A Case for the Relevance of COT 4210

Charles Newton

3 August 2011

The mission statement\(^1\) for the CS program at UCF is as follows:

“The mission of the Computer Science program is to educate majors in the principles and practices of computer science, preparing them for graduate school, for careers in software development and computing systems technology, and a lifetime of learning.”

This paper will argue that COT 4210, or a course that covers equivalent topics, is necessary to fulfill both the mission statement of CS and also the learning objectives set by EECS for the degree program. Specifically, we will look at the following learning objectives\(^2\) established by the Accreditation Board for Engineering and Technology (ABET), the accreditor for UCF’s EECS department:

“The program enables students to achieve, by the time of graduation:

1. An ability to apply knowledge of computing and mathematics appropriate to the discipline.

2. An ability to analyze a problem, and identify and define the computing requirements appropriate to its solution.

3. An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs.”

\(^1\)See the CS tab on \url{http://abet.eecs.ucf.edu/?id=mission}

\(^2\)ABET, “Criteria for Accrediting Computing Programs.”

See pg 14 on \url{http://tinyurl.com/39z9em}
COT 4210 is a particularly unique course at UCF. In contrast to other required upper-level CS classes which typically have students complete a term project that applies the course material (e.g., COP 4331 (OOP Design) and COP 4600 (Operating Systems)), COT 4210 is theory-based. Students are required to apply what they have learned about formal languages and automata/complexity theory by writing rigorous proofs. In fact, excluding COT 4210, I have never been asked to formally prove a statement for any (required) advanced-level CS class. It is difficult to see how a graduating CS major at UCF, sans COT 4210, would have necessarily demonstrated their ability to formally reason about computing systems. COT 4210 is the work-horse, in this regard, for teaching students how to apply mathematics to computer science problems.

Yet, the ability to rigorously reason about algorithms is incredibly important - not only in other classes at UCF, but also in practical workplace situations. For example, in an operating systems class (COP 4600), a student may learn about deadline scheduling. In general, this is difficult to prove since deadline scheduling is NP-Complete. A naïve deadline scheduler can miss task deadlines by taking up an inordinate amount of CPU time to determine which task should be scheduled next. One of the main benefits of deadline schedulers used in the real world is that their conservative nature makes them easy to reason about. Since I took COP 4600 before COT 4210, I completely missed the point about how practical real-time schedulers sidestep NP-Completeness and how this is the gateway for reasoning about such algorithms.

In an analogous industrial situation, the correctness of a deadline scheduler should be proved before letting it control tasks regrading, say airline jets. (The FAA uses a real-time scheduler\textsuperscript{3} in air-traffic control systems.) A CS student without experience with formal logical proofs would either be precluded from working on a very interesting project or in danger of being responsible for a possibly tragic failure in air-traffic control.

Computer science graduates are often asked to field questions regarding regular expressions, parsing, grammars, and other matters about formal languages and automata. A special amount of care must be taken when answering such questions since incorrect advice can lead to solutions that are

ridiculously complex and impossible to maintain. For example, modern regular expression libraries are Turing complete, and thus many find it tempting to recklessly apply them to any parsing or validation situation. Although RFC 822\(^4\) specifies the language of valid e-mail addresses with a context-free grammar, many programmers employ regular expressions for this task. Paul Warren has constructed\(^5\) a tongue-in-cheek 6,343 character Perl regular expression that correctly recognizes e-mail addresses. Also, software developers perennially use regular expressions to parse XHTML / HTML, regardless of the fact that XML is a textbook example of a context-free language. Alumni of COT 4210 are well-equipped to argue against such egregious abuses of automata.

COT 4210 also dovetails students' experiences in other classes. Garey and Johnson’s classic book\(^6\) on NP-Completeness begins with an amusing fictional anecdote about a chief algorithm designer’s task to design an efficient algorithm for constructing “bandersnatch” components from given specifications. Sadly, this problem is found to be NP-Complete. The algorithm designer is able to inform his boss that although he doesn’t know if an efficient algorithm is possible or not, he is at least able to establish an upper-bound on his own incompetence since a host of famous researchers are also stumped on the same problem. The point of the anecdote is, of course, not about job security, but rather, programmers who are aware of NP-Completeness (and who operate under the assumption that \(P \neq NP\)) can direct their time towards developing a reasonable approximation algorithm or finding ways to restrict the problem space.

A similar situation once played out when I took COP 4331 (Object Oriented Processes for Software Development.) The class was structured where students completed software engineering projects in small teams. One of the project choices was particularly daunting: create an online swap meet-style bartering website where people can trade used goods.\(^7\) The customer insisted the developers implement a way for circular exchanges to occur with an unbounded number of parties. This problem is a generalization of the

\(^4\)http://www.ietf.org/rfc/rfc0822.txt
\(^5\)http://ex-parrot.com/~pdw/Mail-RFC822-Address.html
kidney-exchange problem, which is known to be NP-Complete. In the first round of presentations, this group mentioned how they had dedicated a large amount of their planning phase towards algorithm design, but they were unable to find an efficient algorithm and instead opted for a back-tracking solution as a fallback. A great deal of time could have been saved by noticing the NP-Completeness of this problem. If UCF is going to assign students NP-Complete problems as term projects in required classes, then clearly some training in complexity theory should also be required.

Additionally, Dr. Leavens’ offering of COP 4020 (Programming Languages) typically goes over relational programming, which gives general semantics for common patterns in solving problems by exhaustive search. This is an excellent framework for dealing with NP-Complete problems. Unfortunately, NP-Completeness was not mentioned in the course - meaning a student needs to take both COP 4020 and COT 4210 to have a full grasp of what sorts of problems relational programming (assuming \( P \neq NP \)) best expresses.

A goal of undergraduate CS education should be to prepare students for graduate school (and this is reflected in UCF’s CS mission statement.) UCF’s graduate CS program requires students to have a strong background in the following areas: computer architecture, programming paradigms, operating system design, formal languages, and automata/complexity theory. Were COT 4210 not required, a student completing UCF’s undergraduate program could, later, awkwardly discover they lack the prerequisites for graduate study at their own undergraduate institution. Worse, the student would be unqualified for graduate study in CS at most accredited schools.

It is encouraging that UCF’s mission statement for computer science includes a goal of promoting a “lifetime of learning.” In CS, the life span of specific technical skills is quite short - perhaps a few decades at best. (For an extreme example, Dr. Workman has some design pattern illustrations from past courses in object oriented design written in ADA 95!) However, there are certain general skills that will always be useful, such as the ability to write clean code, design efficient algorithms, and one’s ability to make mathemat-

\[\text{8A reduction of 3-dimensional matching to kidney-exchange is given in C. Huang, “Circular Stable Matching and 3-way Kidney Transplant,” Algorithmica 58(1), pgs 137-150.}\]
\[\text{9See course notes on Dr. Leavens’ COP 4020 homepage: http://tinyurl.com/3nqg3ka}\]
\[\text{11http://tinyurl.com/43b9czc}\]
ical arguments. These skills not only ensure a student lifetime success in a work environment, but also give a student the flexibility to learn future developments in computer science. A university has a dual responsibility in vocational training: its students should be technically competent but also literate in their field. Knowledge about formal languages and automata/complexity theory is an integral component of computer science literacy.

It is likely that dropping COT 4210 from UCF’s list of required classes for CS majors would jeopardize both the relevance of a CS degree from UCF and the accreditation of the EECS department. This would be an unfortunate outcome for UCF’s students, faculty, and alumni.