Approved in 37th BoA Meeting (29-10-2020)

Course number	:	CS611
Course name	:	Program Analysis
Credits	:	3-1-0-4
Prerequisites	:	CS202: Data Structures and Algorithms or equivalent; CS208: Mathematical Foundations of Computer Science or equivalent; CS304: Formal Languages and Automata Theory or equivalent
Intended for	:	B.Tech., M.Tech., MS, PhD
Distribution	:	Discipline Elective for BTech CSE, Elective for others

---

1. Preamble

Program analysis approximates the runtime behavior of a program by looking at its source code. The results of program analyses are used to drive a plethora of applications: performance-oriented optimizations, program understanding, verification, debugging, refactoring, finding vulnerabilities, test generation, parallelization, and so on. As a result, once an exclusive part of the back-end of optimizing compilers, now program analysis finds its applications in the development of a large number of tools in a programming language's ecosystem. The aim of this course is threefold: (i) to develop a thorough understanding of the theory behind the long studied discipline of analyzing programs; (ii) to explore modern strategies that balance the trade-offs between the precision of an analysis and the resources consumed therein; and (iii) to get introduced to various applications that benefit from the results of a precise-yet-efficient program analysis.

2. Learning outcomes

After taking this course, the students will:

- have a strong foundation in abstract interpretation, which is a way to soundly approximate the semantics of a computer program;
- have a thorough understanding of the various dimensions along which the precision of an analysis can be varied and the associated trade-offs;
- be able to make intelligent decisions concerning the abstraction used to represent program features and the data structures used to implement static analyses, with an understanding of the corresponding effects on analysis precision and efficiency;
- be able to appreciate the challenges imposed by various features of programming languages on designing the associated compiler technology;
- have hands-on experience with some of the modern tools and frameworks used to implement program analyses in production and research environments; and
- be excited to use program analyses to improve different aspects of a program and the overall programming language ecosystem.

3. Course modules

- *Introduction to static analysis.* Concrete versus abstract semantics. Abstract interpretation. Galois connection. Symbolic execution. Control-flow graphs. Iterative dataflow analysis. Lattices and monotonicity. Analysis dimensions. **[10 hours]**

- *Heap analysis.* Heap modeling. Points-to information. Andersen's and Steensgard's pointer analyses. Variations: alias analysis, null-check analysis, escape analysis. **[9 hours]**

- *Interprocedural analysis.* Call-graph construction. Context sensitivity: functional and call-string approaches. Various context abstractions: value and lsrv contexts, object and type sensitivity. Heap cloning. **[10 hours]**

- *Strategies for efficiency.* Demand-driven analysis. Program slicing. Analysis staging. Partial analysis. Efficient data structures. Heuristics and machine learning. **[9 hours]**

- *Language features and challenges.* Lexical and dynamic scoping. Eager and lazy evaluation. Call-backs and reflection. Concurrency and synchronization: may-happen-in-parallel analysis. Speculative optimizations and deoptimization. Dynamic analysis. **[10 hours]**

- *Sneak peek into applications.* Type checking. Bug detection. Program correctness. Program synthesis and repair. Software refactoring. **[8 hours]**

4. Tutorials and assignments

This course involves hands-on practice with writing different program analyses, implementing the techniques for efficiency, and learning various associated tools and language features. The classes will teach the theory, and tutorials would train the students on the various skill-sets required to finish take-home programming assignments.

It may be noted that as the course covers several recent topics related to designing analyses that are precise-yet-efficient, the classes will use one or two important analyses as running examples (examples being alias analysis and pointer analysis for resolving virtual calls). Hence, another aim of the tutorial hours would be to help students imbibe the concepts learnt by making them write specifications of different analyses and optimization strategies.

The take-home assignments will be based on implementing and understanding (a) intra- and inter-procedural analyses; (b) strategies for imparting efficiency; (c) examples of handling tricky language features; and (d) non-trivial applications such as parallelization, refactoring and security.

5. References

1. Flemming Nielson, Hanne Riis Nielson and Chris Hankin. "Principles of Program Analysis", Corrected Edition, Springer, 1999.
2. Uday P. Khedker, Amitabha Sanyal and Bageshri Karkare. "Data Flow Analysis: Theory and Practice", First Edition, CRC Press, 2009.
3. Y. N. Srikant and Priti Shankar. "The Compiler Design Handbook", Second Edition, CRC Press, 2007.
4. Various research papers related to the course content.

6. **Similarity content declaration with existing courses**: <10% (CS502 Compiler Design: Control-flow graphs, iterative dataflow analysis, type checking.)

7. **Justification of new course proposal if cumulative similarity content is >30%**: N/A