Problem-Directed Discrete Structures Course

Norman Neff
Computer Science Department
The College of New Jersey
Hillwood Lakes, N.J.
609 771 2268
neff@tcnj.edu

ABSTRACT
The "2007 SIGCSE Report On the Implementation of a Discrete Mathematics Course" recommends relocating some discrete structures topic areas to other courses, for the sake of coherency. Our implementation of the one semester course also relocates topics, but additionally features a focus on a significant computer science problem: regular language acceptors. This problem-directed course model promises superior coherency, motivation, and integration with computing.

Categories and Subject Descriptors

General Terms
Algorithms, Languages, Theory

Keywords
CC2001, computing curriculum, course models, discrete mathematics, discrete structures

1. INTRODUCTION
Shortly after the dissemination of the Joint ACM/IEEE Task Force Report on Computing Curricula 2001 [1](CC2001), SIGCSE recognized the need for detailed implementation recommendations for a coherent one semester discrete structures course[2]. This effort culminated in the 2007 SIGCSE Report On the Implementation of a Discrete Mathematics Course[7]. Taking into account the core topic recommendations from CC2001, as well as responses from its survey of undergraduate Computer Science Departments, the 2007 Report found that it was “not possible to develop a coherent package and cover all of the core topics”. Adopting a strategy of relocating topics to other courses in order to improve coherency, the 2007 Report recommended three models for a one semester Discrete Structures course. The three models exhibited varying allocations of emphasis among the six topics. Each of the models relocated one or two entire topics from discrete mathematics to other courses in the curriculum. This paper describes a new model for Discrete Structures, in which content allocation is relaxed in order to further improve focus and coherency. The theme of the model is a deep understanding of a single computer science problem. In subsequent sections we detail such an approach to Discrete Structures via the problem of regular language acceptance.

2. CURRICULAR FRAMEWORK
In designing the course implementation, we sought a course that
* utilizes one semester
* is accessible to underclassmen in computer science and computer engineering
* in combination with the rest of the curriculum, provides coverage of the required CC2001 discrete mathematics topics
* integrates mathematics with computer science applications
* possesses a coherent computer science theme: mathematics related to regular languages

To accomplish all of the above, we begin with students who have some experience in computing, and build on that experience. The prerequisite for the course is sophomore status, which in our curriculum implies some experience with Java, data structures, and some college level math.

A second curricular issue is the existence of a requirement that Computer Science majors complete one semester of probability and statistics. This frees the Discrete Structures course of responsibility for two of the six CC2001 topic areas. One might argue that the combined probability and Discrete Structures courses effectively constitute a two semester Discrete Structures sequence, but we believe that our organization is much more desirable, for the following reasons:

i) The probability course covers discrete and continuous probability, seen as two models of the general axioms for probability measures. Such a grouping makes for a more complete treatment of modern probability theory than would be likely within a two semester Discrete Structures course.

ii) The one semester Discrete Structures course and the probability course may be taken in any order or concurrently.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCSE ’10, March 10–13, 2010, Milwaukee, Wisconsin, USA.
Copyright 2010 ACM 978-1-60558-885-8/10/03 ...$10.00.
iii) By relocating the probability related topics, we enable the Discrete Structures course to deeply cover regular languages. This contributes credit hours to the rest of the Computer Science curriculum by permitting rapid coverage of the regular language material in later courses, such as Theory of Computing and Compilers. Therefore, the net cost of covering the six CC2001 topic areas is less than two semesters.

3. COURSE ORGANIZATION

The theme of the course is the study of regular language acceptors. We apply the two technologies of finite automata and logic programming to this problem. Section 5 provides evidence that the mathematics underlying the two technologies subsumes the necessary discrete mathematics requirements of the CC2001 report.

Given a finite alphabet A, the set of regular languages over A is the closure, with respect to union, language concatenation, and Kleene star, of the set \{ \phi \} \cup \{ \{ s \} : s \in A \}. Regular expressions are well-formed formulas representing individual regular languages [6, 10].

The course consists of three sections.

The first section is a motivational unit called “Introduction to egrep” (2 weeks). It engages students in discovery activities using the egrep [3] regular expression pattern matching utility. Lab activities include mining html files for particular types of tags or answering questions like “How many entries in the online dictionary file are words containing exactly one ‘a’ and exactly one ‘n’?”.

The second section, “Inside egrep” (7 weeks), explores the design of an “egrep” implementation together with the supporting theory. The theory includes constructive proofs that build finite automata to accept regular languages. Figure 1 exhibits the top level of the algorithm for converting any NFA to a DFA with the same acceptance set.

The third section of the course, “Regular Languages through Logic” (4 weeks), covers resolution proof and shows that regular language questions can be reduced to searches for proofs. Figure 2 exhibits a logic program for predicate mbr/2, which checks whether a particular string (in list form) is a member of a given regular language. The program is closely based on predicate logic definitions of the regular language operations. The logic program serves as a language acceptor, and can also generate all strings from a given language if executed under a breadth-first implementation of Prolog. Under SLD resolution, the exhibited program is a decision procedure, but [8] gives a similar logic program that is only a semidecision procedure.

4. PEDAGOGICAL TACTICS

Where possible, the course provides hands-on, concrete experience to improve the accessibility of abstract mathematical ideas. In addition to the egrep work mentioned above, the course includes lab experiences with JFLAP[9], with an “automaton node browser”, and with a Prolog interpreter.

Sequencing of the discrete structures mathematics topics is primarily determined by their relation to the development of the regular language application. For example, detailed coverage of

---

Set qq, qqt, qq0 // DFA states
Set DFAstateSet = new Set( ) // set of DFA states
  // each element is a set of ints
Queue queue = new Queue( )

qq0 = lambda({NFA.startState})
queue.add(qq0)
DFAstateSet.add(qq0)
repeat
  qq = queue.remove()
  for each alphabet symbol s
  qqt = TD(qq, s) // apply DFA transition fn
  DFAedgeSet.addEdge( qq, s, qqt)
  if qqt \notin DFAstateSet
    queue.add(qqt)
    DFAstateSet.add(qqt)
until (queue.isEmpty( ))

Figure 1. Pseudocode for NFA to DFA conversion

mbr([Ch],char(Ch)). mbr([ ],emptyStrLang).
mbr(Str,union(S1,S2)):-mbr(Str,S1).
mbr(Str,union(S1,S2)):-mbr(Str,S2).
mbr(Str,concat(S1,S2)):-mbr(First,S1),mbr(Second,S2),
  append(First,Second,Str).
mbr([ ],star(S)).      (* Case 1 : arg 1 empty *)
mbr([W | Ws],star(S)):-   (* Case 2: arg1 nonempty *)
mbr([ONCE | ONCES],S),
  append([ONCE | ONCES],Rest,[W | Ws]),
mbr(Rest,star(S)).
append([ ],S,S).
append([S | R],T,[S | U]):-append(R,T,U).

Figure 2. Logic program for regular language acceptor

---
Java-based notations. Figure 3 exhibits the mathematical definition of set image together with a corresponding definition in terms of the Java container class.

\[
f(S) = \{ f(s) : s \in S \}
\]

Set imageOf(Set S){
    Set answer = new Set( );
    Iterator iS = S.iterator( );
    while (iS.hasNext())
        answer.add(f(iS.next( )));
    return answer; }

Figure 3. Mathematical and Java-based notations

5. COVERAGE OF DS CORE TOPICS
The CC2001 report identifies six topics in discrete mathematics as the knowledge base for discrete structures: (DS1) functions, relations and sets, (DS2) basic logic, (DS3) proof techniques, (DS4) basics of counting, (DS5) graphs and trees, and (DS6) discrete probability.

Our problem-directed course omits DS6 and most of DS4. The next section discusses our curriculum’s handling of these omissions in the context of the 2007 Report recommendations.

In this section, our intent is to show that the regular language problem affords ample opportunity to study and use all of foundational discrete mathematics (DS) topics, excepting the topics we have assigned to other courses.

Each regular expression implicitly presents the question: “Exactly what set of strings does the expression match?” Within the first, exploratory, part of the course, students are asked to create a grep pattern that generates a particular set of strings. Discussion of proposed solutions leads to coverage of informal proof techniques such as breaking arguments into cases, establishing bidirectional implications, and searching for counterexamples.

For the remaining parts of the course, we can give a sense of the correspondence between the regular language related topics and foundational DS topics. To conserve space, we will give detail only for the second section, “Inside egrep”. Figure 4 lists the principal definitions and proofs in the second section of the course. In the following lines, we comment on the connections between items in Figure 4 and DS topics.

i) Definitions of string, language use: sequence, set, functions

ii) Regular language definition uses: set operations, closure, inductive definition.

iii) Regular expressions are defined by a uniquely-readable inductive definition. A good introduction to proof by induction arises from proving that each regular expression contains equal numbers of left and right parentheses. A recursive function over the set of regular expressions defines the regular language denoted by each regular expression.

iv) A finite automaton is formally a tuple consisting of, a set of states, a start state, a set of final states, and a transition function.

v) regular expression \( \rightarrow \) NFA. One can prove the correctness of the NFA construction by inductive proof over regular expressions.

vi) NFA \( \rightarrow \) DFA. Each DFA state is a set of NFA states. To generate new DFA states from an existing DFA state, we first take the image under an appropriate function, and then perform closure. Transitive reflexive closure with respect to the NFA’s \( \varepsilon \)-relation is computed by fixed point iteration or breadthfirst search. The algorithm of Figure 1 builds a breadthfirst spanning tree of the DFA digraph.

vii) DFA minimization. A sequence of equivalence relations and partitions is defined on the set of DFA states. Well-foundedness implies that the sequence of successive refinements reaches a fixed point, which is the solution to minimization.

viii) Proof that the language is not regular uses: indirect proof, pigeonhole principle.

Figure 5 lists the topics covered in the third section of the course. A similar consideration of the items in Figure 5 shows considerable support for coverage of the formal logic topics, as well as further use of inductive definition, recursion, and inductive proof. Additionally, graphs and trees are involved in the treatment of unification and of logic programming computation trees and termination analysis.

Coverage of the three sections of the problem-directed course implies coverage of all DS foundational topics, other than those topics relocated to other courses.
Our model can be viewed as an extension of the 2007 Report elsewhere in the curriculum. Recurrence relations are covered in analysis of algorithms. In both combinatorics are covered within the probability course, and basic combinatorics and recurrence relations. For our majors, the satisfactorily answer this concern for underclassmen. In fact, the question the necessity of studying the mathematics. It's hard to Structures, some computer science students invariably focus on other difficult problems, such as data mining or market basket analysis [12]. The invariant is organization around a significant computing application.

In our previous experience with math-centric versions of Discrete Structures, some computer science students invariably questioned the necessity of studying the mathematics. It's hard to satisfactorily answer this concern for underclassmen. In fact, the appropriate nature and extent of mathematics content remains an important subject of discussion in the discipline [4, 5, 11]. We believe that our focus on a useful, concrete computer science problem results in greater interest and achievement for computer science majors. For many of our students, the problem-directed course design answers the question “Why do we study math?” in the most convincing way—by experience.

8. REFERENCES