KARTA OPISU PRZEDMIOTU - SYLABUS

Nazwa przedmiotu
Synteza Programów

Przedmiot

Kierunek studiów
Sztuczna Inteligencja
Studia w zakresie (specjalność)

Poziom studiów
drugiego stopnia
Forma studiów
stacjonarne

Rok/semestr
1/1
Profil studiów
ogólnoakademicki
Język oferowanego przedmiotu
angielski
Wymagalność
obligatoryjny

Liczba godzin

Wykład
30
Ćwiczenia

Laboratoria
0
Projekty/seminaria
30

Inne (np. online)

Liczba punktów ECTS

4

Wykładowcy

Odpowiedzialny za przedmiot/wykładowca:
dr inż. Iwo Błądek
email: iwo.bladek@cs.put.poznan.pl
tel. 61 665 3058
Instytut Informatyki, Wydział Informatyki i Telekomunikacji
ul. Piotrowo 2, 60-965 Poznań

Odpowiedzialny za przedmiot/wykładowca:
prof. dr hab. inż. Krzysztof Krawiec
email: krzysztof.krawiec@cs.put.poznan.pl
tel. 61 665 3061
Instytut Informatyki, Wydział Informatyki i Telekomunikacji
ul. Piotrowo 2, 60-965 Poznań

Wymagania wstępne

Student rozpoczynający ten kurs powinien posiadać podstawową wiedzę z zakresu logiki, algorytmów optymalizacji kombinatorycznej oraz uczenia maszynowego. Powinien również posiadać umiejętność pozyskiwania informacji ze wskazanych źródeł oraz współpracy w zespole, gdyż kurs zakłada realizację projektów grupowych.

Cel przedmiotu

1. Zapoznanie studentów z podstawami i wybranymi zaawansowanymi zagadnieniami syntezy programów, z naciskiem na powiązania ze sztuczną inteligencją i uczeniem maszynowym.

2. Rozwijanie umiejętności rozwiązywania problemów związanych z syntezą programów, w szczególności formułowania specyfikacji własności, jakie powinien mieć syntetyzowany program, wyboru odpowiedniej techniki syntezy programów oraz oceny jej wyników.

3. Rozwijanie umiejętności pracy zespołowej nad projektem programistycznym poprzez łączenie studentów w pary do realizacji projektu podczas laboratoriów.

Przedmiotowe efekty uczenia się

Wiedza

Ma uporządkowaną i podbudowaną teoretycznie wiedzę ogólną dotyczącą kluczowych zagadnień z zakresu syntezy programów [K2st_W2].

Ma zaawansowaną wiedzę szczegółową dotyczącą wybranych zagadnień z zakresu syntezy programów, w szczególności różnych typów specyfikacji programów, stosowania logicznego rozumowania w celu zapewnienia poprawności programu i wnioskowania o programie spełniającym specyfikację, oraz rozumienia, w jaki sposób techniki optymalizacji stochastycznej i uczenia maszynowego mogą być wykorzystane do rozwiązywania lub ułatwiania rozwiązywania problemów syntezy programów [K2st_W3].

Ma zaawansowaną i szczegółową wiedzę na temat procesów zachodzących w cyklu życia systemów syntezy programów, w tym technik pozyskiwania danych oraz projektowania, testowania i wdrażania takich systemów [K2st_W5].

Zna zaawansowane metody, techniki i narzędzia stosowane do rozwiązywania złożonych zadań inżynierskich i prowadzenia badań w zakresie syntezy programów, w szczególności metodologię dotyczącą prowadzenia eksperymentów obliczeniowych oraz metryki oceny jakości systemów syntezy programów [K2st_W6].

Umiejętności

Potrafi planować i przeprowadzać eksperymenty, w tym pomiary i symulacje komputerowe, interpretować uzyskane wyniki i wyciągać wnioski oraz formułować i weryfikować hipotezy dotyczące złożonych problemów inżynierskich oraz prostych problemów badawczych w zakresie syntezy programów [K2st_U3].

Potrafi wykorzystać metody analityczne, symulacyjne i eksperymentalne do formułowania i rozwiązywania problemów inżynierskich oraz prostych problemów badawczych w zakresie syntezy programów [K2st_U4].

Potrafi - przy formułowaniu i rozwiązywaniu zadań inżynierskich - integrować wiedzę z różnych obszarów informatyki (a w razie potrzeby także wiedzę z innych dyscyplin naukowych) oraz stosować podejście systemowe, uwzględniając także aspekty pozatechniczne [K2st_U5].

Potrafi ocenić przydatność i możliwość wykorzystania nowych osiągnięć (metod i narzędzi) oraz nowych produktów informatycznych [K2st_U6].

Potrafi przeprowadzić krytyczną analizę istniejących rozwiązań technicznych stosowanych w systemach syntezy programów i zaproponować ich usprawnienia [K2st_U8].

Potrafi - wykorzystując m.in. nowe koncepcyjnie metody - rozwiązywać złożone zadania obejmujące projektowanie i realizację systemów syntezy programów, w tym zadania nietypowe oraz zawierające komponent badawczy [K2st_U10].

Kompetencje społeczne

Student rozumie, że w informatyce wiedza i umiejętności bardzo szybko stają się przestarzałe [K2st_K1].

Student rozumie znaczenie wykorzystywania najnowszej wiedzy z zakresu informatyki w rozwiązywaniu problemów badawczych i praktycznych [K2st_K2].

Metody weryfikacji efektów uczenia się i kryteria oceny

Efekty uczenia się przedstawione wyżej weryfikowane są w następujący sposób:

Ocena formatywna:

a) wykłady:

- zadawanie studentom pytań odnoszących się do materiału prezentowanego na poprzednich wykładach.

b) zajęcia laboratoryjne:

- ocena postępów w realizacji przydzielonych zadań,
- ocena postępów w realizacji projektu (checkpointing).

Ocena podsumowująca:

a) weryfikacja założonych celów dydaktycznych związanych z wykładami:

- Ocena zdobytej wiedzy w formie egzaminu pisemnego (5-8 pytań otwartych odnoszących się do treści wykładów oraz 2-6 pytań zamkniętych). Mniej więcej połowa pytań ma charakter teoretyczny (definiować, opisywać, charakteryzować itp.), druga połowa ma charakter praktyczny (np. symulacja działania algorytmu). Maksymalna łączna liczba punktów: 25, z czego 13 jest wymagane do uzyskania oceny pozytywnej.

b) weryfikacja założonych celów dydaktycznych związanych z zajęciami laboratoryjnymi:

- Ocena postępów w trakcie zajęć semestralnych, na podstawie wykonanych przez studentów zadań (w sumie 35 punktów do zdobycia).
- Ocena "obrony" przez studenta sprawozdania z projektu i prezentacji projektu odbywającej się na ostatnich zajęciach laboratoryjnych, z udziałem pozostałych studentów na widowni (w sumie 35 punktów do zdobycia). Na ocenę będą miały wpływ takie czynniki jak przeprowadzone eksperymenty obliczeniowe, zasadność zastosowanego podejścia oraz jakość stworzonego kodu, raportu i prezentacji końcowej.
- Aby uzyskać ocenę pozytywną, student musi uzyskać co najmniej 50% punktów z każdej z wymienionych części.

Treści programowe

Wykłady:

Wprowadzenie. Definicja problemu syntezy programów oraz jego podobieństwa i różnice z zadaniami pokrewnymi (np. kompilacja i uczenie maszynowe). Wymiary syntezy programów. Różne rodzaje specyfikacji wyrażających intencje użytkownika oraz ich względne mocne i słabe strony. Języki dziedzinowe (DSL). Programowanie funkcyjne i jego zalety. Scenariusze użycia i przykłady udanych praktycznych zastosowań syntezy programów. Programowanie dla użytkowników końcowych.

Techniki syntezy dedukcyjnej. Wprowadzenie do weryfikacji programów. Korespondencja Curry-Howard'a. Weryfikacja modelowa. Niezmienniki pętli. Logika Hoare'a. Spełnialność modulo teorie (SMT). Rozwiązywanie problemów syntezy programów przez transformację do problemu SMT. Synteza indukcyjna kierowana kontrprzykładami. Programowanie przez szkicowanie.

Techniki syntezy enumeratywnej. Przycinanie przestrzeni przeszukiwania i priorytetyzacja przeszukiwania. Redukcja równoważności. Propagacja specyfikacji z góry na dół. Ważone wyszukiwanie enumeratywne. Algorytm syntezy EUSolver.

Techniki syntezy indukcyjnej. Programowanie przez przykłady i programowanie przez demonstrację. MagicHaskeller jako przykład syntezy indukcyjnej.

Heurystyczne algorytmy wyszukiwania. Motywacje, zalety i wady algorytmów heurystycznych do syntezy programów. Programowanie genetyczne (GP). Semantyczne GP. Grammatical Evolution. Counterexample Driven GP. GP oparte na stosie.

Techniki uczenia maszynowego. Sieci neuronowe do syntezy programów lub priorytetyzacji przeszukiwania. DeepCoder. Neural-Guided Deductive Search. DreamCoder. Przetwarzanie języka naturalnego za pomocą rekurencyjnych sieci neuronowych. Systemy neurosymboliczne.

Laboratoria:

Zajęcia laboratoryjne (15 x 2 godziny) odbywają się w laboratoriach komputerowych i dzielą się na dwa rodzaje:

1. Ćwiczenia związane z materiałem omawianym na wykładach (7 x 2 godziny) - zadania o różnym charakterze do wykonania na zajęciach: zadania programistyczne, problemy teoretyczne, ćwiczenia. Każde zajęcia rozpoczynać się będą od krótkiego omówienia materiału przedstawionego na wykładach.

2. Realizacja projektów (8 x 2 godziny) - studenci utworzą pary projektowe i będą wybierać temat z listy lub proponować własny po konsultacji z prowadzącym. Podczas zajęć będą mogli pracować nad projektem i konsultować swoje postępy lub ewentualne problemy z prowadzącym. Na ostatnich zajęciach tego typu studenci zaprezentują swoją pracę i wyniki eksperymentów obliczeniowych przed całą grupą.

Metody dydaktyczne

Wykłady: prezentacja multimedialna, demonstracja oprogramowania.

Laboratoria: ćwiczenia praktyczne, rozwiązywanie problemów, dyskusja, praca zespołowa, konsultacje, prezentacja wyników projektu (oprogramowanie i eksperymenty obliczeniowe).

Literatura

Podstawowa

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.

Uzupełniająca

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.

Bilans nakładu pracy przeciętnego studenta

	Godzin	ECTS
Łączny nakład pracy	100	4
Zajęcia wymagające bezpośredniego kontaktu z nauczycielem	60	2
Praca własna studenta (studia literaturowe, przygotowanie do zajęć laboratoryjnych/ćwiczeń, przygotowanie do kolokwiów/egzaminu, wykonanie projektu)[1]	40	2

[1] niepotrzebne skreślić lub dopisać inne czynności