SYLLABUS

1. Information regarding the programme

1.1 Higher education institution	**Babes-Bolyai University**
1.2 Faculty	**Faculty of Mathematics and Computer Science**
1.3 Department	**Department of Computer Science**
1.4 Field of study	**Computer Science**
1.5 Study cycle	**Bachelor**
1.6 Study programme / Qualification	**Computer Science**

2. Information regarding the discipline

2.1 Name of the discipline				**Virtual Machines: Design and Implementation**			
2.2 Course coordinator				**Assoc. Prof. Ing. Florin Craciun**			
2.3 Seminar coordinator				**Assoc. Prof. Ing. Florin Craciun**			
2.4. Year of study	**3**	2.5 Semester	**6**	2.6. Type of evaluation	**E**	2.7 Type of discipline	**Optional**

3. Total estimated time (hours/semester of didactic activities)

3.1 Hours per week	3	Of which: 3.2 course	2	3.3 seminar/laboratory	1
3.4 Total hours in the curriculum	36	Of which: 3.5 course	24	3.6 seminar/laboratory	12
Time allotment:					hours
Learning using manual, course support, bibliography, course notes					20
Additional documentation (in libraries, on electronic platforms, field documentation)					10
Preparation for seminars/labs, homework, papers, portfolios and essays					88
Tutorship					10
Evaluations					10
Other activities:					-

3.7 Total individual study hours	138
3.8 Total hours per semester	175
3.9 Number of ECTS credits	7

4. Prerequisites (if necessary)

4.1. curriculum	Fundamentals of Programming, Algorithms and Data Structures, Object-Oriented Programming, Advanced Programming Methods, Logic and Functional Programming
4.2. competencies	Basic knowledge in Python, Java, C#, C++

5. Conditions (if necessary)

5.1. for the course	Projector for lecture presentations
5.2. for the seminar /lab activities	Computers for practical assignments

6. Specific competencies acquired

Professional competencies	• Good programming skills in high-level languages • Better understanding of the program execution • Ability to design and implement DSL (Domain Specific Languages) • Better knowledge about program semantics • Better knowledge about automated program verification • Better knowledge about writing correct code • Better knowledge about code optimization
Transversal competencies	• Ability to design and build dependable software systems • Ability to design and build critical systems

7. Objectives of the discipline (outcome of the acquired competencies)

7.1 General objective of the discipline	• Understanding of the main concepts and techniques to design and implement a language interpreter (virtual machine)
7.2 Specific objective of the discipline	• To understand the execution model of a program • To understand the automated program analyse • To understand how an interpreter (virtual machine) works • To understand how to implement a DSL • To understand the automated techniques to optimized the program • To understand the automated program verification • To become familiar with the tools which automatically analise, optimize and verify the programs

8. Content

8.1 Course	Teaching methods	Remarks
1. Introduction into code interpretation. Exemple of virtual machine: Java VM, .NET CLI, SECD machine, WAM machine.	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
2. Principles of declarative programming. Basics of	• Interactive exposure	

OCaml language.	• Explanation • Conversation • Didactical demonstration	
3. Practical OCaml programming	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
4. Operational semantics. Exemples for a simple imperative language and a simple object-oriented language	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
5. Static semantics. Type systems for a simple imperative language and a simple object-oriented language.	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
6. Symbolic execution of a program. Program representations: abstract syntax tree vs control flow graph	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
7. Domain Specific Languages: design and implementation	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
8. DataFlow Analyses for code optimization	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
9. DataFlow Analyses for code verification	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
10. ControlFlow Analyses	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
11. Pointer Analyses	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
12. Code genration vs code interpretation	• Interactive exposure • Explanation • Conversation	

	• Didactical demonstration	
13. Code verification using Separation Logic	• Interactive exposure • Conversation	
14. Code verification using Separation Logic	• Interactive exposure • Conversation	

Bibliography

1. F. Nielson, H.R. Nielson, C. Hankin, Principles of Program Analysis
2. OCAML handbook. http://caml.inria.fr/pub/docs/manual-ocaml/
3. A. Appel. Modern compiler implementation in Java
4. A. Appel. Modern compiler implentation in ML
5. Benjamin Pierce. Types and Programming Languages

8.2 Seminar / laboratory	Teaching methods	Remarks
1. Principles of declarative programming. Learning OCAML language by examples	Conversation, debate, case studies, examples	The laboratory is structured as 2 hours classes every second week
2. Initiate the project: design and implementation of an interprete for an OO language in Ocaml. Design the language and generate its AST.	•	
3. Implemetation: Operational Semantic and Symbolic Execution	•	
4. Implementation: Type System		
5. Implementation: DataFlow Analyses	•	
6. Implementation: ControlFlowAnalyses	•	
7. Implementation: Modular Verification of the code	•	
	•	
	•	

Bibliography

The latest academic tools open source. The students will be able to change/adapt the tools.

9. Corroborating the content of the discipline with the expectations of the epistemic community, professional associations and representative employers within the field of the program

- The course respects the IEEE and ACM Curriculla Recommendations for Computer Science studies
- The content of the course is considered by the software companies as important for average software development skills

10. Evaluation

Course	• - know the basic principle of • the domain; • - apply the course concepts problem solving	Written Final Exam	30.00%
	•		
	•		
Seminar/lab activities	• - be able to use course • concepts in solving the real problems	Laboratory Project	70.00%
	•		
• At least grade 5 (from a scale of 1 to 10) at written final exam and at each laboratory assignment.			

Date | Signature of course coordinator | Signature of seminar coordinator

Assoc. Prof. Florin Craciun | Assoc. Prof. Florin Craciun

Date of approval | Signature of the head of department