POZNAN UNIVERSITY OF TECHNOLOGY

EUROPEAN CREDIT TRANSFER AND ACCUMULATION SYSTEM (ECTS)

COURSE DESCRIPTION CARD - SYLLABUS

Course name
Program Synthesis [S2SI1E>SPR]

Course

Field of study
Artificial Intelligence

Year/Semester
1/1

Area of study (specialization)
–

Profile of study
general academic

Level of study
second-cycle

Course offered in
english

Form of study
full-time

Requirements
compulsory

Number of hours

Lecture
30

Laboratory classes
0

Other (e.g. online)
0

Tutorials
0

Projects/seminars
30

Number of credit points

4,00

Coordinators

dr inż. Iwo Błądek
iwo.bladek@put.poznan.pl

prof. dr hab. inż. Krzysztof Krawiec
krzysztof.krawiec@put.poznan.pl

Lecturers

dr inż. Iwo Błądek
iwo.bladek@put.poznan.pl

prof. dr hab. inż. Krzysztof Krawiec
krzysztof.krawiec@put.poznan.pl

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:
- evaluation of progress in project realization.

Summary assessment:
a) verification of learning objectives related to lectures:
- Evaluation of acquired knowledge in the form of a written test, containing both open-ended and closed-ended questions. Questions are either theoretical (define, describe, characterize, etc.), or practical (e.g., simulating the working of an algorithm). Each question is worth a certain amount of points, and to get a positive grade a student needs to get at least 50% of points possible to achieve for the test.
b) verification of learning objectives related to laboratory classes:
- The evaluation, on a standard grading scale, of the student's project implementation, including the project report and presentation held during the final laboratory classes with the participation of other students in the audience. The assessment will be influenced by factors such as the validity of the conducted computational experiments, the applied approach, and the quality of the created code, report, and final presentation.

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.

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.

Project:
In the project classes, students are divided into teams of 2 or 3 (in the case of larger projects) and choose a project topic from the list or propose their own after consultation with the instructor. Then, during the semester, they work on the chosen project and periodically show their progress. In the last classes, students present the completed project and the results of computational experiments to the entire group.

Teaching methods

Lectures: multimedia presentation, software demonstration.
Laboratories: 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.
3. Wojciech Jamroga, "Logical Methods for Specification and Verification of Multi-Agent Systems", 2015, Monograph Series no 10, ICS PAS Publishing House.

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	4,00
Classes requiring direct contact with the teacher	60	2,00
Student's own work (literature studies, preparation for laboratory classes/ tutorials, preparation for tests/exam, project preparation)	40	2,00