Proceedings of the Third International DERIVE/TI-92 Conference

Learning Discrete Mathematics with DERIVE

Nancy L. Hagelgans, Ursinus College, NHagelgans@acad.Ursinus.edu

Introduction

Students in my Discrete Mathematics course learn mathematics while they are working in cooperative learning groups on challenging projects with DERIVE. The course includes propositional and predicate logic, methods of proof, elementary number theory, mathematical induction and recursion, set theory, functions and relations, and some graph theory.

This course is offered at Ursinus College, a coeducational, independent, liberal arts college located in the suburbs of Philadelphia, Pennsylvania, USA. It is the required writing course in both the mathematics and computer science majors at the College. The class meets for three hours each week during a 15-week semester. Students usually enroll in the course during the second semester of the sophomore year, when they are typically 19 or 20 years old. Although the mathematics prerequisite is only one semester of single-variable calculus at the college level, most students have completed more advanced courses, such as the second semester of single-variable calculus, mutivariate calculus, and linear algebra. All the students have used DERIVE in calculus courses, and some students have studied the programming language C++ in computer science courses. The maximum class size is 20 students.

The Initial Experiences with DERIVE

During the first week of the Discrete Mathematics course, the class meets for one of the regular class hours in a computer laboratory. There the students have their first experience of the semester with DERIVE, and they are assigned to cooperative learning groups that will remain stable throughout the semester. Although there are more computers than students available, the groups, which each comprises four or five students, are instructed to use two adjacent computers and to discuss each step as they work through the questions. The initial assignment includes a review of the applicable DERIVE that the students have used in their calculus courses and an introduction to additional features, such as Boolean constants and variables, that they need in their study of propositional logic. The students work with predefined Boolean functions and with truth tables.

The student groups complete the assignment during group meetings that they arrange outside class time. The four questions on the top of the next page are questions that the students must answer as a part of the first assignment:

1. What order of precedence does DERIVE use for the five predefined logical operators?

2. Define the biconditional function **IFF** of two variables. For example, the expression **IFF(true, false)** should simplify to **false**. Check your definition of the function **IFF** with a truth table.

3. Use truth tables to determine whether each of the following is a contradiction, a tautology, or neither: $\sim p \wedge p$, $p \rightarrow \sim p$, $(p \rightarrow q) \leftrightarrow (q \rightarrow p)$, $(p \wedge \sim p) \rightarrow (p \vee \sim p)$.

4. Verify that the operator < on the set {-2, -1, 0, 1, 2} is not commutative.
   Hint: Simplify the expression:
   VECTOR(VECTOR(IFF (x < y , y < x), x, -2, 2), y, -2, 2).

All four questions require the examination of results of DERIVE computations to determine or verify answers. Students learn that truth tables and the VECTOR function efficiently produce many examples to examine. And the students define their first DERIVE function in the course. Thus the first experiences foreshadow the types of problems that the students will be required to solve in subsequent assignments.

Projects on the Concepts of Discrete Mathematics

After the introductory exercises using the predefined Boolean functions, many problems involve implementing the concepts of discrete mathematics using the numbers *0* and *1* to represent the Boolean constants *false* and *true*, respectively. Thus the students must translate mathematical concepts between two different mathematical systems. Since the binary operators are implemented as functions of two variables, the students learn to work with functional notation rather than the usual infix notation of certain operators. Instead of using the predefined Boolean functions, students define their own functions. During the semester, each cooperative learning group develops a collection of DERIVE functions that can be used in their projects.

The complete directions and problem statements for the second assignment of the course follow. As in subsequent assignments, the students must use DERIVE to verify their results, and they must write explanations of their function definitions. This is the first assignment that requires the use of the numbers *0* and *1* as Boolean values.

In order that we can use common names for functions that we define in DERIVE, we will override one default choice for input of function names. To allow case-sensitive names, make the following DERIVE menu choices:

Options Input Character Sensitive

at the beginning of each DERIVE session.

1. In many applications of logic, such as in digital circuit design, the integers **0** and correspond to the logic values **false** and **true**, respectively. With this correspondence in mind, we will develop a system of logic using these integers rather than logic values. Let **S = {0, 1}**. Use arithmetic operations to define DERIVE functions:

not: S → S
and: S × S → S

that behave as their names indicate. For example, some values of these functions should be:

not(0) = 1
and(1, 1) = 1
and(0, 1) = 0.

Verify that your functions yield correct results in all cases.

2. Suppose that you are given only the definitions of the logic operators **negation** and **conjunction** with truth tables. Define the operators **disjunction**, **conditional**, and **biconditional** in terms of the two given operators (without using truth tables).

3. Define DERIVE functions:

or: S × S → S
imp: S × S → S
iff: S × S → S.

Hint: Use the definitions developed for the two preceding questions.
Verify that your functions yield correct results in all cases.

The students are amazed to find that they can actually use the theorems that we have studied. After defining the first two functions using algebraic expressions:

not(p) := 1 - p and *and(p, q) := pq*,

the other functions can be defined directly from the laws of propositional logic. For example, the following definition of the function *or* is based on DeMorgan's law:

or(p, q) := not(and(not(p), not(q)).

Similarly, the students can define the conditional function *imp* and biconditional function *iff* essentially by stating theorems:

imp(p, q) := or (not(p), q) and *iff(p, q) := and (imp(p, q), imp(q, p))*.

When the defining expressions are simplified in DERIVE, their algebraic form is returned.

In the next assignment, both the predefined logic functions and the student-defined functions are used. Sets are represented by DERIVE vectors, and recursion is introduced by giving a recursive function definition for the students to examine. The computer science students, in a course on programming languages that they will take later, will recognize the similarities between this use of recursion applied to DERIVE vectors and the recursion applied to lists in the programming language LISP. The students are required both to test functions and to write explanations of the definitions of these functions. The student groups work outside scheduled class time on the following problems.

1. Define functions **FIRST** and **TAIL** that take a vector as input. The function **FIRST** returns the first component of the input vector, and the function **TAIL** returns the vector obtained by deleting the first component of the input vector. For example, **FIRST([15, 2, 8, 19]) = 15**, and **TAIL([15, 2, 9, 19]) = [2, 9, 19]**. Test your functions on the following vectors: [], [3], [4, 0, 1], and [true, false, false, false, false]. Write detailed descriptions of your definitions.

2. Define a function **CHECK** of two variables: the first variable is a vector **v** of two components and the second variable is a value **n**. The function checks to see of the first component of the vector **v** equals **n**. For example,

CHECK([1, false], 2) = false, and **CHECK ([1, false], 1) = true**.

Test your function **CHECK**, and write a complete description of your definition.

3. The function **VAL** defined below is a function of two variables: a predicate **p** expressed as a vector and a value **n**. This function evaluates the predicate **p** on **n** whenever **p** is defined on **n**. For example, if we consider:

p := [[0, true], [1, false], [2, true]],

then we want: **VAL(p, 0) = true**, **VAL(p, 1) = false**, and **VAL(p, 2) = true**.
The definition of **VAL** is recursive because its definition refers to itself:

```
VAL(p, n) := IF (DIMENSION(p) = 0, false, CHECK(FIRST(p), n)
        AND (FIRST(p)) SUB 2 OR (VAL(TAIL(p), n))
```

Test the function **VAL**, and write a complete description of how **VAL** computes the desired result. Try to think of a different definition for the function **VAL**.

4. Define functions **ALL** and **EXISTS** of two variables: a set **s** (represented as a vector) and a predicate **P**. The functions check to see if all or at least one member of the set **s** has the property **P**, respectively. For example, if we use **P** of question #3, we want the following to hold: **ALL([0, 2], P) = true**, **ALL([0, 1, 2], P) = false**, **EXISTS([1, 2], P) = true**, and **EXISTS([1], P) = false**.
Test your functions, and write complete descriptions of their definitions.

Now we will identify the logic constants **true** and **false** with the integers **1** and **0**, respectively.

5. Describe how the following vectors can be interpreted as predicates. What is the domain of each predicate?

```
IS_ODD1 := VECTOR( [i, MOD(i, 2)], i, 1, 10)
IS_EVEN1 := VECTOR( [i, 1 - MOD(i, 2)], i, 1, 10 )
IS_ODD2 := VECTOR( [i, MOD(i, 2)], i, -100, 100)
IS_EVEN2 := VECTOR( [i, 1 - MOD(i, 2)], i, -100, 100 )
```

6. Define a function **val** that corresponds to the function **VAL**.

7. Test the following function and describe how it computes the desired value:
exists(S, P) :=
STEP (SUM (val(P, S SUB i), i, 1, DIMENSION(S)) - 0.5)
For example, **exists([1,3,2], IS_ODD1 = 1**, and **exists([8,4,0,6], IS_ODD1) = 0**. Try to think of a different definition for the function **exists**.

8. Define an analogous function **all**. Test the function **all** and describe how the desired result is computed.

9. Use the functions **all** and **exists** with particular predicates **p** and **q**, and sets **s** to show that the following arguments are invalid:
If there exists x in S such that P(x), then for all x in S, P(x).
If for all x in S either P(x) or Q(x), then either for all x in S P(x) or for all x in S Q(x).
If for all x in S P(x) implies that for all x in S Q(x), then for all x in S P(x) implies Q(x).
Explain how your examples show that the arguments are invalid.

The next exercises are related to the students' study of sets. Students are required to define, test, and describe DERIVE predicates that test whether or not an element is a member of a set, one set is a subset of another set, two sets are equal, and two sets are disjoint. In addition, students define, test and describe DERIVE functions that return the union, intersection and difference of two sets. They test and describe a given doubly-recursive function **set** that removes any multiple occurrences of an element in a vector representing a set. Some student groups use mathematical rather than exclusively programming ideas to implement the functions. For example, one group used the function **set** and the function **intersection**, which they had just defined, in their definition of the function **is-subset**:

is_subset(v, w) := IF(intersection(v, w) = set(v), 1, 0, 0).

Their definition worked, but it depended on the particular definitions of the auxiliary functions to keep the order of the sets' elements consistent.

Further problems involve functions and relations with finite domains. These functions and relations are represented in DERIVE as vectors whose elements are vectors of dimension two. This representation of functions and relations implements their definitions as sets of ordered pairs of elements. Students define DERIVE predicates that test whether or not such a vector defines a function, an injection, a surjection, or a bijection, and they also define DERIVE functions that, given a function represented as such a vector, return the domain, range, restriction to a set, and value on an element of this function. Predicates on relations that test for reflexivity, symmetry, transitivity, and antisymmetry also are defined.

Explorations with DERIVE

In the Discrete Mathematics course, the methods of proof studied are illustrated with theorems of elementary number theory. Students use examples generated by DERIVE to investigate and to make conjectures as part of their study of number theory. Several such question follow.

1. Describe all integers **a**, **b**, and **c** for which the following statement is true:
   If **a** divides **c** and **b** divides **c**, then their product **ab** divides **c**. Use DERIVE to generate many examples, and then make a conjecture. Prove your conjecture.
   Hint: Use the **VECTOR** function to generate tables of the vector **[a, b, c, t]**, where **t** is the truth value of the given statement for the particular values of **a**, **b**, and **c**.

2. Suppose that **a** and **b** are integers. Consider the following equalities:

   GCD(a, b) = GCD(a + b, a - b) (*)
   2 * GCD(a, b) = GCD(a + b, a - b) (**)

   Use examples generated by DERIVE to help you answer the questions below. Then precisely state and prove your four conclusions.
   a. For which integers **a** and **b** does the equality (*) hold?
   b. For which integers **a** and **b** does the equality (**) hold?
   Hint: DERIVE commands such as the one below quickly generate many examples:

   VECTOR (VECTOR([a, b, IF (GCD(a, b)=GCD(a + b, a - b))], a, 1, 10), b, 1, 6)

No student group has completely solved the second problem. Although one group did make a correct conjecture with the most general solution, that group did not prove this conjecture correctly despite several submissions of proofs. Other groups have been able to prove their correct conjectures that did not provide the most general solution to the problem.

An assignment on mathematical induction requires the student groups to make conjectures about predicates defined on subsets of integers, and in particular, to discover the base case. Here again the VECTOR function allows the student groups to observe many cases. Two such problems that I have assigned follow:

1. Investigate the predicate: The positive integer **n** divides **(n - 1)! + 1**. Formulate a conjecture. Prove your conjecture.

2. Which Fibonacci numbers are even, and which are odd? After investigating with DERIVE, state and prove your conclusions.

Since DERIVE “knows” the standard closed forms of finite sums and products, such as

the sum of the first n positive integers, related problems are not suitable for student investigation. On the other hand, students are interested in hearing that these formulas are available and useful.

Conclusions

Students find these assignments very challenging, but most groups are highly successful. The students frequently work alone on the problems as preparation for the meetings of their groups so that they can use their time together effectively. The students work many hours on the problems, and they persist until they obtain correct results on difficult problems. While the students are working together in their cooperative learning groups, they are discussing mathematical ideas, and they are developing their problem-solving ability as they analyze each problem. Students must understand the involved mathematical concepts before writing correct DERIVE functions to implement these concepts.

Students generally have a positive attitude toward the class, and they are proud and relieved when they solve these problems. The computer algebra system DERIVE affords the capability of generating many examples for exploration, and it allows to students to verify their conjectures.

References

Baxter, Nancy; Dubinsky, Ed; Levin, Gary, *Learning Discrete Mathematics with ISETL*, Springer-Verlag, New York, 1989.

Epp, Susanna S., *Discrete Mathematics with Applications*, Second Edition, PWS Publishing Co., Boston, 1995.

Hagelgans, Nancy L., "Constructing The Concepts Of Discrete Mathematics with DERIVE", *The International DERIVE Journal*, V 2, No.1, 1995.

Hagelgans, Nancy; Reynolds, Barbara; Schwingendorf, Keith; Vidakovic, Draga; Dubinsky, Ed; Shahin, Mazen; Wimbish, Joseph, *A Practical Guide To Cooperative Learning In Collegiate Mathematics*, MAA Notes Number 37, Mathematical Association of America, 1995.