COURSE DESCRIPTION CARD - SYLLABUS

Course name
Program Synthesis

Course

Field of study
Artificial Intelligence
Area of study (specialization)

Level of study
second-cycle studies
Form of study
full-time

Year/Semester
1/1
Profile of study
general academic
Course offered in
English
Requirements
compulsory

Number of hours

Lecture	Laboratory classes	Other (e.g. online)
30	0	
Tutorials	Projects/seminars	
	30	

Number of credit points

4

Lecturers

Responsible for the course/lecturer:
Iwo Błądek, PhD
email: iwo.bladek@cs.put.poznan.pl
tel. 61 665 3058
Institute of Computing Science, Faculty of Computing and Telecommunications
Piotrowo 2, 60-965 Poznań

Responsible for the course/lecturer:
Professor Krzysztof Krawiec
email: krzysztof.krawiec@cs.put.poznan.pl
tel. 61 665 3061
Institute of Computing Science, Faculty of Computing and Telecommunications
Piotrowo 2, 60-965 Poznań

Prerequisites

A student starting this course should have basic knowledge of logic, combinatorial optimization algorithms, and machine learning. He/she should also have the ability to obtain information from the indicated sources and cooperate as part of the team, since the course involves group projects.

Course objective

1. Familiarizing the students with the fundamentals and selected advanced topics of program synthesis, with emphasis on connections with artificial intelligence and machine learning.

2. Developing skills of solving problems pertaining to program synthesis, in particular formulating a specification of properties the synthesized program should have, selecting an appropriate program synthesis technique, and evaluating its outcomes.

3. Developing an ability of working as a team on a software project by pairing students for the project during laboratories.

Course-related learning outcomes

Knowledge

Has a structured and theoretically founded general knowledge related to key issues in the field of program synthesis [K2st_W2].

Has advanced detailed knowledge regarding selected topics in program synthesis, in particular different types of program specification, using logical reasoning to ensure program correctness and deduce the program satisfying the specification, and understanding how stochastic optimization and machine learning techniques can be used to solve or facilitate solving of the program synthesis problems [K2st_W3].

Has advanced and detailed knowledge of the processes occurring in the life cycle of program synthesis systems, including data acquisition techniques, and designing, testing and deployment of such systems [K2st_W5].

Knows advanced methods, techniques and tools used to solve complex engineering tasks and conduct research within program synthesis, in particular the methodology pertaining to conducting computational experiments and metrics for assessment of the quality of program synthesis systems [K2st_W6].

Skills

Is able to plan and carry out experiments, including computer measurements and simulations, interpret the obtained results and draw conclusions and formulate and verify hypotheses related to complex engineering problems and simple research problems in program synthesis [K2st_U3].

Can use analytical, simulation and experimental methods to formulate and solve engineering problems and simple research problems in program synthesis [K2st_U4].

Can - when formulating and solving engineering tasks - integrate knowledge from different areas of computer science (and if necessary also knowledge from other scientific disciplines) and apply a systemic approach, also taking into account non-technical aspects [K2st_U5].

Is able to assess the suitability and the possibility of using new achievements (methods and tools) and new IT products [K2st_U6].

Can carry out a critical analysis of existing technical solutions used in program synthesis systems and propose their improvements [K2st_U8].

Is able - using among others conceptually new methods - to solve complex tasks involving design and implementation of program synthesis systems, including atypical tasks and tasks containing a research component [K2st_U10].

Social competences

The student understands that in computer science knowledge and skills very quickly become obsolete [K2st_K1].

The student understands the importance of using the latest knowledge in the field of computer science in solving research and practical problems. [K2st_K2].

Methods for verifying learning outcomes and assessment criteria

Learning outcomes presented above are verified as follows:

Formative assessment:

a) lectures:

- asking students questions pertaining to the material presented in previous lectures.

b) laboratory classes:

- assessment of the progress in the implementation of assigned tasks,
- evaluation of progress in project realization (checkpointing).

Summary assessment:

a) verification of assumed learning objectives related to lectures:

- Evaluation of acquired knowledge in the form of a written exam (5-8 open questions pertaining to lecture contents, and 2-6 closed questions). Roughly half of questions are theoretical (define, describe, characterize, etc.), the other half are practical (e.g., simulating the working of an algorithm). Maximum total score: 25 points, of which 13 are required to obtain a positive grade.

b) verification of assumed learning objectives related to laboratory classes:

- Evaluation of progress along the semester classes, based on the tasks carried out by students (in total 35 points to score).
- Evaluation of student's "defense" of a project report and project presentation taking place at the last laboratory class, with the other students in the audience (in total 35 points to score). The grade will be influenced by such factors as the conducted computational experiments, soundness of the approach used, and quality of the created code, report, and final presentation.
- To get a passing grade, a student needs to get at least 50% of points from each part mentioned above.

Programme content

Lectures:

Introduction. The definition of program synthesis problem and its similarities and differences with related tasks (e.g.compilation and machine learning). Dimensions of program synthesis. Different types of specification expressing user intent, and their relative strengths and weaknesses. Domain specific languages. Functional programming and its advantages. Usage scenarios and examples of successful practical applications of program synthesis. End user programming.

Deductive synthesis techniques. Introduction to program verification. Curry-Howard correspondence. Model checking. Loop invariants. Hoare logic. Satisfiability modulo theories (SMT). Solving program synthesis problems by transformation to an SMT problem. Counterexample guided inductive synthesis. Programming by sketching.

Enumerative synthesis techniques. Pruning search space and prioritizing search. Equivalence reduction. Top-down specification propagation. Weighted enumerative search. EUSolver synthesis algorithm.

Inductive synthesis techniques. Programming by example and programming by demonstration. MagicHaskeller as an example of inductive synthesis.

Heuristic search algorithms. Motivations, advantages and disadvantages of heuristic algorithms for program synthesis. Genetic Programming (GP). Semantic GP. Grammatical Evolution. Counterexample Driven GP. Stack-based GP.

Machine learning techniques. Neural networks for synthesizing programs or prioritizing search. DeepCoder. Neural-Guided Deductive Search. DreamCoder. Natural Language Processing with Recurrent Neural Networks. Neurosymbolic systems.

Laboratories:

The lab classes (15 x 2 hours) take place in computer laboratories and are divided into two types:
1. Exercises related to the material covered during lectures (7 x 2 hours) – tasks of various nature to be done during class: programming tasks, theoretical problems, exercises. Each class will start with a short discussion of the material presented during lectures.
2. Realization of the projects (8 x 2 hours) – students will form project pairs and they will select a topic from the list or propose their own after consultation with the lecturer. During the class they will be able to work on the project and consult their progress or eventual issues with the lecturer. In the final class of this type the students will present their work and results of computational experiments before the whole group.

Teaching methods

Lectures: multimedia presentation, software demonstration.

Laboratories: practical exercises, problem solving, discussion, teamwork, consultations, presentation of project outcomes (software and computational experiments).

Bibliography

Basic

1. Krzysztof Krawiec, **Behavioral Program Synthesis with Genetic Programming**, Springer, 2016.

2. Sumit Gulwani, Oleksandr Polozov, Rishabh Singh, **Program Synthesis**, Foundations and Trends in Programming Languages, 4(1-2), 2017, 1–119.

Additional

1. Sumit Gulwani. **Dimensions in Program Synthesis**, Proceedings of the 12th international ACM SIGPLAN symposium on Principles and practice of declarative programming, 2010, 13–24.

2. Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit Seshia, Rajdeep Singh, Armando Solar-Lezama, Emina Torlak, Abhishek Udupa. **Syntax-Guided Synthesis**, Formal Methods in Computer-Aided Design (FMCAD), IEEE, 2013, 1–8.

3. Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, Daniel Tarlow. **DeepCoder: Learning to Write Programs**, Proceedings of the International Conference on Learning Representations, 2017.

4. Armando Solar-Lezama, Liviu Tancau, Rastislav Bodík, Sanjit A. Seshia, Vijay A. Saraswat. **Combinatorial Sketching for Finite Programs**, Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, 2006, 404–415.

5. Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, Ashish Tiwari. **Oracle-Guided Component-Based Program Synthesis**, In 2010 ACM/IEEE 32nd International Conference on Software Engineering, volume 1, 2010, 215–224.

6. Swarat Chaudhuri, Kevin Ellis, Oleksandr Polozov, Rishabh Singh, Armando Solar-Lezama, and Yisong Yue. **Neurosymbolic Programming.** Foundations and Trends in Programming Languages, 7, 3 (Dec 2021), 158–243. https://doi.org/10.1561/2500000049.

Breakdown of average student's workload

	Hours	ECTS
Total workload	100	4
Classes requiring direct contact with the teacher	60	2
Student's own work (literature studies, preparation for laboratory classes/tutorials, preparation for tests/exam, project preparation) [1]	40	2

[1] delete or add other activities as appropriate