess the student's grasp of the relationships between syntax, semantics and proof systems, both in terms of abstract concepts and in terms of small concrete proofs in deductive systems or with the help of algorithmic decision or semi-decision procedures like resolution. They will also be required to prove or apply basic meta-theoretic results characterizing particular logical systems.

Intended learning outcomes

The students understand the fundamental difference between syntax and semantics. In particular, they are able to perform proofs in a deductive system and can prove the soundness and completeness of that system wrt a semantics. They understand the limitations of logic and the relationship between logic and computability. In particular they can prove the indecidability of some basic problem like validity of formulas. They can formalize problems in informatics or mathematics in logic and can solve them algorithmically using decision or semi-decision procedures like resolution.

Contents

In the context of propositional and predicate logic the following basic notions are covered: - Syntax and semantics - Proof system and proof search - Soundness and completeness - Decidability - Expressiveness Possible further topics (non exhaustive): - Proof systems - Automated theorem proving - Verification - Decision procedures - Prologue - Data base theory - Higher-order logic - Set theory - Constructive logic - Fuzzy logic.

In the context of propositional and predicate logic the following basic notions are covered: - Syntax and semantics - Proof system and proof search - Soundness and completeness - Decidability - Expressiveness Possible further topics (non exhaustive): - Proof systems - Automated theorem proving - Verification - Decision procedures - Prologue - Data base theory - Higher-order logic - Set theory - Constructive logic - Fuzzy logic.

teaching and learning method

The module consists of lectures and tutorials. In the lectures, the material is presented by the teacher, in dialogue with the students. During the tutorials, the students work on given exercises either individually or in small groups with help from the tutors. Exercises are primarily pen and paper based but may also involve computer-based components.

MODEL CHECKING

General competences

Description of study/examination achievements

Students are assessed by means of a written exam (120 minutes) consisting of a list of exercises. Some exercises test the ability of the student to model small toy examples of hardware and software systems. Other exercises test if the student is able to formalize specifications described in natural language. Further exercises test if the students can select the most adequate technique for the state-explosion problem in a given system and apply it.

Intended learning outcomes

Successful participation enables students to - explain the role of model checking for ensuring hardware and software quality - give formal models of simple hardware and software systems - write formal specifications of simple safety and liveness properties using temporal logic - explain algorithms for checking if a program satisfies a given specification and apply them to small examples - understand and explain techniques for palliating the state-explosion problem

Contents

The course teaches the fundamentals of Model Chekcing. Topics include: - Representation of hardware and software systems using formal models of computations, eg, transition systems, pushdown systems, Kripke structures, finite and infinite state automata. - Representation of specifications using formal logics and automata, eg, linear and branching time temporal logics, automata on finite and infinite. - Decidability and complexity of various classes of formal models and specifications, eg, complexity of finite state model checking for linear and branching time logics. - Techniques for palliating the state-explosion problem eg, binary decision diagrams, partial-order reduction, abstraction. - Industrial application of model checking, eg, discovery of concurrency bugs, verification of systems software. - Practical experience in applying model checking tools, eg, Spin, Blast, CBMC.

The course teaches the fundamentals of Model Chekcing. Topics include: - Representation of hardware and software systems using formal models of computations, eg, transition systems, pushdown systems, Kripke structures, finite and infinite state automata. - Representation of specifications using formal logics and automata, eg, linear and branching time temporal logics, automata on finite and infinite. - Decidability and complexity of various classes of formal models and specifications, eg, complexity of finite state model checking for linear and branching time logics. - Techniques for palliating the state-explosion problem eg, binary decision diagrams, partial-order reduction, abstraction. - Industrial application of model checking, eg, discovery of concurrency bugs, verification of systems software. - Practical experience in applying model checking tools, eg, Spin, Blast, CBMC.

teaching and learning method

The module consists of lectures and tutorials. During the lectures students are asked to solve small exercises online. Students also received weekly assignments, whose solution is discussed in the tutorials.

NETWORK ANALYSIS - STATISTICAL AND FORMAL MODELS AND METHODS

General competences

Description of study/examination achievements

The examination takes the form of a 75-minute written exam. Knowledge questions test familiarity with modeling concepts and simulation methods. Using tasks with concrete case studies, students demonstrate that they know and can apply selected techniques for modeling, simulation and result evaluation.

Intended learning outcomes

Participants master methodological fundamentals and practical techniques of modeling, simulation and result evaluation and are familiar with selected case studies.

Contents

- Modeling of systems - Concepts of discrete-time simulation - Components of simulation models and simulation software - Random number generation and distribution functions - Time and performance behavior of systems - Performance metrics - Simulation experiments - Case studies

- Modeling of systems - Concepts of discrete-time simulation - Components of simulation models and simulation software - Random number generation and distribution functions - Time and performance behavior of systems - Performance metrics - Simulation experiments - Case studies

teaching and learning method

Lecture, exercise, self-study tasks

PETRI NETS

General competences

Description of study/examination achievements

The assessment takes the form of a 75-minute written exam in which students demonstrate that they can solve three types of tasks. Modeling tasks test students' ability to design simple Petri nets that have a given functionality or satisfy a certain set of properties. Analysis tasks test whether students can apply the techniques taught in the course to decide whether a given Petri net satisfies a property. Comprehension tasks test whether students have understood the underlying principle of a technique, typically by designing a variant of it or deciding whether a given variant is correct.

Intended learning outcomes

After successfully completing this module, students will be able to - model simple concurrent systems using Petri nets; - analyze small Petri nets by hand; - select automatic analysis techniques and software tools for analyzing larger nets; - use the analysis results to evaluate the behavior of concurrent systems.

Contents

Petri nets are a formal model for describing and analyzing concurrent systems such as distributed algorithms, communication protocols, manufacturing systems, business processes, or asynchronous circuits. The strengths of Petri nets are a simple, clear notation with precise semantics and a wealth of existing techniques for analysis and verification. The module is an introduction to the theory of Petri nets. It presents several variants of Petri nets and their most relevant analysis techniques: reachability and coverability graphs, linear invariants, structure theory, and unfoldings. The techniques are also illustrated by software tools.

Petri nets are a formal model for describing and analyzing concurrent systems such as distributed algorithms, communication protocols, manufacturing systems, business processes, or asynchronous circuits. The strengths of Petri nets are a simple, clear notation with precise semantics and a wealth of existing techniques for analysis and verification. The module is an introduction to the theory of Petri nets. It presents several variants of Petri nets and their most relevant analysis techniques: reachability and coverability graphs, linear invariants, structure theory, and unfoldings. The techniques are also illustrated by software tools.

teaching and learning method

The module consists of a lecture and an accompanying exercise. The content of the lecture is conveyed in the lecture and through presentations. Students are encouraged to engage with the content of the topics through small tasks set during the lectures and by solving exercise sheets. The solutions to the exercises are discussed in the exercise.

PROGRAM OPTIMIZATION

General competences

Description of study/examination achievements

The examination takes the form of a 120-minute written exam. Individual tasks use small problems from the area of lattice theory to test the extent to which students are familiar with the basic concepts of program analysis and can apply them independently. Other tasks enable students to demonstrate that they can apply the analyses and transformations they have learned to small example programs. In further tasks, students are asked to develop new analyses or optimizations themselves. Successful completion of the homework can be included as a bonus in the assessment of the exam. The exact rules for this will be announced in good time at the start of the module.

Intended learning outcomes

Participants are aware of the frequently occurring contradiction between good structure and readability of a program and the equally desired greatest possible efficiency in program execution. They are proficient in the techniques that can be used in the optimization phase of a compiler to increase the efficiency of program execution through analysis and, if necessary, optimizing transformations. They can apply these to given small programs and, if necessary, develop new analyses and optimizing transformations.

Contents

The lecture begins with basic data flow analyses such as availability of expressions and liveness of variables together with the optimizing transformations made possible by this information. Motivated by these examples, the necessary theory of complete lattices and monotone functions is developed, which is required to calculate the program analyses. We then look at more elaborate analyses, such as constant propagation. With interval analysis, we discuss methods that are also applicable in the case when the usual fixed point iteration does not terminate. Other topics are: - interprocedural analysis; - pointer analyses; - fixed point algorithms. We also look at hardware-specific optimization techniques such as: - register distribution; - instruction selection; - instruction scheduling as well as techniques for better utilization of caches.

The lecture begins with basic data flow analyses such as availability of expressions and liveness of variables together with the optimizing transformations made possible by this information. Motivated by these examples, the necessary theory of complete lattices and monotone functions is developed, which is required to calculate the program analyses. We then look at more elaborate analyses, such as constant propagation. With interval analysis, we discuss methods that are also applicable in the case when the usual fixed point iteration does not terminate. Other topics are: - interprocedural analysis; - pointer analyses; - fixed point algorithms. We also look at hardware-specific optimization techniques such as: - register distribution; - instruction selection; - instruction scheduling as well as techniques for better utilization of caches.

teaching and learning method

Using a slide or blackboard presentation, the lecture introduces the basic techniques for analyzing and optimizing programs and explains them using examples. In the accompanying

exercises, the understanding of the content of the lecture is deepened using suitable tasks, the application of the learned analyses and optimizations to small programs is practiced and the ability to carry out your own analyses and optimizations is developed.

QUANTITATIVE VERIFICATION

General competences

Description of study/examination achievements

Students are assessed by means of a written exam consisting of a list of exercises. Some exercises test the ability of the student to model small toy examples of hardware and software systems. Other exercises test if the student is able to formalize specifications described in natural language. Further exercises test if the students can select the most adequate technique for the analysis of a given system and apply it. Type of Assessment: written Duration of Assessment: 120 min Homework:N Term Paper:N Oral Presentation:N Conversation:N

Intended learning outcomes

Successful participation enables students to • explain the role of quantitative modeling and analysis for ensuring hardware and software quality. • give formal models of simple hardware and software systems describing their probabilistic and timed behavior. • write formal specifications of simple safety, dependability, reliability, and performance properties. • explain algorithms for checking if a system satisfies a given specification and apply them to small examples. • understand and explain techniques for improving practical performance of the analysis algorithms.

Contents

The course teaches the fundamentals of modeling and analysis of systems featuring quantitative behavior, such as time and probabilities. Topics include: • Representation of hardware and software systems using formal models of computations, eg, transition systems, discrete-time and continuous-time Markov chains, Markov decision processes, stochastic games, timed automata. • Representation of specifications using formal logics, eg, probabilistic linear temporal logic, probabilistic computation-tree logic, continuous stochastic logic, metric temporal logic. • Algorithms for various classes of formal models and specifications, EG, transient analysis, stationary analysis, mean-payoff optimization, reachability-game solving. • Theoretically optimal as well as practical techniques for analysis of large systems, eg, linear programming, value iteration, strategy iteration, reinforcement learning. • Industrial application of the analysis, eg, randomized communication protocols, embedded systems, cyber-physical systems, safety-critical missions. • Practical experience in applying model checking tools, eg, PRISM, MRMC, UPPAAL.

The course teaches the fundamentals of modeling and analysis of systems featuring quantitative behavior, such as time and probabilities. Topics include: • Representation of hardware and software systems using formal models of computations, eg, transition systems, discrete-time and continuous-time Markov chains, Markov decision processes, stochastic games, timed automata. • Representation of specifications using formal logics, eg, probabilistic linear temporal logic, probabilistic computation-tree logic, continuous stochastic logic, metric temporal logic. • Algorithms for various classes of formal models and specifications, EG, transient analysis, stationary analysis, mean-payoff optimization, reachability-game solving. • Theoretically optimal as well as practical techniques for analysis of large systems, eg, linear programming, value iteration, strategy iteration, reinforcement learning. • Industrial application of the analysis, eg, randomized communication protocols,

embedded systems, cyber-physical systems, safety-critical missions. • Practical experience in applying model checking tools, eg, PRISM, MRMC, UPPAAL.

teaching and learning method

The module consists of lectures and tutorials. During the lectures students are asked to solve small exercises online. Students also received weekly assignments, whose solution is discussed in the tutorials.

SEMANTICS

General competences

Description of study/examination achievements

The examination takes the form of a 120-minute written exam. Knowledge questions test familiarity with concepts of semantics, construction tasks test the ability to define semantics for individual language constructs, and proof tasks test the ability to logically verify statements about individual programs or entire semantics or about the semantic correctness of tools.

Intended learning outcomes

After successfully completing this module, participants will understand the essential concepts of programming language semantics at a scientific level. Participants will have an in-depth understanding of at least one of the three most important techniques for defining the semantics of programming languages: operational, denotational and axiomatic. In particular, students will have the following skills: They can - read semantics and understand their logical consequences; - define the semantics of individual language constructs themselves; - logically derive properties of both individual programs and semantics; - verify tools such as compilers and program analyzers with respect to language semantics.

Contents

The following core topics of the semantics of programming languages are covered: Operational and axiomatic semantics and their relationship to each other. Basic inductive definition principles and proof methods. Basic language properties such as type safety. Applications to tools such as compilers and program analyzers. Program verification with axiomatic semantics. Depending on the form, a simple imperative and/or functional language serves as the basis. Optional additional topics (examples): - Denotational semantics - Semantics of procedures, arrays and pointers - Semantics of object-oriented languages

The following core topics of the semantics of programming languages are covered: Operational and axiomatic semantics and their relationship to each other. Basic inductive definition principles and proof methods. Basic language properties such as type safety. Applications to tools such as compilers and program analyzers. Program verification with axiomatic semantics. Depending on the form, a simple imperative and/or functional language serves as the basis. Optional additional topics (examples): - Denotational semantics - Semantics of procedures, arrays and pointers - Semantics of object-oriented languages

teaching and learning method

In the lecture, the content is presented and explained in dialogue with the students. In the accompanying exercises, the desired learning outcomes are practiced using concrete examples, either individually or in small groups, and with the help of the tutor. Where possible, both programming and proof tasks can be completed using computers.

VIRTUAL MACHINES

General competences

Description of study/examination achievements

The examination takes the form of a 90-minute written exam. Individual tasks require the learned translation schemes to be applied to small example programs. This is intended to test both the knowledge of various programming constructs and the ability to translate these into machine instructions. Other tasks reflect the concept of virtual machines itself by developing new translation schemes for additional programming language concepts. Successful completion of the homework can be included as a bonus in the assessment of the exam. The exact rules for this will be announced in good time at the start of the module.

Intended learning outcomes

Participants are familiar with virtual machines for imperative, functional, logical and object-oriented programming languages. They have mastered the principles of how different programming language concepts are mapped to sequences of machine instructions. They are able to generate code in a virtual machine for sections of programs themselves, and they can apply the principles they have learned to independently develop translation schemes for given constructs.

Contents

If you want to generate code for a programming language such as Prolog, you will quickly find that you would like to use commands when translating that are not provided by a specific machine. On the other hand, the instruction sets of modern computers change so quickly that it does not seem sensible to limit the compiler too much to operations that are provided randomly. Such a limitation could mean that after just a few years you would have to feel compelled to rewrite the compiler completely. When the first Pascal compiler was implemented, the idea was therefore already struck to first generate code for a slightly idealized machine, whose commands then only had to be implemented on the various specific target computers. The translation of modern programming languages such as Prolog, Haskell or Java is also based on this principle. On the one hand, this approach makes the compiler easier to port. On the other hand, it also simplifies the translation itself, since the instruction set can be selected to suit the programming language being translated. We will specifically cover: - the translation of C; - the translation of a functional language; - the translation of Prolog; - the translation of a concurrent variant of C.

If you want to generate code for a programming language such as Prolog, you will quickly find that you would like to use commands when translating that are not provided by a specific machine. On the other hand, the instruction sets of modern computers change so quickly that it does not seem sensible to limit the compiler too much to operations that are provided randomly. Such a limitation could mean that after just a few years you would have to feel compelled to rewrite the compiler completely. When the first Pascal compiler was implemented, the idea was therefore already struck to first generate code for a slightly idealized machine, whose commands then only had to be implemented on the various specific target computers. The translation of modern programming languages such as Prolog, Haskell or Java is also based on this principle. On the one hand, this approach makes the compiler easier to port. On the other hand, it also simplifies the translation itself, since the

instruction set can be selected to suit the programming language being translated. We will specifically cover: - the translation of C; - the translation of a functional language; - the translation of Prolog; - the translation of a concurrent variant of C.

teaching and learning method

Using a slide or blackboard presentation, the lecture presents schemes for translating various program constructs and illustrates these with small examples. In the accompanying exercises, the understanding of the content of the lecture is deepened using suitable tasks for self-study, the ability to apply the learned translation schemes is practiced and new schemes are developed for selected program constructs.

BUSINESS ANALYTICS AND MACHINE LEARNING

General competences

Description of Achievement and Assessment Methods

The examination takes the form of a written 90 minutes test, in which students solve problems to prove they understand the functioning of various methods and their assumptions. Participants demonstrate their ability to interpret the results of different statistical processes and to evaluate their model quality in the exercises. The correct responses require the independent construction of analytical solutions with the help of techniques learned in the module.

Intended Learning Outcomes

After successful completion of the module students are familiarized with common methods of classification, numerical prediction and clustering. They know the assumptions of these processes and understand their functioning, as well as their typical operational applications. Participants are able to analyze data sets with the programming language R and can interpret the results of these analyses.

Content

Regression Analysis, Regression Diagnostics, Generalized Linear Models, Naïve Bayes, Decision Tree Classifiers, Data Preparation, Causal Inference, Model Selection, Ensemble Methods, Clustering, High-Dimensional Problems, Neural Networks, Convex Optimization

Regression Analysis, Regression Diagnostics, Generalized Linear Models, Naïve Bayes, Decision Tree Classifiers, Data Preparation, Causal Inference, Model Selection, Ensemble Methods, Clustering, High-Dimensional Problems, Neural Networks, Convex Optimization

Teaching and Learning Methods

The module consists of a lecture and a content-aligned tutorial. The lecturer presents the content of the module, parts of the corresponding literature and application examples from practice interactively. Students are accustomed with the statistical and machine learning methods and learn to differentiate their usage. In the tutorial participants solve exercises in supervised work and evaluate the respective techniques. In addition, they practice to solve common problems by working with data in teamwork together with their tutor. Students learn to develop their own, data-based solutions. Participants particularly train their technical abilities with programming environments such as R or Python.

DATA ANALYSIS AND VISUALIZATION IN R

General competences

Description of Achievement and Assessment Methods

Written exam and project work: The listed achievements, see Intended Learning Outcomes, are evaluated by one written exam of 90 min. There will be moreover two case studies, where the students must provide the source code that generates the report of an analysis of a given dataset. The analysis of this data covers all topics stated under Intended Learning Outcomes. The first case study covers topics 1-7. The second covers the topics 8-16. The final mark is the exam mark with bonus points for the two case studies.

Intended Learning Outcomes

At the end of the module students are able to: - 1. produce scripts that automatically generate data analysis report - 2. import data from various sources into R - 3. apply the concepts of tidy data to clean and organize a dataset - 4. decide which plot is appropriate for a given question about the data - 5. generate such plots - 6. know the methods of hierarchical clustering, k-means, PCA - 7. apply the above methods and interpret their outcome on real-life datasets - 8. know the concept of statistical testing - 9. devise and implement resampling procedures to assess statistical significance - 10. know the conditions of applications and how to perform in R the following statistical tests: Fisher test, Wilcoxon test, T-test. - 11. know the concept of regression and classification - 12 apply regression and classification algorithms in R - 13. know the concept of error in generalization, cross-validation - 14. implement in R a cross-validation scheme. - 15. know the concepts of sensitivity, specificity, ROC curves - 16. assess the latter in R

Content

R programming basics 1 R programming basics 2 (including report generation with R markdown) Data importing Cleaning and organizing data: Tidy data 1 Cleaning and organizing data: Tidy data 2 Base plot Grammar of graphics 1 Grammar of graphics 2 Unsupervised learning (hierarchical clustering, k-means, PCA) Case study I Drawing robust interpretations 1: empirical testing by sampling Drawing robust interpretations 2: classical statistical tests Supervised learning 1: regression, cross-validation Supervised learning 2: classification, ROC curve, precision, recall Case study II

R programming basics 1 R programming basics 2 (including report generation with R markdown) Data importing Cleaning and organizing data: Tidy data 1 Cleaning and organizing data: Tidy data 2 Base plot Grammar of graphics 1 Grammar of graphics 2 Unsupervised learning (hierarchical clustering, k-means, PCA) Case study I Drawing robust interpretations 1: empirical testing by sampling Drawing robust interpretations 2: classical statistical tests Supervised learning 1: regression, cross-validation Supervised learning 2: classification, ROC curve, precision, recall Case study II

Teaching and Learning Methods

Lecture provides the concept + programming exercises where these concepts are applied on data. The goal of each exercise is the generation of report documents.

MACHINE LEARNING

General competences

Description of Achievement and Assessment Methods

The examination is a written test of 120 minutes. In this exam, students should prove that they are able to select suitable learning algorithms for specific problems and that they understand the probabilistic basics.

Intended Learning Outcomes

Getting familiar with the statistics of machine learning. Basic knowledge of essential learning algorithms. Skill to select, describe and derive appropriate algorithms given a certain problem. Competence to apply these in practical application.

Content

kNN & k-means; linear classifiers; linear regression; Bayes: maximum likelihood and maximum a posteriori estimator, mixture of Gaussians, Hidden Markov Models, Expectation Maximization algorithm, nonlinear neural networks and backprop, Support Vector Machines, unsupervised learning

kNN & k-means; linear classifiers; linear regression; Bayes: maximum likelihood and maximum a posteriori estimator, mixture of Gaussians, Hidden Markov Models, Expectation Maximization algorithm, nonlinear neural networks and backprop, Support Vector Machines, unsupervised learning

Teaching and Learning Methods

Flipped Classroom Lecture on the topics: Probability theory; kNN; multivariate Gaussian; linear regression and classification; kernels; constrained optimization; SVM; GP; neural network; unsupervised learning; expectation maximization; learning theory. Exercise hours on the above topics Homework for self-study on the above topics.

ADVANCED COMPUTER ARCHITECTURE

General competences

Description of Achievement and Assessment Methods

The exam takes the form of written 90 minutes examwithout aids. Questions allow to assess acquaintance with the concepts of Computer Architecture. Questions describing scenarios for the interaction of programs with certain architectures will assess the student's ability to evaluate architectural components and to apply the obtained knowledge.

Intended Learning Outcomes

At the end of the module students know and understand the architecture of current processors as well as of entire IT systems. They can evaluate and assess different designs. The students understand the interaction of architecture and compiler technology. They understand the different classes of parallel architectures and can evaluate their advantages and disadvantages for certain applications.

Content

After an introduction to the goals and the learning outcomes of the module, cross cutting aspects for all advanced architectures are presented. This section covers performance, availability, reliability, fault tolerance, parallelism, memory hierarchy and virtualization. After a recap of the computer architecture basics, the module covers the major types of parallelism and the respective architectures. For instruction level parallelism advanced concepts of the instruction pipeline are discussed as well as superscalar and VLIW processors. This part also covers advanced techniques for the memory hierarchy and compiler support for instruction level parallelism. The next architecture class, data parallel systems, covers vector units in standard processors, array computers, GPGPUs and vector supercomputers. The section presents also the programming interfaces and discusses their interaction with the architectures. Shared memory systems supporting thread level parallelism are discussed next. First the general concepts coherence, memory consistency and synchronization are covered. Then their implementation in uniform and non-uniform memory architectures is presented, ranging from standard multicore systems to large-scale shared memory systems. The last presented architecture class covers distributed memory systems supporting process-level parallelism. This section presents high performance communication networks and design alternatives for network interfaces, manycore processors and massively parallel systems. Parallel file systems are discussed as they are important for all these systems. The module closes with optional presentations about energy efficiency, parallel applications, parallel programming, performance evaluation and non-conventional architectures.

After an introduction to the goals and the learning outcomes of the module, cross cutting aspects for all advanced architectures are presented. This section covers performance, availability, reliability, fault tolerance, parallelism, memory hierarchy and virtualization. After a recap of the computer architecture basics, the module covers the major types of parallelism and the respective architectures. For instruction level parallelism advanced concepts of the instruction pipeline are discussed as well as superscalar and VLIW processors. This part also covers advanced techniques for the memory hierarchy and compiler support for instruction level parallelism. The next architecture class, data parallel

systems, covers vector units in standard processors, array computers, GPGPUs and vector supercomputers. The section presents also the programming interfaces and discusses their interaction with the architectures. Shared memory systems supporting thread level parallelism are discussed next. First the general concepts coherence, memory consistency and synchronization are covered. Then their implementation in uniform and non-uniform memory architectures is presented, ranging from standard multicore systems to large-scale shared memory systems. The last presented architecture class covers distributed memory systems supporting process-level parallelism. This section presents high performance communication networks and design alternatives for network interfaces, manycore processors and massively parallel systems. Parallel file systems are discussed as they are important for all these systems. The module closes with optional presentations about energy efficiency, parallel applications, parallel programming, performance evaluation and non-conventional architectures.

Teaching and Learning Methods

The module consists of a four hour lecture. The students need 90 hours to learn the presented concepts, and to understand and extend the presented examples. They need to come up with own examples to deepen their knowledge and should compare the learned concepts with presentations in the recommended text books.

CLOUD COMPUTING

General competences

Description of Achievement and Assessment Methods

The exam takes the form of an written 60 minutes test without aids. Questions allow to asses acquaintance with the concepts of Cloud and Grid Computing. Questions describing usage scenarios and asking for the evaluation of the learned techniques in these scenarios are used to assess the ability to apply the learned techniques. In a discussion, their ability to solve research question is assessed.

Intended Learning Outcomes

The students know the goals of Cloud and Grid computing. They can present application scenarios in different domains. They are familiar with the fundamental techniques in the areas security, application development and resource management. They can identify the differences and similarities between Cloud and Grid computing and distributed systems. They are able to participate in Cloud and Grid-related research projects.

Content

The lecture starts with an introduction and a presentation of the base technologies for Cloud and Grid computing. The layered architecture of Grids and the base services are presented. Cloud Computing is then introduced and the different models SaaS, PaaS, IaaS. The list of base services is extended for Cloud Computing. The lecture also covers a discussion of legal issues.

The lecture starts with an introduction and a presentation of the base technologies for Cloud and Grid computing. The layered architecture of Grids and the base services are presented. Cloud Computing is then introduced and the different models SaaS, PaaS, IaaS. The list of base services is extended for Cloud Computing. The lecture also covers a discussion of legal issues.

Teaching and Learning Methods

The concepts of Grid and Cloud Computing are introduced in the lecture. In the exercises, the student work on assignments that allow them to train the development of Cloud applications. References to current literature allow the students to deepen their understanding of the concepts.

CONNECTED MOBILITY BASICS

General competences

Description of Achievement and Assessment Methods

Type of Assessment: written exam The exam takes the form of written 120 minutes test and no additional materials are allowed in the exam. The exam assesses the acquaintance with and understanding of concepts and techniques related to connected mobile services and systems. Using small problems, the exam allows to assess the ability to link and to apply this knowledge regarding the desired learning outcomes. During the semester, students will perform small theoretical and practical exercises that allow them to assess their learning progress and to prepare for the exam.

Intended Learning Outcomes

After successful completion of this module, students will remember the basic elements that constitute connected mobile systems and understand the (technical) issues relevant to mobile system and service design and their evaluation. They recognize challenges arising from mobility and mobile networking. They understand the requirements for and the limitations of simulation-based evaluation of mobile services. They have an overview of the different components and techniques at their disposal to realize mobile services. They understand the principles of these underlying methods, know their advantages and disadvantages and can apply methods of design, modeling, and evaluation to simple example cases.

Content

The module IN2324 offers a broad introduction, including the following topics: - Introduction to mobile systems - Mobile systems by example - Mobility: models and simulations - Wireless networking technologies - Architectures and protocols for mobile networks - Architectures for mobile services and systems - Mobile measurements - Security and privacy

The module IN2324 offers a broad introduction, including the following topics: - Introduction to mobile systems - Mobile systems by example - Mobility: models and simulations - Wireless networking technologies - Architectures and protocols for mobile networks - Architectures for mobile services and systems - Mobile measurements - Security and privacy

Teaching and Learning Methods

The lecture is conducted as Blended Learning. We combine a four-hour presence lecture with two-hour presence exercises, which are augmented by homework.

INTRODUCTION TO EMERGING COMPUTING TECHNOLOGIES

General competences

Description of Achievement and Assessment Methods

The examination is performed in form of an oral exam (30mins; applied when the course has less than 20 students) or a written exam (120mins; applied when the course has 20 or more students). The exam will cover tasks conducted before in the hands-on sessions/exercises (which in turn, cover the main content of the lecture in a practical fashion). The tasks/exercises in the exam allow to evaluate how well the students understood the respective concepts and how well they can implement corresponding algorithms for logic synthesis and physical design.

Intended Learning Outcomes

At the end of the module, students will know the basic concepts of emerging technologies and their realizations from a design perspective. They will understand the potential of these technologies in principle and for what applications they can be used. In addition to that, students will get a rough overview how corresponding applications can be realized, how corresponding design steps differ from the design of conventional systems, and what methods are available for their design. Through the programming tasks/mini-projects, students will get hands-on experiences on those issues. Correspondingly considered emerging technologies include reversible computing, quantum computing, microfluidics, and field-coupled nanotechnologies.

Content

Computer technologies will change in the near future. The exponential growth of conventional technologies (according to Moore’s Law) will come to a halt, since physical boundaries will be reached soon. At the same time, further system concepts beyond pure electronics emerge. As a consequence, researchers and engineers are currently considering alternative (emerging) computer technologies which work differently to established (conventional) computation paradigms. Examples include quantum computing, reversible circuits, microfludic devices (also known as Labs-on-a-Chip), or field-coupled nanotechnologies. This module provides an overview of these technologies and the corresponding paradigms. This covers an introduction into the respective concepts as well as possible applications. Afterwards, questions of how to efficiently design applications/solutions for these technologies are discussed.

Computer technologies will change in the near future. The exponential growth of conventional technologies (according to Moore’s Law) will come to a halt, since physical boundaries will be reached soon. At the same time, further system concepts beyond pure electronics emerge. As a consequence, researchers and engineers are currently considering alternative (emerging) computer technologies which work differently to established (conventional) computation paradigms. Examples include quantum computing, reversible circuits, microfludic devices (also known as Labs-on-a-Chip), or field-coupled nanotechnologies. This module provides an overview of these technologies and the corresponding paradigms. This covers an introduction into the respective concepts as well as possible applications. Afterwards, questions of how to efficiently design applications/solutions for these technologies are discussed.

Teaching and Learning Methods

The module will be held in the form of presentations about selected topics in emerging technologies. Using slides presentations and whiteboard sketched, the main concepts of the respectively considered technologies and their design steps are provided. In addition, the students will have the opportunity to deepen der knowledge through individual hands-on experiences with corresponding design tools and methods that allow them to apply the learnt concepts to solve problems in the design of emerging technologies.

MICROPROCESSORS

General competences

Description of Achievement and Assessment Methods

Students have to hand in 4 exercise reports. The students demonstrate with the reports that they have gained deeper knowledge of the specific exerises including low level programming at assembly level, vector processing with intrinsics, cache parameter measurements, cache coherence protocols as well as the difference between several instruction set architectures. They show that they are able to solve extensive low level programming tasks, know how different microprocessors operate at machine language level, and how their performance can be optimised at core level. Students know how to document this knowledge in reports by using literature and with more time at their disposal. In the written 75 minutes examination students demonstrate by answering questions under time pressure and without helping material the theoretical knowledge of microprocessor components at single and multi-core level as well as on their role in high performance computing. The final grade is an averaged grade from the written examination (70 %) and from the lab reports (30 %).

Intended Learning Outcomes

Upon completion of the module, students are able to understand and analyse: - the architecture of a microprocessor in general - features that have been added over processor generations since the first microprocessor was introduced - the state of the art in modern microprocessor architectures - possible trends with regard to future processor architectures - techniques to program and optimise different architectures at assembly language level. Also students are able to understand the role of a microprocessor in high performance computing systems.

Content

Topology and structure of computing systems Technical development of Microprocessors: From 4004 to contemporary processor architectures RISC vs. CISC- Architektur und Instruction Set Architecture and addressing modes Pipelining Pipelining conflicts Examples: x86, ARM Cache structures, cache coherency, MESI and directory based protocols Superscalar vs. VLIW architectures SSE- and AVX Vector Intrinsics Branch prediction Speculatice execution Outlook

Topology and structure of computing systems Technical development of Microprocessors: From 4004 to contemporary processor architectures RISC vs. CISC- Architektur und Instruction Set Architecture and addressing modes Pipelining Pipelining conflicts Examples: x86, ARM Cache structures, cache coherency, MESI and directory based protocols Superscalar vs. VLIW architectures SSE- and AVX Vector Intrinsics Branch prediction Speculatice execution Outlook

Teaching and Learning Methods

Lecture, presentation, workshop, team work, blended learning, experiment

MOBILE DISTRIBUTED SYSTEMS

General competences

Description of Achievement and Assessment Methods

In the written exam (90 minutes) students should prove to be able to identify given problems of the foundations of mobile distributed systems including wireless communication and mobile operating systems and find solutions within limited time.

Intended Learning Outcomes

The students know the basic foundations of mobile distributed systems (MDS). They are able to understand the foundations, problems, and solutions of MDS. They are able to assess current developments. In addition they are able to apply their knowledge to new developments in the area of wireless communication (including their protocols) and the area of mobile operating systems (including important applications).

Content

Mobility, wireless communication, mobile operating systems, architectures, concepts, mechanisms, techniques and their evaluation

Mobility, wireless communication, mobile operating systems, architectures, concepts, mechanisms, techniques and their evaluation

Teaching and Learning Methods

Lecture (presentation and discussion of the foundations), exercises with presentations (application of basic knowledge on various examples), self assessment (with well-established questions)

NETWORK CODING

General competences

Description of Achievement and Assessment Methods

A written examination is performed in the form of a 90 minute exam. Questions of comprehension and arithmetic tasks assess the familiarity with and the mastering of the content covered in the module. Practical content and socket programming are dealt with by examples that have to be explained.

Intended Learning Outcomes

Students know about the basics of NC, both in theory and practical implementations. The projects accompanying the lecture offer students the possibility to gain practical knowledge about coded packet networks, in particular but not limited to wireless networks. Furthermore, students acquire notable skills in socket/network programming.

Content

Conventional routing is unable to achieve a network's capacity as given by the min-cut max-flow theorem - a rather theoretical point of view. Network Coding (NC) is an approach to achieve this very capacity in practice and can thus be considered a generalization of routing. Just think about it: Intermediate nodes (let us call them "router") normally only send identical copies of previously received packets (forget about the headers). NC extends their capability by demanding that sent packets are arbitrary linear combinations of previously received packets. Sounds strange? It is... Here is a brief example to make you curious: Assume A wants to transmit n packets over a lossy link to B. With conventional approaches, B would have to acknowledge successful receipt of each individual packet such that A can retransmit any packets that got lost in transmission. If you use NC, A does not send the actual information contained in each individual packet but linear combinations of all packets. Obviously, B needs n linear independent packets to decode. If any packet gets lost during transmission, B does not need to signal a specific loss to A. Instead, A continues to transmit random (independent) linear combinations of those n packets until B has assembled enough packets to decode the whole batch. This example is very basic, and actually only even a degenerated case of NC. But it gives you the underlying idea. Now just assume there are not only A and B but a number of intermediate nodes, each of them listening, buffering packets, and sending linear combinations. Instead of deciding which specific packet to send where, those nodes have to decide how often a random linear combination is being broadcast. The lecture is a combination of usual classes combined with practical sessions during classes, i.e., you are encouraged to bring your notebooks and participate in the projects. The theoretic part of the lecture covers the following topics: - NC as generalization of routing - min-cut/max-flow theorem - linear programming - packet loss and channel estimation - ARQ mechanisms - random linear network coding (RLNC) - applications of NC - bidirectional coding - NC in transport and link-layer protocols - NC in wireless networks - acknowledgement schemes for NC - combination of NC on different layers The practical part (projects) cover the following topics: - socket programming - IEEE802.11 raw sockets - implementation of RLNC and acknowledgement schemes - window protocols for NC - many more if you want

Conventional routing is unable to achieve a network's capacity as given by the min-cut max-flow theorem - a rather theoretical point of view. Network Coding (NC) is an approach to achieve this very capacity in practice and can thus be considered a generalization of routing. Just think about it: Intermediate nodes (let us call them "router") normally only send identical copies of previously received packets (forget about the headers). NC extends their capability by demanding that sent packets are arbitrary linear combinations of previously received packets. Sounds strange? It is... Here is a brief example to make you curious: Assume A wants to transmit n packets over a lossy link to B. With conventional approaches, B would have to acknowledge successful receipt of each individual packet such that A can retransmit any packets that got lost in transmission. If you use NC, A does not send the actual information contained in each individual packet but linear combinations of all packets. Obviously, B needs n linear independent packets to decode. If any packet gets lost during transmission, B does not need to signal a specific loss to A. Instead, A continues to transmit random (independent) linear combinations of those n packets until B has assembled enough packets to decode the whole batch. This example is very basic, and actually only even a degenerated case of NC. But it gives you the underlying idea. Now just assume there are not only A and B but a number of intermediate nodes, each of them listening, buffering packets, and sending linear combinations. Instead of deciding which specific packet to send where, those nodes have to decide how often a random linear combination is being broadcast. The lecture is a combination of usual classes combined with practical sessions during classes, i.e., you are encouraged to bring your notebooks and participate in the projects. The theoretic part of the lecture covers the following topics: - NC as generalization of routing - min-cut/max-flow theorem - linear programming - packet loss and channel estimation - ARQ mechanisms - random linear network coding (RLNC) - applications of NC - bidirectional coding - NC in transport and link-layer protocols - NC in wireless networks - acknowledgement schemes for NC - combination of NC on different layers The practical part (projects) cover the following topics: - socket programming - IEEE802.11 raw sockets - implementation of RLNC and acknowledgement schemes - window protocols for NC - many more if you want

Teaching and Learning Methods

Lecture with exercises The lecture is complemented by integrated exercises and demonstrations. On that basis, students are encouraged to propose individual practical projects, which are worked on in the second half of the lecture. Those projects may be done in groups of at most two students and are supervised during class hours. At the end of the lecture period students should present their projects to their class mates by presentation and/or demonstration.

NETWORK AND SYSTEM MANAGEMENT

General competences

Description of Achievement and Assessment Methods

written exam, programming project The exam takes the form of a 60 minutes written test.

Intended Learning Outcomes

The students understand the main issues of the management of IT systems, including its incorporation into IT processes of an organisation, and know relevant solutions.

Content

Starting from basics of networked systems requirements of IT-management (esp. network and systems management) are analyzed. This leads to the concept of management-architectures. Examples of management architectures are discussed: OSI/TMN, Internet/SNMP, CORBA, DMTF, Web-based management. An introduction of management platforms and important management tools follows. Finally various management scenarios and IT-management process in the service management area are discussed.

Starting from basics of networked systems requirements of IT-management (esp. network and systems management) are analyzed. This leads to the concept of management-architectures. Examples of management architectures are discussed: OSI/TMN, Internet/SNMP, CORBA, DMTF, Web-based management. An introduction of management platforms and important management tools follows. Finally various management scenarios and IT-management process in the service management area are discussed.

Teaching and Learning Methods

Lecture, exercise course, problems for individual study, programming project in a group

VIRTUALIZATION TECHNIQUES

General competences

Description of Achievement and Assessment Methods

The exam takes the form of a written 75 minutes exam. Questions allow to asses acquaintance with the classification and concepts of virtual machines, as well as techniques used therein for improved performance. Questions asking for advantages and disadvantages of given virtualization approaches and optimization techniques allow to evaluate the understanding of benefits of various approaches.

Intended Learning Outcomes

At the end of the module students are able to classify and evaluate different virtual machines, both concepts and actual products in use today. They can analyse their usefulness in various scenarios, and they understand the technical design trade-offs, implementation challenges as well as typical solutions for better performance.

Content

- Classification of Virtual Machines - Operating system Virtualization - ISA Virtualization ++ Interpretation ++ Binary Translation ++ Dynamic Optimization - Process Virtual Machines - High-Level Language Virtual Machines - HW/SW-Codesign Virtual Machines - system Virtual Machines - Multiprocessor Virtualization - Applications

- Classification of Virtual Machines - Operating system Virtualization - ISA Virtualization ++ Interpretation ++ Binary Translation ++ Dynamic Optimization - Process Virtual Machines - High-Level Language Virtual Machines - HW/SW-Codesign Virtual Machines - system Virtual Machines - Multiprocessor Virtualization - Applications

Teaching and Learning Methods

On the one hand, the module consists of lectures to present the content, do discussions on presented problems and solutions, and run tasks for student teams to search for solutions to presented problems. On the other hand, there are online and offline exercises in which existing implementations of virtual machines get studied by students, techniques get evaluated and extended. Literature references are provided for self-study on details. On a voluntary basis, students get the chance to give short presentations on selected topics.

AUTONOMOUS DRIVING

General competences

Description of Achievement and Assessment Methods

The written exam (60min) tests the student's understanding of Autonomous Driving concepts and the students ability to apply the learned methodologies to according scenarios.

Intended Learning Outcomes

At the end of the module students are able to understand the basics of autonomous driving, and know how they relate to the scientific and practical challenges in the field. The students know exemplary solution approaches for central questions and can apply them to simple problems.

Content

The lecture teaches the foundations of autonomous driving and relates them to current scientific and industrial topics in the domain. Central topic areas are discussed and exemplary solution approaches demonstrated, to enable the students in applying these approaches to basic autonomous driving scenarios. Overview: Path planning, Decision Making, Trajectories Multi-Sensor-Data-Fusion, Tracking, Detection, Sensormodels Lane detection, modelling and mapping AI in vehicles, Scenario Classification and Prediction Semantic concepts for scenario description and reasoning Architectures and Operating Systems for Autonomous Driving Connected Vehicles & Intelligent Infrastructure Functional and Operational Safety A bonus of 0.3 grades can be gained by developing a python demonstrator for one of the discussed central topics.

The lecture teaches the foundations of autonomous driving and relates them to current scientific and industrial topics in the domain. Central topic areas are discussed and exemplary solution approaches demonstrated, to enable the students in applying these approaches to basic autonomous driving scenarios. Overview: Path planning, Decision Making, Trajectories Multi-Sensor-Data-Fusion, Tracking, Detection, Sensormodels Lane detection, modelling and mapping AI in vehicles, Scenario Classification and Prediction Semantic concepts for scenario description and reasoning Architectures and Operating Systems for Autonomous Driving Connected Vehicles & Intelligent Infrastructure Functional and Operational Safety A bonus of 0.3 grades can be gained by developing a python demonstrator for one of the discussed central topics.

Teaching and Learning Methods

The contents of the module are presented as a lecture with slides. A deeper relation to current industrial topics is achieved through talks by invited experts. The student's ability to practically apply the learned concepts can be enhanced by developing a demonstrator in python.

AUTONOMOUS NAVIGATION FOR FLYING ROBOTS

General competences

Description of Achievement and Assessment Methods

The module examination consists of a 60-minute written, electronic exam that takes place on a computer separate from the Internet. Answering the questions partly requires ticking given multiple answers, performing (simple) calculations by hand or calculator, and solving a programming task using the interactive quadrocopter simulator in Python.

Intended Learning Outcomes

After successful participation of this module, students will be able to - understand the flight principles of quadrotors and their application potential - specify the pose of objects in 3D space and to perform calculations between them (e.g. compute the relative motion) - explain the principles of Bayesian state estimation - implement and apply an extended Kalman filter (EKF), and to select appropriate parameters for it, - implement and apply a PID controller for state control, and to fine tune its parameters, - understand and explain the principles of visual motion estimation and 3D mapping

Content

In recent years, flying robots such as miniature helicopters or quadrotors have received a large gain in popularity. Potential applications range from aerial filming over remote visual inspection of industrial sites to automatic 3D reconstruction of buildings. Navigating a quadrotor manually requires a skilled pilot and constant concentration. Therefore, there is a strong scientific interest to develop solutions that enable quadrotors to fly autonomously and without constant human supervision. This is a challenging research problem because the payload of a quadrotor is uttermost constrained and so both the quality of the onboard sensors and the available computing power is strongly limited. In this course, we will introduce the basic concepts for autonomous navigation for quadrotors. The following topics will be covered: - 3D geometry, - probabilistic state estimation, - visual odometry, SLAM, 3D mapping, - linear control. In particular, you will learn how to infer the position of the quadrotor from its sensor readings and how to navigate it along a trajectory. The course consists of a series of weekly lecture videos that will be interleaved by interactive quizzes and hands-on programming tasks. For the flight experiments, we provide a browser-based quadrotor simulator which requires the students to write small code snippets in Python. This course is intended for undergraduate and graduate students in computer science, electrical engineering or mechanical engineering. This course has been offered by TUM for the first time in summer term 2014 on EdX with more than 20.000 registered students of which 1.400 passed examination. The MOOC is based on the previous TUM lecture “Visual Navigation for Flying Robots” which received the TUM TeachInf Award for the best lecture in 2012 and 2013.

In recent years, flying robots such as miniature helicopters or quadrotors have received a large gain in popularity. Potential applications range from aerial filming over remote visual inspection of industrial sites to automatic 3D reconstruction of buildings. Navigating a quadrotor manually requires a skilled pilot and constant concentration. Therefore, there is a strong scientific interest to develop solutions that enable quadrotors to fly autonomously and without constant human supervision. This is a challenging research problem because the

payload of a quadrotor is uttermost constrained and so both the quality of the onboard sensors and the available computing power is strongly limited. In this course, we will introduce the basic concepts for autonomous navigation for quadrotors. The following topics will be covered: - 3D geometry, - probabilistic state estimation, - visual odometry, SLAM, 3D mapping, - linear control. In particular, you will learn how to infer the position of the quadrotor from its sensor readings and how to navigate it along a trajectory. The course consists of a series of weekly lecture videos that will be interleaved by interactive quizzes and hands-on programming tasks. For the flight experiments, we provide a browser-based quadrotor simulator which requires the students to write small code snippets in Python. This course is intended for undergraduate and graduate students in computer science, electrical engineering or mechanical engineering. This course has been offered by TUM for the first time in summer term 2014 on EdX with more than 20.000 registered students of which 1.400 passed examination. The MOOC is based on the previous TUM lecture “Visual Navigation for Flying Robots” which received the TUM TeachInf Award for the best lecture in 2012 and 2013.

Teaching and Learning Methods

Teaching format: MOOC, Learning methods: Video lectures, experiments (simulation), individual exercises (homework)

REAL-TIME SYSTEMS

General competences

Description of Achievement and Assessment Methods

The examination is carried out in the form of a 90-minute written exam. This is intended to demonstrate that specific problems of real-time systems have been understood and can be solved through the use of suitable algorithms and simulations. No aids are allowed.

Intended Learning Outcomes

At the end of this module students are able to analyse the specific requirements of real-time systems. They are able to select from a variety of possible solutions based on relevant aspects, such as modeling concepts, scheduling algorithms, operating systems, programming languages, etc. They understand typical problems with cuncurrent programming and are familiar with different mechanisms to overcome these problems.

Content

The lecture series comprises special methods, solutions and problems pertaining to real-time systems. To do so, concrete examples where real-time systems are required are presented. The difference between real-time and non-real-time systems is highlighted and it is demonstrated why closed and open-loop control algorithms are fundamentally dependent on real-time systems with their special hard- and software properties. Modelling of real-time systems, concurrency, scheduling algorithms, special operating systems and programming paradigms, clocks, real-time communication and typical hardware for sensors and actuators are discussed in detail. The lecture series concludes with an introduction to fault-tolerant systems.

The lecture series comprises special methods, solutions and problems pertaining to real-time systems. To do so, concrete examples where real-time systems are required are presented. The difference between real-time and non-real-time systems is highlighted and it is demonstrated why closed and open-loop control algorithms are fundamentally dependent on real-time systems with their special hard- and software properties. Modelling of real-time systems, concurrency, scheduling algorithms, special operating systems and programming paradigms, clocks, real-time communication and typical hardware for sensors and actuators are discussed in detail. The lecture series concludes with an introduction to fault-tolerant systems.

Teaching and Learning Methods

The module consists of a lecture, an exercise and tasks for self-study. The contents of the lecture are conveyed in the lecture and through presentations. Students are encouraged to deal with the content of the topics through small tasks set in the course of the lectures, as well as by solving exercise sheets. The solution to the exercises is discussed in the exercise.

INTRODUCTION TO DIGITAL SIGNAL PROCESSING

General competences

Description of Achievement and Assessment Methods

The examination is carried out in the form of a 105-minute written exam. This is intended to demonstrate that modern methods of signal processing are mastered and suitable algorithms can be selected for a specific problem. No aids are allowed.

Intended Learning Outcomes

At the end of this module students are able to evaluate theoretical concepts and underlying algorithms of modern digital signal processing systems. In the exercise course, the students learn to apply these concepts to a concrete problem and to decide on a proper implementation.

Content

Signals and systems in time and frequency domain, examples, discretization, filter and frequency transforms, compression, realization (signal processors), outlook to digital image processing

Signals and systems in time and frequency domain, examples, discretization, filter and frequency transforms, compression, realization (signal processors), outlook to digital image processing

Teaching and Learning Methods

The module consists of a lecture, an exercise and tasks for self-study. The contents of the lecture are conveyed in the lecture and through presentations. Students are encouraged to deal with the content of the topics through small tasks set in the course of the lectures, as well as by solving exercise sheets. The solution to the exercises is discussed in the exercise.

FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE

General competences

Description of Achievement and Assessment Methods

Written exam at the end of the semester lasting 90min. The questions will cover most of the learned material and are typically shorter than the problems solved in the exercise, but similar in difficulty. All questions will be aligned with the learning outcomes. The correct answers to these questions may involve mathematical derivations and calculations. As an incentive to create artificial intelligence oneself, we provide programming challenges: if students solve a required number of programming challenges, they obtain a 0.3 grade bonus for their exam. During the exam, students are allowed to use a pen (color should not be red or green and pencils are not allowed), a non-programmable calculator, and a formula sheet. Whether this formula sheet is provided or can be written by the students will be announced before each exam.

Intended Learning Outcomes

After attending the module, you are able to create artificial intelligence on a basic level using search techniques, logics, probability theory and decision theory. Your learned abilities will be the foundation for more advanced topics in artificial intelligence. In particular, you will acquire the following skills: - You can analyze problems of artificial intelligence and judge how difficult it is to solve them. - You can recall the basic concepts of intelligent agents and know possible task environments. - You can formalize, apply, and understand search problems. - You understand the difference between constraint satisfaction and classical search problems as well as apply and evaluate various constraint satisfaction approaches. - You can critically assess the advantages and disadvantages of logics in artificial intelligence. - You can formalize problems using propositional and first-order logic. - You can apply automatic reasoning techniques in propositional and first-order logic. - You understand the advantages and disadvantages of probabilistic and logic-based reasoning. - You can apply and critically asses methods for probabilistic reasoning with Bayesian networks and Hidden Markov Models. - You understand and know how to compute rational decisions. - You have a basic understanding on how a machine learns. - You know the basic areas and concepts in robotics.

Content

- Task environments and the structure of intelligent agents. - Solving problems by searching: breadth-first search, uniform-cost search, depth-first search, depth-limited search, iterative deepening search, greedy best-first search, A* search. - Constraint satisfaction problems: defining constraint satisfaction problems, backtracking search for constraint satisfaction problems, heuristics for backtracking search, interleaving search and inference, the structure of constraint satisfaction problems. - Logical agents: propositional logic, propositional theorem proving, syntax and semantics of first-order logic, using first-order logic, knowledge engineering in first-order logic, reducing first-order inference to propositional inference, unification and lifting, forward chaining, backward chaining, resolution. - Bayesian networks: acting under uncertainty, basics of probability theory, Bayesian networks, inference in Bayesian networks, approximate inference in Bayesian networks. - Hidden Markov models: time and uncertainty, inference in hidden Markov models (filtering, prediction, smoothing, most likely explanation), approximate inference in hidden Markov models. - Rational

decisions: introduction to utility theory, utility functions, decision networks, the value of information, Markov decision processes, value iteration, policy iteration, partially observable Markov decision processes. - Learning: types of learning, supervised learning, learning decision trees, reinforcement learning. - Introduction to robotics: robot hardware, robotic perception, path planning, planning uncertain movements, control of movements, application domains.

- Task environments and the structure of intelligent agents. - Solving problems by searching: breadth-first search, uniform-cost search, depth-first search, depth-limited search, iterative deepening search, greedy best-first search, A* search. - Constraint satisfaction problems: defining constraint satisfaction problems, backtracking search for constraint satisfaction problems, heuristics for backtracking search, interleaving search and inference, the structure of constraint satisfaction problems. - Logical agents: propositional logic, propositional theorem proving, syntax and semantics of first-order logic, using first-order logic, knowledge engineering in first-order logic, reducing first-order inference to propositional inference, unification and lifting, forward chaining, backward chaining, resolution. - Bayesian networks: acting under uncertainty, basics of probability theory, Bayesian networks, inference in Bayesian networks, approximate inference in Bayesian networks. - Hidden Markov models: time and uncertainty, inference in hidden Markov models (filtering, prediction, smoothing, most likely explanation), approximate inference in hidden Markov models. - Rational decisions: introduction to utility theory, utility functions, decision networks, the value of information, Markov decision processes, value iteration, policy iteration, partially observable Markov decision processes. - Learning: types of learning, supervised learning, learning decision trees, reinforcement learning. - Introduction to robotics: robot hardware, robotic perception, path planning, planning uncertain movements, control of movements, application domains.

Teaching and Learning Methods

The module consists of a lecture and exercise classes. The content of the lecture is presented via slides, which are completed during the lecture using the blackboard and/or an electronic writing pad. Students are encouraged to additionally study the relevant literature. In the exercise classes, the learned content is applied to practical examples to consolidate the content of the lecture. Students should ideally have tried to solve the problems before they attend the exercise. To encourage more participation, students are regularly asked questions or encouraged to participate in online polls. As an incentive to create artificial intelligence oneself, we provide programming challenges: if students solve a required number of programming challenges, they obtain a 0.3 grade bonus for their exam.

INTRODUCTION TO MOBILE ROBOTICS

General competences

Description of Achievement and Assessment Methods

Written exam at the end of the semester lasting 90min and no additional materials are allowed in the exam. In the written exam, students demonstrate that they understood the key concepts of mobile robots that can autonomously navigate their environments. The exam questions will focus on the principles which have been discussed during the lectures and students will have to apply their knowledge in the exam to solve problems similar to those discussed during the exercises as part of the lecture series.

Intended Learning Outcomes

After successfully finishing this module, students will be able to design and develop mobile robots that can autonomously navigate in their environment. The key concepts taught in this course include basics of probability theory, recursive Bayes filtering, probabilistic sensor and motion models, probabilistic approaches to robot localization and mapping, techniques for simultaneous localization and mapping (SLAM), basic approaches to collision avoidance, path planning and exploration. In particular, students will acquire the following skills: - They can analyze sensors (ultrasound, LiDAR, Cameras) used for robot navigation and design probabilistic models for them. - They can design probabilistic motion models. - They can realize a probabilistic localization approach using either a particle filter, a discrete filter, or variants of the Kalman filter. - They are able to develop and realize an approach to learn occupancy grid maps from sensor data given the poses of the robot are known. - They have a fundamental understanding of recursive Bayes filtering, are able to derive its equation and can critically assess the advantages and disadvantages of different representations of distributions for the recursive Bayes filter. - They have a fundamental understanding of the problem of simultaneous localization and mapping (SLAM). - They have a basic understanding of motion-planning and collision avoidance methods. - They have the capability to develop a navigation system for wheeled mobile robots.

Content

- Wheeled locomotion - Proximity sensors - Probabilistic Robotics - Probabilistic Motion Models - Probabilistic Sensor Models - Bayes Filter – Discrete Filters - Bayes Filter – Kalman Filters - Bayes Filter – Particle Filter - Occupancy grid maps and reflection maps - SLAM – Simultaneous Localization and Mapping - SLAM – Landmark-based SLAM - SLAM – Grid-based SLAM - SLAM – Graph-based SLAM - Iterative Closest Point Algorithm - Path planning and Collision Avoidance - Exploration

- Wheeled locomotion - Proximity sensors - Probabilistic Robotics - Probabilistic Motion Models - Probabilistic Sensor Models - Bayes Filter – Discrete Filters - Bayes Filter – Kalman Filters - Bayes Filter – Particle Filter - Occupancy grid maps and reflection maps - SLAM – Simultaneous Localization and Mapping - SLAM – Landmark-based SLAM - SLAM – Grid-based SLAM - SLAM – Graph-based SLAM - Iterative Closest Point Algorithm - Path planning and Collision Avoidance - Exploration

Teaching and Learning Methods

The module consists of a lecture and exercise classes. The content of the lecture is presented via slides, writing on a blackboard or digital media. Students are encouraged to additionally study the relevant literature. In the exercise classes, the learned content is applied to practical examples to consolidate the content of the lecture. Students should ideally have tried to solve the problems before they attend the exercise. To encourage more participation, students are regularly asked questions or encouraged to participate in online polls. Students are also encouraged to implement fundamental algorithms themselves.

SENSOR-BASED ROBOTIC MANIPULATION AND LOCOMOTION

General competences

Description of Achievement and Assessment Methods

Type of Assessment: written exam (60 minutes) The exam takes the form of written test of 60 minutes. Questions allow assessing acquaintance with concepts of Informatics and programming, small programming tasks assess the ability to conceive appropriate algorithmic solutions and realize concurrent applications.

Intended Learning Outcomes

The lecture introduces methods for modelling as well as kinematic and dynamic control of complex (for example humanoid) robot systems which interact with unknown environments. Students will gain an overview on the main algorithms and methods used in robotics for manipulation and locomotion.

Content

Introduction: Manipulation und locomotion with complex robot systems - Basics of differential geometry Representations of orientation: so3, se3 Transformation of coordinates for vectors, convectors, tensors - Task based control decoupling in task space Robot model and control in task coordinates - Redundant systems, micro-/macro-manipulation Inverse Kinematics Task priorities: Cartesian and null-space coordinates - Parallel robots Dual arm systems Dexterous hands Humanoid manipulators Legged robots - Grasp planning - Mobile manipulation - Visual servoing - Mobile, non-holonomic robots sensor-guided surgical robotics

Introduction: Manipulation und locomotion with complex robot systems - Basics of differential geometry Representations of orientation: so3, se3 Transformation of coordinates for vectors, convectors, tensors - Task based control decoupling in task space Robot model and control in task coordinates - Redundant systems, micro-/macro-manipulation Inverse Kinematics Task priorities: Cartesian and null-space coordinates - Parallel robots Dual arm systems Dexterous hands Humanoid manipulators Legged robots - Grasp planning - Mobile manipulation - Visual servoing - Mobile, non-holonomic robots sensor-guided surgical robotics

Teaching and Learning Methods

Class lecture, combined with their own experimental elaboration of the examples on the computer and further reading for review technical issues.

APPLIED CRYPTOGRAPHY FOR DECENTRALIZED SYSTEMS (ACDS)

General competences

Description of Achievement and Assessment Methods

A written examination is performed in the form of a 75-minute exam. Questions of comprehension and arithmetic tasks check the familiarity with the technologies and methods of cryptographic primitives, protocols, and zero-knowledge proof systems for decentralized systems, and confidential computing principles covered in the module.

Intended Learning Outcomes

After participating in the module, students can describe the goals and properties and internals of cryptography of selected decentralized systems and blockchain systems. They can describe the concepts of zero-knowledge-proof constructions and their application in decentralized systems. They will be able apply advanced security concepts such as threshold cryptography and trusted execution environments in a correct manner to solve security problems. They are able to assess performance properties of different cryptographic concepts for decentralized systems, and to develop solutions based on the methods and concepts they have learned.

Content

Cryptographic Proof Basics • Hash Functions, Commitment Schemes, ZKP Basics • Sigma Protocols • SNARK Constructions Polynomials in Cryptography (Secret Sharing, Polynomial Commitments) • Arithmetization • PlonK SNARK Applications • Circuit programming • zkVMs Trusted Execution Environments • Process and VM based solutions • Building blocks, Performance and use-cases Secure Multiparty Computation • Oblivious Transfer, Garbled Circuits • Beaver Triples, BGW and GMW protocols

Cryptographic Proof Basics • Hash Functions, Commitment Schemes, ZKP Basics • Sigma Protocols • SNARK Constructions Polynomials in Cryptography (Secret Sharing, Polynomial Commitments) • Arithmetization • PlonK SNARK Applications • Circuit programming • zkVMs Trusted Execution Environments • Process and VM based solutions • Building blocks, Performance and use-cases Secure Multiparty Computation • Oblivious Transfer, Garbled Circuits • Beaver Triples, BGW and GMW protocols

Teaching and Learning Methods

Lecture for content transfer, as well as tasks for self-study in order to deepen the subject, as well as programming challenges to test and apply the learned knowledge.

IT SECURITY 2

General competences

Description of Achievement and Assessment Methods

The examination is performed in the form of a 90-minute exam. The exam questions test whether the candidate has acquired a subset of the skills in the list below. List of Skills: The student - masters basic and advanced concepts and techniques to secure systems and individual artifacts, - can apply learned techniques correctly, - is able to identify security problems using concrete examples, - knows the advantages and disadvantages of the most important classical and advanced security concepts.

Intended Learning Outcomes

After participating in the module, students are able to evaluate complex security solutions. They are able to analyze the security level of applications. They will be able apply basic and advanced security concepts in a correct manner to solve security problems. They understand the causes of security problems in networks but also in mobile and embedded systems. They master the most important concepts for their protection and can also apply advanced approaches. They are able to identify possible security weaknesses in designs or protocols and to develop solutions based on the methods and concepts they have learned.

Content

The module treats selected types in depth and discusses special topics of IT security. Current concepts and solutions in the field of digital identity, such as smart cards, physically unclonable functions (PUF), SSI and token-based authentication in distributed systems are dealt with in depth. In the area of application security, selected issues such as the security of Instant Messenger services are discussed. In the area of system security, the module is dedicated to advanced concepts such as Trusted Computing, Trusted Execution Environments and Confidential Computing. It also discusses selected aspects of hardware-security. The module deals with the current and developing state of the security of wireless and mobile communication architectures (e.g. 5G) and with concepts for ad-hoc security in networked IoT devices (e.g. BluetoothLE). The module will also give an insight into the methodological development and evaluation of secure systems (security engineering).

The module treats selected types in depth and discusses special topics of IT security. Current concepts and solutions in the field of digital identity, such as smart cards, physically unclonable functions (PUF), SSI and token-based authentication in distributed systems are dealt with in depth. In the area of application security, selected issues such as the security of Instant Messenger services are discussed. In the area of system security, the module is dedicated to advanced concepts such as Trusted Computing, Trusted Execution Environments and Confidential Computing. It also discusses selected aspects of hardware-security. The module deals with the current and developing state of the security of wireless and mobile communication architectures (e.g. 5G) and with concepts for ad-hoc security in networked IoT devices (e.g. BluetoothLE). The module will also give an insight into the methodological development and evaluation of secure systems (security engineering).

Teaching and Learning Methods

The module consists of a lecture and an accompanying exercise. In the lecture, the teaching content is conveyed and the students are encouraged to study the literature and to deal with the content of the topics. In the exercises, concrete questions are discussed and examples are worked on, in groups. With 6-8 exercise sheets, the students are guided to work through the topics of the lecture and to deepen the material of the lecture using concrete tasks.

CRYPTOGRAPHY

General competences

Description of Achievement and Assessment Methods

The examination takes the form of a 90-minute written test. The exam questions test whether the candidate has acquired a subset of the skills in the list below List of Skills: The student - masters the most important modern cryptographic schemes and describes basic elements (e.g. AES, ChaCha, RSA, AEAD, rCTR, NMAC, CBC-MAC, ENC-THEN-MAC, OAEP, FDH, PSS, DH, Elgamal, hybrid encryption) and how they work and explain the mathematical basis; - understands the functionality and starting points of attacks on cryptographic processes and is able to explain which properties a secure cryptographic system must have; - understands the need for (pseudo-)randomization in cryptography and the difference between randomness and pseudo-randomness; - can explain the assumptions underlying public key cryptography; - can use the definitions to decide whether a simple cryptographic scheme is secure or not; - can create provably secure cryptographic schemes based on these constructions and primitives; - can describe and apply the algebraic and number-theoretic fundamentals of RSA and DLP-based cryptography; - can explain the basic advantages and disadvantages of elliptic curves in DLP-based cryptography.

Intended Learning Outcomes

After participating in the module, the student is able to - recognize the basic primitives of symmetric and asymmetric cryptography, - understand the theoretical foundations of these primitives, - analyze cryptographic methods based on it, - understand the most important security definitions.

Content

- Theoretical basics: ++ Security definitions: perfect secrecy, computational security (IND-CPA,IND-CCA,IND-CCA2), semantic security ++ Cryptographic primitives and pseudo-random: pseudo-random number generator (PRG), -functions (PRF) and -permutations (PRP), one-way functions (OWF) and -permutations (OWP) (with trapdoor (TDP)), cryptographic hash functions, tweakable blockcipher (TBC) ++ Basics of group and number theory, elliptic curves - Symmetric cryptography: ++ Block ciphers: AES, DES and stream ciphers: ChaCha ++ Construction of encryption methods based on block ciphers. ++ Construction of message authentication code: CBC-MAC, NMAC, HMAC - Asymmetric cryptography: ++ The RSA problem and the encryption and signature methods derived from it: RSA-OAEP, RSA-FDH, RSA-PSS ++ The discrete logarithm and methods derived from it: Diffie-Hellman protocol, El Gamal, DH-KEM, DSA

- Theoretical basics: ++ Security definitions: perfect secrecy, computational security (IND-CPA,IND-CCA,IND-CCA2), semantic security ++ Cryptographic primitives and pseudo-random: pseudo-random number generator (PRG), -functions (PRF) and -permutations (PRP), one-way functions (OWF) and -permutations (OWP) (with trapdoor (TDP)), cryptographic hash functions, tweakable blockcipher (TBC) ++ Basics of group and number theory, elliptic curves - Symmetric cryptography: ++ Block ciphers: AES, DES and stream ciphers: ChaCha ++ Construction of encryption methods based on block ciphers. ++ Construction of message authentication code: CBC-MAC, NMAC, HMAC - Asymmetric cryptography: ++ The RSA problem and the encryption and signature methods derived from

it: RSA-OAEP, RSA-FDH, RSA-PSS ++ The discrete logarithm and methods derived from it: Diffie-Hellman protocol, El Gamal, DH-KEM, DSA

Teaching and Learning Methods

The module consists of a lecture and an accompanying exercise. In the lecture, the teaching content is conveyed and the students are encouraged to study the literature and to deal with the content of the topics. In the exercises, concrete questions are discussed and examples are worked on, sometimes in groups.

NETWORK SECURITY

General competences

Description of Achievement and Assessment Methods

The examination will take the form of a 75-minute examination. Questions of comprehension and arithmetic tasks check the familiarity with the technologies and methods of cryptographic procedures and protocols and mechanisms for network security covered in the module.

Intended Learning Outcomes

Participants understand security goals for the Internet and the components in which communication protocols are implemented. They understand the possibilities available to attackers in the network. They understand the protection offered by cryptographic and network security mechanisms, and have the knowledge to apply network security protocols and implement architectures that can achieve specific security goals.

Content

The course provides an introduction to the field of network security. Starting with possible threats and attack scenarios, requirements for providing specific security services are derived. After introducing the basic concepts of security mechanisms, the integration of security mechanisms into network architectures and network protocols are discussed. Security vulnerabilities of existing network architectures are also discussed. As a basis for the realization of security mechanisms, cryptographic algorithms (in particular symmetric cryptography, public key cryptography and cryptographic hash functions) are presented. Afterwards, the basics and methods for security protocols for authentication, authorization, access control, message integrity, confidentiality and non-repudiation are discussed. Subsequent sections present specific security mechanisms, in particular of the TCP/IP protocol family. The standard examples include PKI, Kerberos, IPSec, and TLS, Firewall-architectures and Intrusion Detection Systems.

The course provides an introduction to the field of network security. Starting with possible threats and attack scenarios, requirements for providing specific security services are derived. After introducing the basic concepts of security mechanisms, the integration of security mechanisms into network architectures and network protocols are discussed. Security vulnerabilities of existing network architectures are also discussed. As a basis for the realization of security mechanisms, cryptographic algorithms (in particular symmetric cryptography, public key cryptography and cryptographic hash functions) are presented. Afterwards, the basics and methods for security protocols for authentication, authorization, access control, message integrity, confidentiality and non-repudiation are discussed. Subsequent sections present specific security mechanisms, in particular of the TCP/IP protocol family. The standard examples include PKI, Kerberos, IPSec, and TLS, Firewall-architectures and Intrusion Detection Systems.

Teaching and Learning Methods

Lecture for content transfer, as well as tasks for self-study in order to deepen the subject, as well as programming challenges to test and apply the learned knowledge.

NETWORKS FOR MONETARY TRANSACTIONS

General competences

Description of Achievement and Assessment Methods

The exam takes the form of a 60 minutes written test. In the written exam students should prove to be able to identify a given problem and find solutions within limited time.

Intended Learning Outcomes

Verstehen und Einschätzen der Grundlagen, Architektur und Sicherheit von Netzwerken für den nationalen und internationalen Zahlungsverkehr sowie deren rechtliche Rahmenbedingungen

Content

Systemarchitektur (Hardware, Software) Protokollschichten und Protokolle (eg. ISO 8583) Sicherheitsarchitekturen und Sicherheitsstandards Authorisierungslösungen und Signaturen Arten von Attacken Mobile Architekturen und deren Sicherheitskonzepte (OTA Services) Anwendungen

Systemarchitektur (Hardware, Software) Protokollschichten und Protokolle (eg. ISO 8583) Sicherheitsarchitekturen und Sicherheitsstandards Authorisierungslösungen und Signaturen Arten von Attacken Mobile Architekturen und deren Sicherheitskonzepte (OTA Services) Anwendungen

Teaching and Learning Methods

Lecture

PEER-TO-PEER-SYSTEMS AND SECURITY

General competences

Description of Achievement and Assessment Methods

Students will be evaluated based on their project in which they design and implement components of a peer-to-peer protocol or application. The outcome of the project allows to assess the competences of a student of understanding and explaining goals and properties of peer-to-peer systems, to utilize selected mechanisms and protocols of peer-to-peer networks in order to design system components with specific properties, and to implement such components. The project will be evaluated based on the implementation and tests (70%), the design and implementation of the P2P protocol (20%) and the different reports (10%).

Intended Learning Outcomes

At the end of the module the students can describe the goals and properties of peer-to-peer systems. They can recite the internals of selected systems. They can utilize the fundamental mechanisms and protocols of peer-to-peer networks, in particular for search of data and managing membership. They can explain security goals of such networks and discuss concepts to provide security services. They can assess various mechanisms to provide anonymity and can compare selected anonymity systems. Finally, they are able to apply this theoretical knowledge into the design and implementation of new protocols and applications, can explain their new design, predict its behavior and critially analyze its weaknesses.

Content

The course covers the following topics: - P2P applications, advantages and disadvantages of P2P systems - common problems and operations in P2P systems - routing in structured and unstructured overlay networks, in particular distributed hash tables - network address translation and traversal - decentralized network size estimation - reputation systems - common attacks on P2P protocols and defenses against them - anonymity, protocols to create anonymity and attacks on anonymity The various protocols are described in theory and practical implementations of those ideas are discussed (in particular Freenet, Bittorrent, Tor, JAP, GNUnet, I2P and Gnutella). The course will also include a practical component focusing on the design and implementation of a new P2P protocol or application with design choices being discussed in class

The course covers the following topics: - P2P applications, advantages and disadvantages of P2P systems - common problems and operations in P2P systems - routing in structured and unstructured overlay networks, in particular distributed hash tables - network address translation and traversal - decentralized network size estimation - reputation systems - common attacks on P2P protocols and defenses against them - anonymity, protocols to create anonymity and attacks on anonymity The various protocols are described in theory and practical implementations of those ideas are discussed (in particular Freenet, Bittorrent, Tor, JAP, GNUnet, I2P and Gnutella). The course will also include a practical component focusing on the design and implementation of a new P2P protocol or application with design choices being discussed in class

Teaching and Learning Methods

Lecture for content transfer with accompanying weekly tutorial exercises, in which tasks are treated, as well as programming tasks for computer-aided deepening.

SECURITY ENGINEERING

General competences

Description of Achievement and Assessment Methods

75 minutes written exam, closed book examination

Intended Learning Outcomes

At the end of the module students are able to understand and apply processes and techniques required for analysis, design, evaluation and operation of secure IT applications, as well as for design, implementation and verification of secure software. Thus students are enabled to analyze and assess the IT security of applications, and to develop and represent systematically appropriate IT security solutions.

Content

A secure IT application requires both presence of desired security features (example: access control), and absence of undesired security flaws (example: processing of un-validated user input). We discuss basic terms (example: security versus safety), and security goals as well as secure design principles (example: least privilege). We get familiar with the engineering process for security critical applications. We focus on system analysis, threat and risk analysis, derivation of security requirements, and recommendation of potential security controls, also taking into account economical considerations. We learn how to specify security policies and how these policies reflect in a security architecture in the context of an overall system architecture. We exemplary take into account basic security controls (example: PKI). We address methods to validate and assess secure applications (example: Common Criteria) and deal with concepts for IT security management (example: ISO 2700x) and secure operation of applications. A further focal point is the development of secure software. We talk about basic attack vectors of hackers, crackers and malware authors. We experience, how insecure programming opens up vulnerabilities (example: cross site scripting). We learn how to detect such vulnerabilities in a systematical manner. We investigate the professional handling of security incidents and the work of Computer Emergency Response Teams (CERTs). We address principles of and guidelines for secure software design and implementation, as well as corresponding verification techniques. We exemplary take a closer look at Microsoft's Security Development Cycle, and get to know models to assess and improve the quality of a secure software development process (example: SSE-CMM).

A secure IT application requires both presence of desired security features (example: access control), and absence of undesired security flaws (example: processing of un-validated user input). We discuss basic terms (example: security versus safety), and security goals as well as secure design principles (example: least privilege). We get familiar with the engineering process for security critical applications. We focus on system analysis, threat and risk analysis, derivation of security requirements, and recommendation of potential security controls, also taking into account economical considerations. We learn how to specify security policies and how these policies reflect in a security architecture in the context of an overall system architecture. We exemplary take into account basic security controls (example: PKI). We address methods to validate and assess secure applications (example: Common Criteria) and deal with concepts for IT security management (example: ISO 2700x)

and secure operation of applications. A further focal point is the development of secure software. We talk about basic attack vectors of hackers, crackers and malware authors. We experience, how insecure programming opens up vulnerabilities (example: cross site scripting). We learn how to detect such vulnerabilities in a systematical manner. We investigate the professional handling of security incidents and the work of Computer Emergency Response Teams (CERTs). We address principles of and guidelines for secure software design and implementation, as well as corresponding verification techniques. We exemplary take a closer look at Microsoft's Security Development Cycle, and get to know models to assess and improve the quality of a secure software development process (example: SSE-CMM).

Teaching and Learning Methods

Combintion of lecture, case studies (partly in team work), exercises, and home work for individual study. Students are encouraged to study literature, to examine the addressed subjects and to analyze related current news. By applying the lecture contents to case studies the mediated theoretical foundations are substantiated via practical tasks.

ALGORITHMS FOR SCIENTIFIC COMPUTING

General competences

Description of Achievement and Assessment Methods

Type of Assessment: The exam takes the form of a 120 minutes written exam. The examination consists of a written exam of 120 minutes in which students show that they are able to find solutions for algorithmic problems arising in the field of scientific computing in a limited time. Questions and small tasks concerning code are used to test the student´s knowledge on known and related hierarchical methods as well as their ability to implement them. Example algorithms and questions are used to examine the capability of analyzing the efficiency of algorithms and the accuracy of a given method.

Exam retake next semester

No

Exam retake at the end of semester

Yes

Description

Prerequisites (recommended)

-

Intended Learning Outcomes

At the end of the module, students are able to identify, explain, and implement selected hierarchical methods that are of particular interest to the informatical aspects of scientific computing because of their algorithmic structure and their significance for practical applications. Participants can analyse and judge the efficiency of such methods by deriving statements about the required computational cost and - where applicable - the achieved accuracy and by comparing them with corresponding results for other methods. The students are able to transfer the methodology to new methods for related problems.

Content

Discrete Fourier transform and related transforms - FFT: derivation and implementation - Fast discrete cosine/sine transforms: derivation and implementation via FFT - Applications: multi-dimensional data (images, video, audio) and FFT-based solvers for linear systems of equations Space-filling curves - Peano- and Hilbert curves: representation by algebraic and grammatical means - Applications: organisation of multi-dimensional data; parallel, adaptive, and cache oblivious algorithms Hierarchical numerical methods - Hierarchical bases for one- and multi-dimensional problems - Computational cost versus accuracy; Sparse Grids - Adaptive representation of continuous data - Applications: numerical quadrature, differential equations - Outlook: multigrid methods, wavelets

Discrete Fourier transform and related transforms - FFT: derivation and implementation - Fast discrete cosine/sine transforms: derivation and implementation via FFT - Applications: multi-dimensional data (images, video, audio) and FFT-based solvers for linear systems of equations Space-filling curves - Peano- and Hilbert curves: representation by algebraic and

grammatical means - Applications: organisation of multi-dimensional data; parallel, adaptive, and cache oblivious algorithms Hierarchical numerical methods - Hierarchical bases for one- and multi-dimensional problems - Computational cost versus accuracy; Sparse Grids - Adaptive representation of continuous data - Applications: numerical quadrature, differential equations - Outlook: multigrid methods, wavelets

Teaching and Learning Methods

This module comprises lectures and accompanying tutorials. The contents of the lectures will be taught by talks and presentations. Students will be encouraged to study literature and to get involved with the topics in depth. In the tutorials, concrete problems will be solved - partially in teamwork - and selected examples will be discussed.

ALGORITHMS FOR SCIENTIFIC COMPUTING II

General competences

Description of Achievement and Assessment Methods

Type of Assessment: written exam (90 minutes) In the exam students should prove to be able to identify a given problem and find solutions within limited time. The examination will completely cover the content of the lectures. The answers will require own formulations. In addition, questions requiring short calculations may be posed. Exam questions evaluate the participants' capability to adapt numerical methods to ambitious applications from the practice of scientific computing involving advanced problems and methods in scientific computing. Exam tasks test the students' knowledge on hierarchical aspects.

Exam retake next semester

No

Exam retake at the end of semester

Yes

Description

Prerequisites (recommended)

-

Intended Learning Outcomes

At the end of the module, students are able to adapt numerical methods to ambitious applications from the practice of scientific computing. They understand and are familiar with advanced problems and methods in scientific computing. These advanced methods comprise in particular hierarchical aspects that students are able to understand, explain and apply to more complex applications.

Content

Courses of this module exist in three different specifications: - Sparse Matrices (Huckle): ++ Data structures, Sparsity Graphs ++ Sparse BLAS ++ Reordering ++ Direct solution of linear equations and algebraic pivoting ++ Iterative solution of linear equations and preconditioning. - Scientific Computing (Mendl): ++ Introduction to quantum mechanics ++ Bell inequalities ++ Quantum circuits and algorithms ++ Quantum error correction ++ Mapping of NP problems to quantum ground states ++ Numerical methods for quantum systems - Scientific Computing (Bungartz): ++ Molecular dynamics simulation ++ Sparse grid methods ++ Algebraic multigrid methods

Courses of this module exist in three different specifications: - Sparse Matrices (Huckle): ++ Data structures, Sparsity Graphs ++ Sparse BLAS ++ Reordering ++ Direct solution of linear equations and algebraic pivoting ++ Iterative solution of linear equations and preconditioning. - Scientific Computing (Mendl): ++ Introduction to quantum mechanics ++ Bell inequalities ++ Quantum circuits and algorithms ++ Quantum error correction ++ Mapping of NP problems to quantum ground states ++ Numerical methods for quantum

systems - Scientific Computing (Bungartz): ++ Molecular dynamics simulation ++ Sparse grid methods ++ Algebraic multigrid methods

Teaching and Learning Methods

This module comprises lectures and accompanying tutorials. The contents of the lectures will be taught by talks and presentations. Students will be encouraged to study literature and to get involved with the topics in depth. In the tutorials, concrete problems will be solved - partially in teamwork - and selected examples will be discussed.

INTRODUCTION TO QUANTUM COMPUTING

General competences

Description of Achievement and Assessment Methods

The assessment is by means of a written exam of 90 minutes. Allowed tools: one DIN-A4 sheet (both sides) with own notes; can be handwritten or prepared digitally and printed. Problems related to the fundamental mathematical formalism may ask the students to compute the quantum mechanical state vector after passing through an experimental setup, or the derivation of a mathematical relation. Problems on quantum circuits test to what degree students have understood the individual building blocks (e.g., Pauli matrices or Hadamard gates), and can use these for the design of (simple) circuits. The exam assesses the knowledge of quantum algorithms for example by asking the students to analyze the number of required operations. Reaching a pre-specified number of the maximum homework points can be rewarded by a grade bonus in the evaluation of the exam. The exact details are timely announced at the beginning of the course.

Exam retake next semester

No

Exam retake at the end of semester

Yes

Description

Prerequisites (recommended)

Linear Algebra, e.g., MA0901 Linear Algebra for Informatics

Intended Learning Outcomes

After successful completion of this module, students are familiar with the fundamentals and the mathematical formalism of quantum computing. They can analyse quantum algorithms, like Grover search, and construct quantum circuits for simple algorithms. The students can apply software tools (like Qiskit or Cirq), and can evaluate possible fields of applications of (future) quantum computers.

Content

- Mathematical formalism and foundations of quantum mechanics - Quantum circuits - Quantum teleportation - Bell inequality - Fundamental quantum algorithms

- Mathematical formalism and foundations of quantum mechanics - Quantum circuits - Quantum teleportation - Bell inequality - Fundamental quantum algorithms

Teaching and Learning Methods

The whiteboard lectures convey the fundamentals and the mathematical formalism of quantum computing in-depth, and the slide presentations illustrate state-of-the art technical developments. The accompanying exercises for individual study deepen the understanding of the topics explained in the lecture, and foster the creative application of the learnt techniques.

ADVANCED CONCEPTS OF QUANTUM COMPUTING

General competences

Description of Achievement and Assessment Methods

The assessment is by means of a written exam of 90 minutes. Allowed tools: one DIN-A4 sheet (both sides) with own notes; can be handwritten or prepared digitally and printed. Problems related to quantum operations may ask the students to derive such an operation from a quantum circuit with principal and auxiliary qubits, the identification of a phase estimation step in a circuit, or using quantum Fourier transformation to solve a specific task. Problems on quantum error-correction and the stabilizer formalism might ask students to prove commutation relations, identity the subspace stabilized by a subgroup of the Pauli group, or transform such a subspace after conjugation by unitary gates. Reaching a pre-specified number of the maximum homework points can be rewarded by a grade bonus in the evaluation of the exam. The exact details are timely announced at the beginning of the course.

Exam retake next semester

No

Exam retake at the end of semester

Yes

Description

Prerequisites (recommended)

- Introduction to Quantum Computing (IN2381) - Linear Algebra, e.g., MA0901 Linear Algebra for Informatics - Analysis (for Quantum Fourier transform), e.g., MA0902 Analysis for Computer Science - Fundamentals of group theory helpful (for quantum error correction), but not strictly required, e.g., MA2010 Algebra

Intended Learning Outcomes

After successful completion of this module, students are familiar with advanced concepts and algorithms related to quantum computing, in particular the quantum Fourier transformation, quantum operations and quantum error correction. They can differentiate the quantum Fourier transform from other algorithms, understand its relevance for integer factorization, and apply the quantum Fourier transform in new scenarios. The students also understand the mathematical formalism of quantum error correction and related concepts like the stabilizer formalism.

Content

- Quantum Fourier transform - Shor's algorithm for integer factorization - Quantum operations - Quantum error-correction

- Quantum Fourier transform - Shor's algorithm for integer factorization - Quantum operations - Quantum error-correction

Teaching and Learning Methods

The whiteboard lectures convey the advanced concepts of quantum computing in-depth, and the slide presentations illustrate state-of-the art technical developments. The accompanying exercises for individual study deepen the understanding of the topics explained in the lecture, and foster the creative application of the learnt techniques.

PARALLEL PROGRAM ENGINEERING

General competences

Description of Achievement and Assessment Methods

In the written 75 minutes or 20 minutes oral exam, students should prove their knowledge of concepts, techniques and tools. They should show their ability to apply the knowledge to outline solutions for tasks in the software development process of parallel applications. The examination will cover the content of the lectures. The answers will require own formulations.

Exam retake next semester

No

Exam retake at the end of semester

No

Description

Prerequisites (recommended)

IN2076 Advanced Computer Architecture, IN2147 Parallel Programming

Intended Learning Outcomes

The participants will be able to apply a structured approach in the development of parallel applications. They are able to assess and apply existing tools for a given problem. They understand tool development infrastructures and are able to design and implement own solutions to issues in the development process of parallel software.

Content

This module starts with a short introduction to parallel programming, programming models and languages, applications, and the parallelization approach. It presents the software development process focusing on aspects and requirements for parallel and high performance applications. Development environments supporting the orchestration of parallel programs are introduced. The module then concentrates on concepts and tools for engineering parallel programs focusing on: e.g. omain decomposition, Debugging, Performance analysis, Performance modeling and prediction, Application tuning, Performance engineering workflows, and tool development infrastructures.

This module starts with a short introduction to parallel programming, programming models and languages, applications, and the parallelization approach. It presents the software development process focusing on aspects and requirements for parallel and high performance applications. Development environments supporting the orchestration of parallel programs are introduced. The module then concentrates on concepts and tools for engineering parallel programs focusing on: e.g. omain decomposition, Debugging, Performance analysis, Performance modeling and prediction, Application tuning, Performance engineering workflows, and tool development infrastructures.

Teaching and Learning Methods

This module comprises lectures and accompanying project work. The lectures will be based on presentations. Interactive brainstorming elements will be integrated. Students will be encouraged to study literature based on the given books and available research papers. Participants will apply the concepts learned based on real tools in multiple small projects executed in teamwork in the exercises.

PARALLEL PROGRAMMING

General competences

Description of Achievement and Assessment Methods

The exam takes the form of a 120 minutes written test. Questions allow to asses acquaintance with the concepts of parallel programming models, languages, and tools. Code snippets of sequential and parallel programs are given. Students apply their knowledge on dependence analysis and code transformations to these codes. Based on code snippets the students apply the learned parallel models to demonstrate their ability to evaluate different parallelization strategies, to parallelize code, and tune applications.

Exam retake next semester

No

Exam retake at the end of semester

Yes

Description

Prerequisites (recommended)

IN2076 Advanced Computer Architecture

Intended Learning Outcomes

At the end of the module students are able to create parallel programs in MPI and OpenMP. They understand the performance aspects of different parallelization strategies and can evaluate those parallelization strategies in the context of applications. They are able to apply data dependence analysis and program transformations. They can analyze and tune the performance of parallel applications.

Content

The module starts with a motivation for parallel programming and a classification of parallel architectures. It focuses first on parallelization for distributed memory architectures with MPI. It introduces the major concepts, e.g., point to point communication, collective operations, communicators, virtual topologies, non-blocking communication, single-sided communication and parallel IO. In addition, it covers the overall parallelization approach based on four phases, i.e., decomposition, assignment, orchestration and mapping. The next section presents dependence analysis as the major theoretical basis for parallelization. It introduces program transformations and discusses their profitability and safety based on data dependence analysis. The second major programming interface in the module is OpenMP for shared memory systems. This section covers most of the language concepts as well as proposed extensions. In the last part, the module presents novel programming interfaces, such as PGAS languages, threading building blocks, CUDA, OpenCL, and OpenACC.

The module starts with a motivation for parallel programming and a classification of parallel architectures. It focuses first on parallelization for distributed memory architectures with MPI. It introduces the major concepts, e.g., point to point communication, collective operations, communicators, virtual topologies, non-blocking communication, single-sided

communication and parallel IO. In addition, it covers the overall parallelization approach based on four phases, i.e., decomposition, assignment, orchestration and mapping. The next section presents dependence analysis as the major theoretical basis for parallelization. It introduces program transformations and discusses their profitability and safety based on data dependence analysis. The second major programming interface in the module is OpenMP for shared memory systems. This section covers most of the language concepts as well as proposed extensions. In the last part, the module presents novel programming interfaces, such as PGAS languages, threading building blocks, CUDA, OpenCL, and OpenACC.

Teaching and Learning Methods

The different parallel programming models and parallelization techniques are introduced in the lecture. Voluntary short student presentations demonstrate the techniques in application areas. Within a central exercise session, assignments are presented and discussed. The students solve the assignments and submit the solutions which are checked for correctness. In the assigments the students apply the learned concepts to larger example programs.

TENSOR NETWORKS

General competences

Description of Achievement and Assessment Methods

The assessment is by means of a written exam of 90 minutes, or a 20 minute oral examination. Problems on the underlying approximation theory test whether students can apply the acquired knowledge and methods to a new domain. Problems related to the graphical representation may ask students to express an algorithm stated in matrix notation into a graphical form. The exam assesses the knowledge of tensor network methods for simulating quantum systems for example by asking them to tailor an algorithm for a given quantum Hamiltonian. Reaching a certain fraction of the maximum number of homework points can be rewarded by a grade step bonus (“eine Notenstufe”) in the evaluation of the exam. The exact details are timely announced at the beginning of the lecture.

Exam retake next semester

Yes

Exam retake at the end of semester

No

Description

Prerequisites (recommended)

• MA0901 Linear Algebra for Informatics • MA0902 Analysis for Informatics • IN0018 Discrete Probability Theory • Knowledge about quantum mechanics or computing helpful (but not a formal prerequisite)

Intended Learning Outcomes

After successful completion of this module, students are familiar with the mathematical formalism and graphical notation for tensor networks. They can assess and apply tensor network approaches for approximating high-dimensional data. They understand why tensor network methods are suitable for simulating strongly correlated quantum systems, and are familiar with corresponding algorithms.

Content

- Fundamentals and graphical representation of tensor networks - Mathematical approximation theory - Backpropagation through tensor network operations - Simulating strongly correlated quantum systems and digital quantum computers - Probability distribution sampling using tensor networks

- Fundamentals and graphical representation of tensor networks - Mathematical approximation theory - Backpropagation through tensor network operations - Simulating strongly correlated quantum systems and digital quantum computers - Probability distribution sampling using tensor networks

Teaching and Learning Methods

The whiteboard lectures convey the mathematical formalism and graphical representation of tensor network methods and algorithm in-depth. The accompanying exercises for individual study deepen the understanding of the topics explained in the lecture, and foster the creative application of the learnt techniques.

TIME INTEGRATION AND DIFFERENTIAL EQUATIONS

General competences

Description of Achievement and Assessment Methods

Depending on course size either oral exam (30min per student) or 60min written exam. Both options are without additional aids and result in one grade for the course. Questions allow to assess the student's ability to understand the fundamental concepts and interplay of time and space discretization methods to develop solvers for time dependent partial differential equations.

Exam retake next semester

No

Exam retake at the end of semester

Yes

Description

Prerequisites (recommended)

MA0902 Analysis for Informatics Basic knowledge on numerics (e.g. via course on numerical programming), Basic Python programming skills for the tutorials

Intended Learning Outcomes

Upon completion of the module students will be able to develop solvers for initial value problems of certain classes of partial differential equations. Furthermore, they will be able to evaluate some of them regarding various numerical properties. They will be able to understand the interplay between time and various space discretizations. Finally, they will be able to apply advanced time integration methods.

Content

- Basics -- Basics of ordinary differential equations -- Basics of standard time integration (Runge-Kutta) -- Dispersion properties -- Splitting methods - Partial differential equations (PDEs) -- Classification -- Analytical properties and solutions - Discretization of spatial dimension - Discretization of PDE operators - DE solvers on high-performance computers - Advanced time integration (examples) -- Semi-Lagrangian Methods -- Spectral Deferred Correction Methods -- Exponential time integration -- Parareal -- Parallel Full Approximation Scheme in Space and Time (PFASST)

- Basics -- Basics of ordinary differential equations -- Basics of standard time integration (Runge-Kutta) -- Dispersion properties -- Splitting methods - Partial differential equations (PDEs) -- Classification -- Analytical properties and solutions - Discretization of spatial dimension - Discretization of PDE operators - DE solvers on high-performance computers - Advanced time integration (examples) -- Semi-Lagrangian Methods -- Spectral Deferred Correction Methods -- Exponential time integration -- Parareal -- Parallel Full Approximation Scheme in Space and Time (PFASST)

Teaching and Learning Methods

Lecture, Tutorials, Assignments for self-study and consolidation

AUGMENTED REALITY

General competences

Description of Achievement and Assessment Methods

The examination takes the form of a 90-minute written examination. Half of this refers to theoretical concepts, and the other half to abstracted (generalized) characteristics and experiences with the tools and practices used or developed in the exercises and homework. developed in the exercises and homework. The questions will be asked in English; they can be answered in German or English. No lecture notes are permitted during the exam; a calculator and a dictionary (English mother tongue) may be used as aids. Knowledge questions test familiarity with the basic concepts of augmented reality (related to the theoretical and overview sections). Short programming tasks, questions on short program examples, and questions on programming concepts test the ability to handle an AR system and to understand and expand small systems (related to the practical exercise part). Successfully completing of programming tasks is assessed as a graded project as a voluntary mid-term achievement. The tasks are distributed throughout the semester. The points achieved are converted into a grade according to the maximum number of points. If the calculation of 1/3 voluntary mid-term performance and 2/3 of the exam grade results in an improvement, this is accepted as the module grade.

Exam retake next semester

No

Exam retake at the end of semester

Yes

Description

Prerequisites (recommended)

IN0001 Introduction to Informatics, IN0006 Introduction to Software Engineering, IN0007 Fundamentals of Algorithms and Data Structures, MA0901 Linear Algebra for Informatics, Basic mathematical and programming skills.

Intended Learning Outcomes

At the end of the module students are able to describe and analyze the underlying mathematical and programming aspects (tracking and rendering) of AR systems, and they can extend them towards own, further-reaching solutions. They can also evaluate existing input and output devices regarding their suitability for special application scenarios.

Content

Foundations of Augmented Reality - Geometric transformations (homogeneous matrices, projective geometry, scene graph) - Three-dimensional information presentation (OpenGL, 3D APIs) - Head-mounted displays and other display devices - Physical foundations of several tracking devices - Mathematical foundations of optical tracking, sensor fusion and device calibration - Concepts: augmented reality in comparison with mixed reality, ubiquitous computing, etc.

Foundations of Augmented Reality - Geometric transformations (homogeneous matrices, projective geometry, scene graph) - Three-dimensional information presentation (OpenGL, 3D APIs) - Head-mounted displays and other display devices - Physical foundations of several tracking devices - Mathematical foundations of optical tracking, sensor fusion and device calibration - Concepts: augmented reality in comparison with mixed reality, ubiquitous computing, etc.

Teaching and Learning Methods

Lecture, exercise, problems for individual study. The assignments are provided on a weekly basis via the teaching portal. They are discussed in the next exercise class, and a solution is presented. Work on the assignments and participation in the exercise class are voluntary. They serve as a means for students to deepen and test their acquired knowledge – as a self-monitoring aid to prepare for the written exam. In the second half of the semester, students can participated in a team project (final project) in which they can apply and discuss the learned concepts and methods in a freely selectable application context. Successful presentation of a demonstrator yields a bonus.

ELEARNING - TECHNIQUES AND INFRASTRUCTURES

General competences

Description of Achievement and Assessment Methods

The examination is taken in oral form (20 minutes). The students have to show that they master the functionalities, architecture and implementation of eLearning and campus management systems and they can write simple XML programs.

Exam retake next semester

No

Exam retake at the end of semester

No

Description

Prerequisites (recommended)

IN2038 Distributed and Web Information Systems

Intended Learning Outcomes

After attending the module, students are able to oversee te state of the art in eLearning. They can judge the value of Learning/Teaching Environments and evaluate the functionality, architecture and implementation of eLearning environments. They understand the basic technical embedding of eLearning systems into the general functions of campus management systems. Basic skills in programming XML are acquired and can be applied to the definition of eLearning programs.

Content

- Introduction to eLearning: - eLearning in Universities - Pedagogy and didactics: learning process, elements of eLearning, design and structure. - eLearning concepts, systems and services - eLearning architectures: CSCW, eLearning portals, Learning Management Systems, Content Management Systems - Tools: Authoring tools - eLearning integration into University information management systems: User management (LDAP), campus management - Production of eLearning content with XML: eLearning XML languages, transformation of XML documents - Standards and integration: Learning Technology Standards, eLearning metadata, LMS integration - Knowledge Management

- Introduction to eLearning: - eLearning in Universities - Pedagogy and didactics: learning process, elements of eLearning, design and structure. - eLearning concepts, systems and services - eLearning architectures: CSCW, eLearning portals, Learning Management Systems, Content Management Systems - Tools: Authoring tools - eLearning integration into University information management systems: User management (LDAP), campus management - Production of eLearning content with XML: eLearning XML languages, transformation of XML documents - Standards and integration: Learning Technology Standards, eLearning metadata, LMS integration - Knowledge Management

Teaching and Learning Methods

lecture

FUNDAMENTALS OF HUMAN-CENTERED ROBOTICS

General competences

Description of Achievement and Assessment Methods

The module grade is based on the student's performance in a presentation, a demonstration, as well as a final oral exam (approx. 10 min). This evaluates the students' knowledge in theoretical basics of human-centered robotics and examines whether they are able to apply this to real world problems and present this result as well as demonstrate.

Exam retake next semester

Yes

Exam retake at the end of semester

No

Description

Prerequisites (recommended)

Control Systems 1 Robotics

Intended Learning Outcomes

Upon completion of this module, students will be able to understand and evaluate the various methods and technologies of safe physical human-robot interaction. They know how to apply the theoretical principles of robot control for safe human-robot interaction. They will be able to develop robot programs based on the latest methods for global motion planning in dynamic environments. In addition, they have an overview of motion planning algorithms for collision avoidance and can implement them practically in robotics. Starting from a specific problem, they can develop safe and human-friendly programs for physical human-robot interaction and implement and evaluate them in practice.

Content

The Human-Centered Robotics lecture covers a thorough introduction into the technology and intelligent control of collaborative robotic systems. The theoretical foundation and potential applications of interacting robots are discussed in depth. Tutorials deepen conceptual understanding of interactive robotics and corresponding software tools. The students will learn the basics of robot safety, human-friendly AI-based task and path planning in dynamic environments. To gain practical experience and apply the course concepts in cutting-edge robot technology, Robothons are offered that aim to solve real-world challenges with highly motivated student teams consisting of 4-6 members from different disciplines. The course outline is illustrated below. I. Introduction - A short history of robotics - What is Human-Friendly Robotics? II. Mechanical design and dynamic modelling for safety and performance - Rigid robots - Robots with flexible elements - Intrinsically compliant robots - Safety for physical Human-Robot Interaction III. Control methods for unknown environments and Human-Robot Interaction - Perception and actuation for interactive robotics - Motion control - Force and impedance control - Collision handling IV. Real-time motion planning - Real-time trajectory planning - Environment sensing and perception - Real-time motion planning and obstacle avoidance behavior - Hierarchical motion control Exercises will focus

on following topics: - Robot Modeling – Introduction into the modeling of rigid and flexible joint robots - Robot Control – Introduction into applied robot control in dynamic environments focusing on human safety - Motion Planning – Design and execution of safe dynamic trajectory planning During the Robothon, students will gain hands-on experience in solving real-world problems using robotic systems. Students will work together in interdisciplinary teams, where each team selects a desired robotic task to implement, with a focus on human-robot collaboration. The students are responsible for managing the entire project, creating the project plan, executing the project in an interdisciplinary team, finishing the project on time, and demonstrating their results to the course participants. For project examples from previous Robothons in Hannover, see the following: https://www.roboterfabrik.uni-hannover.de

The Human-Centered Robotics lecture covers a thorough introduction into the technology and intelligent control of collaborative robotic systems. The theoretical foundation and potential applications of interacting robots are discussed in depth. Tutorials deepen conceptual understanding of interactive robotics and corresponding software tools. The students will learn the basics of robot safety, human-friendly AI-based task and path planning in dynamic environments. To gain practical experience and apply the course concepts in cutting-edge robot technology, Robothons are offered that aim to solve real-world challenges with highly motivated student teams consisting of 4-6 members from different disciplines. The course outline is illustrated below. I. Introduction - A short history of robotics - What is Human-Friendly Robotics? II. Mechanical design and dynamic modelling for safety and performance - Rigid robots - Robots with flexible elements - Intrinsically compliant robots - Safety for physical Human-Robot Interaction III. Control methods for unknown environments and Human-Robot Interaction - Perception and actuation for interactive robotics - Motion control - Force and impedance control - Collision handling IV. Real-time motion planning - Real-time trajectory planning - Environment sensing and perception - Real-time motion planning and obstacle avoidance behavior - Hierarchical motion control Exercises will focus on following topics: - Robot Modeling – Introduction into the modeling of rigid and flexible joint robots - Robot Control – Introduction into applied robot control in dynamic environments focusing on human safety - Motion Planning – Design and execution of safe dynamic trajectory planning During the Robothon, students will gain hands-on experience in solving real-world problems using robotic systems. Students will work together in interdisciplinary teams, where each team selects a desired robotic task to implement, with a focus on human-robot collaboration. The students are responsible for managing the entire project, creating the project plan, executing the project in an interdisciplinary team, finishing the project on time, and demonstrating their results to the course participants. For project examples from previous Robothons in Hannover, see the following: https://www.roboterfabrik.uni-hannover.de

Teaching and Learning Methods

This course consists of lectures, theoretical exercises, and a project. Exercises are made available, presented and discussed alongside the lectures. Sample solutions as well as some papers for reading are provided. The Robothon introduces the students to real world problems that should be solved with real state-of-the-art robots in the form of a collaborative and interdisciplinary project. This combines the knowledge from the lectures with real world applications and the proficient use of modern robotic technology.

3D USER INTERFACES

General competences

Description of Achievement and Assessment Methods

The exam takes the form of a 90-minute written test. Half of the exam focuses on theoretical concepts, and the other half covers tools and approaches that were learned and used in the exercises and assignments (practical part). Questions are posed in English; answers can be given also in German. No lecture notes are permitted during the exam; a calculator and a dictionary (English mother tongue) may be used as aids. Questions allow for the assessment of acquaintance with principal concepts of three-dimensional user interfaces (w.r.t. to theoretical part and overview part). Short programming tasks and questions about program examples and three-dimensional user interfaces, as well as programming concepts, assess the ability to work with three-dimensional user interfaces and the ability to understand and extend small systems (w.r.t. the practical part). Successfully completing of programming tasks is assessed as a graded project as a voluntary mid-term achievement. The tasks are distributed throughout the semester. The points achieved are converted into a grade according to the maximum number of points. If the calculation of 1/3 voluntary mid-term performance and 2/3 of the exam grade results in an improvement, this is accepted as the module grade.

Exam retake next semester

No

Exam retake at the end of semester

Yes

Description

Prerequisites (recommended)

IN0001 Introduction to Informatics, IN0006 Introduction to Software Engineering, IN0007 Fundamentals of Algorithms and Data Structures, MA0901 Linear Algebra for Informatics

Intended Learning Outcomes

Students receive in-detail knowledge on the advance conception and implementation of 3D user interfaces in interactive, immersive computer applications. At the end of the module they can analyze current concepts of three-dimensional interaction regarding their requirements on input and output devices and with respect to their functionality and usability. Furthermore, they can create and implement novel interaction concepts.

Content

In-depth coverage of concepts of three-dimensional human computer interaction - Current input and output devices - Interaction techniques: selection, manipulation, travel, wayfinding, system control - Evaluation of 3D user interfaces - Future 3D user interfaces

In-depth coverage of concepts of three-dimensional human computer interaction - Current input and output devices - Interaction techniques: selection, manipulation, travel, wayfinding, system control - Evaluation of 3D user interfaces - Future 3D user interfaces

Teaching and Learning Methods

Lecture, exercise, problems for individual study. The assignments are provided on a weekly basis via the teaching portal. They are discussed in the next exercise class, and a solution is presented. Work on the assignments and participation in the exercise class are voluntary. They serve as a means for students to deepen and test their acquired knowledge – as a self-monitoring aid to prepare for the written exam. In the second half of the semester, students can participate in a team project (final project) in which they can apply and discuss the learned concepts and methods in a freely selectable application context. The successful creation and presentation of a demonstrator as a final project is rewarded with a bonus.

CONCEPTS OF C++ PROGRAMMING

General competences

Description of Achievement and Assessment Methods

The exam consists of a computer-based test under supervision after the lecture period. Optional homework submissions can be used as a 0,3 grade bonus. For the homework, 1–2 graded weekly programming assignments will be given throughout the course, which participants have to solve and submit in electronic form. This ensures that participants understood the relevant concepts and can apply their knowledge successfully on programming tasks by writing appropriate C++ code. By building on results of previous tasks and working with non-trivial existing code bases, students show their ability to select and implement appropriate concepts and patterns in complex contexts. The 90-minutes computer-based test consists of questions, where the students are expected to show that they have profound knowledge of the C++ programming concepts covered in the module, and short programming tasks, where students prove that they are able to recognize and analyze problems and use C++ to find elegant and efficient solutions within a limited amount of time.

Exam retake next semester

No

Exam retake at the end of semester

Yes

Description

Prerequisites (recommended)

IN0001: Introduction to Informatics IN0002: Fundamentals of Programming (Exercises & Laboratory) IN0007: Fundamentals of Algorithms and Data Structures

Intended Learning Outcomes

The participants master the concepts of C++ programming mentioned below. They are able to independently analyze them and to apply the corresponding analysis to related programming problems. Furthermore, the participants are able to use the handled programming concepts, to modify them if necessary, and to compare the quality of different solutions.

Content

First, the module studies the basic concepts of the C++ language (syntax, strong typing, type deduction, focus on runtime efficiency). Then, the module studies the central, modern C++ programming concepts. - Concepts for efficient resource management: The module presents RAII, smart pointers, universal references, ownership and copy/move. - Concepts for procedural programming: The module investigates C++ mechanisms for procedural programming, such as functions, parameter passing, lambdas, overloads, and error handling. - Concepts for object-oriented programming: The module presents classes, inheritance (single and multiple), polymorphy, and RTTI. - Concepts for generic programming: The module investigates templates, variadic templates and fold expressions, expression

templates for lazy evaluation, and typical patterns such as CRTP. - Concepts of compile-time programming: The module investigates concepts such as template recursion, constexpr, and type traits. - Concepts for containers, iterators, and ranges: The module presents the STL standard containers, algorithms and runtime guarantees, iterator concepts as well as views and ranges. - Concepts for build systems and dependency management: concepts for automatic compilation, linking and management of dependencies are studied. Additionally, concepts for continuous testing and integration are investigated. - Optionally, concepts for parallel programming are investigated, such as threads, atomics and async/futures, as well as recent C++ extensions such as concepts and modules.

First, the module studies the basic concepts of the C++ language (syntax, strong typing, type deduction, focus on runtime efficiency). Then, the module studies the central, modern C++ programming concepts. - Concepts for efficient resource management: The module presents RAII, smart pointers, universal references, ownership and copy/move. - Concepts for procedural programming: The module investigates C++ mechanisms for procedural programming, such as functions, parameter passing, lambdas, overloads, and error handling. - Concepts for object-oriented programming: The module presents classes, inheritance (single and multiple), polymorphy, and RTTI. - Concepts for generic programming: The module investigates templates, variadic templates and fold expressions, expression templates for lazy evaluation, and typical patterns such as CRTP. - Concepts of compile-time programming: The module investigates concepts such as template recursion, constexpr, and type traits. - Concepts for containers, iterators, and ranges: The module presents the STL standard containers, algorithms and runtime guarantees, iterator concepts as well as views and ranges. - Concepts for build systems and dependency management: concepts for automatic compilation, linking and management of dependencies are studied. Additionally, concepts for continuous testing and integration are investigated. - Optionally, concepts for parallel programming are investigated, such as threads, atomics and async/futures, as well as recent C++ extensions such as concepts and modules.

Teaching and Learning Methods

The module consists of lectures with integrated exercises and tutorial sessions. The content of the lectures is conveyed in presentations of the scientific material. The lectures include short exercises to encourage active participation. By solving homework assignments, the students are encouraged to work intensively on the respective topics. The solutions of the assignments are discussed in the tutorial sessions.

LECTURE SERIES: WHAT IS DIGITALIZATION?

General competences

Description of Achievement and Assessment Methods

Written exam, 60 minutes Students have to write an essay about a provided topics that covers one or the connection of various topics from the lecture. Different emphases are possible: analyze, argue, assess, prove or characterize, define, explain and compare. The essay is assessed as "passed" or "failed".

Exam retake next semester

No

Exam retake at the end of semester

Yes

Description

Prerequisites (recommended)

Basic understanding of the fundamentals of informatics

Intended Learning Outcomes

After attending this class, students can understand, analyze and contextualize different facets of digitalization at the level of university education. They can identify trade-offs and conflicting goals.

Content

Aspects of digitalization from the perspective of different disciplines: informatics, information systems, law, economics, politics, sociology, philosophy and ethics, security and privacy, and from the perspective of industrial applications.

Aspects of digitalization from the perspective of different disciplines: informatics, information systems, law, economics, politics, sociology, philosophy and ethics, security and privacy, and from the perspective of industrial applications.

Teaching and Learning Methods

Presentations with slides due to the nature of this class with speakers from different disciplines as well as from theory and practice.