FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

AN INNOVATIVE APPROACH TO TEACHING STRUCTURAL INDUCTION FOR COMPUTER SCIENCE

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Irene Polycarpou

2008

To: Interim Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by Irene Polycarpou, and entitled An Innovative Approach to Teaching Structural Induction for Computer Science, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Peter J. Clarke

Malek Adjouadi

Vagelis Hristidis

Cengiz Alacaci

Ana Pasztor, Major Professor

Date of Defense: May 23, 2008

The dissertation of Irene Polycarpou is approved.

Interim Dean Amir Mirmiran
College of Engineering and Computing

Dean George Walker
University Graduate School

Florida International University, 2008

DEDICATION

I dedicate this dissertation with my deepest gratitude and love to my parents, who took on much more than their share to enable me to complete this effort. Without their patience, understanding, support, and more of all love, the completion of this work would not have been possible.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my dissertation advisor, Dr. Ana Pasztor, for her guidance, support, patience, encouragement, and most of all, open-mindedness throughout my graduate studies. Without her personal, technical, and editorial advice, as well as her confidence in my abilities, this dissertation would not have been possible. I would also like to thank her for the many things she taught me and for being a great mentor.

I am very grateful and indebted to all the members of my committee for their continued support, direction, and suggestions on my dissertation. Many thanks to Dr. Malek Adjouadi for financially supporting most of my graduate studies through the NSF grants CNS 0426125 and HRD 0317692 with the CATE center, Dr. Peter J. Clarke for taking on much more than his share to help and support me complete this dissertation, Dr. Cengiz Alacaci for helping and guiding me toward a qualitative methodology for the studies I have conducted as part of this dissertation, and Dr. Vagelis Hristidis for all his thoughtful suggestions and feedback on my dissertation.

My special thanks go to Dr. Yi Deng for his support and personal advise throughout my studies at the School of Computing and Information Sciences, Dr. Alex Pelin for helping me with the research studies I have conducted as part of this dissertation, and my friend Adeline Lenquette for helping me with the development of many of the animations that are part of the electronic book that I have developed as part of this dissertation.

Last but not least, I would like to thank my dear friends and colleagues Phanos Achilleos, Jose Andre Morales, Medha Bhadkamkar, and especially, John Christofides,

who helped and supported me in many ways throughout my graduate studies and encouraged and enabled me to complete this effort.

I gratefully acknowledge the financial support of NSF under grants CNS 0426125 and HRD 0317692 with the CATE center, the School of Computing and Information Sciences in the form of a graduate assistantship, and the University Graduate School in the form of a Dissertation Year Fellowship.

ABSTRACT OF THE DISSERTATION

AN INNOVATIVE APPROACH TO

TEACHING STRUCTURAL INDUCTION FOR COMPUTER SCIENCE

by

Irene Polycarpou

Florida International University, 2008

Miami, Florida

Professor Ana Pasztor, Major Professor

Proofs by induction are central to many computer science areas such as data structures, theory of computation, programming languages, program efficiency-time complexity, and program correctness. Proofs by induction can also improve students' understanding and performance of computer science concepts such as programming languages, algorithm design, and recursion, as well as serve as a medium for teaching them.

Even though students are exposed to proofs by induction in many courses of their curricula, they still have difficulties understanding and performing them. This impacts the whole course of their studies, since proofs by induction are omnipresent in computer science. Specifically, students do not gain conceptual understanding of induction early in the curriculum and as a result, they have difficulties applying it to more advanced areas later on in their studies.

The goal of my dissertation is twofold: 1. identifying sources of computer science students' difficulties with proofs by induction, and 2. developing a new approach to teaching proofs by induction by way of an interactive and multimodal electronic book (e-

book). For the first goal, I undertook a study to identify possible sources of computer science students' difficulties with proofs by induction. Its results suggest that there is a close correlation between students' understanding of inductive definitions and their understanding and performance of proofs by induction. For designing and developing my e-book, I took into consideration the results of my study, as well as the drawbacks of the current methodologies of teaching proofs by induction for computer science. I designed my e-book to be used as a standalone and complete educational environment. I also conducted a study on the effectiveness of my e-book in the classroom. The results of my study suggest that, unlike the current methodologies of teaching proofs by induction for computer science, my e-book helped students overcome many of their difficulties and gain conceptual understanding of proofs induction.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 The Research Problem

Proof by structural induction is one of the most powerful proof techniques used in computer science. It is most commonly used to prove that every element of an inductively defined set has a certain property. There is a special case of structural induction, where the inductively defined set involved in the proof is the set of all natural numbers. This is what we call proof by mathematical induction (induction on numbers). In the literature, the term "induction" refers to either mathematical induction or structural induction, or both. Here, I use the term "induction" to refer to both structural and mathematical induction.

In the field of computer science, there are many concepts (sets) that are defined inductively and for which we often need to prove their properties. Therefore, proofs by induction are at the core of many areas of computer science, such as data structures, theory of computation (Hopcroft et al., 2001; Barwise & Etchemendy, 1998; Sipser, 1997), programming languages (Pierce, 2002), program verification (Kaplan et al., 2004; Holland-Minkley, 2002; Bundy et al., 1991), program efficiency-time complexity (Cormen et al., 2001), and correctness of algorithms—be they recursive or non-recursive (Weiss, 2006; Page, 2003; Cormen et al., 2001; Krone & Feil, 2001; Best, 1996; Lynch, 1996).

Moreover, induction can improve students' understanding and performance of computer science concepts such as programming languages, recursion (Fressola & Krone,

2003; Wu, Dale, & Bethel, 1998), and algorithm design (Manber, 1989), as well as serve as a medium for teaching them. According to Bruce, Drysdale, and Kelemen (2003), "Programmers with a good understanding of mathematical induction find it much easier to write and, even more importantly, provide convincing arguments for the correctness of recursive algorithms" (p.5). In addition, the results of a study conducted by Page (2003) suggest that proofs by induction play a major role in software design and implementation.

Major professional computer science societies such as the Association for Computing Machinery (ACM), and the Institute of Electrical and Electronics Engineers, Inc. (IEEE), as well as the Curriculum Renewal Across the First Two Years (CRAFTY), the subcomittee of the Committee for the Undergraduate Program in Mathematics (CUPM) of the Mathematical Association of America (MAA), emphasize the necessity to include proofs by induction in the undergraduate computer science curriculum (Barker et al., 2004; Engel & Roberts, 2001).

Even though proofs in general are indispensable in the teaching of numerous computer science concepts, it is a well known secret that students have proof phobia. This makes it difficult for the instructors to effectively convey the often complex nature of proofs. Proofs by induction are no exception. Students are exposed to proofs by induction in many courses of their curricula, and yet it is well documented in the literature that in general, students have difficulties understanding and performing them (Polycarpou, 2006; Polycarpou, Pasztor, & Alacaci, 2006; Wu Yu, 2000; Sheard, 1998; Thompson, 1996; Baker, 1995; Movshovitz-Hadar, 1993; Lowenthal & Eisenberg, 1992; Dubinsky, 1989; Dubinsky 1986; Dubinsky & Lewin, 1986; Ernest, 1984; Brumfiel, 1974). Such difficulties include difficulties understanding the steps involved in a proof by induction,

the substance of the proof (seeing the proof as a convincing argument and not as a procedure to be followed), how to prove an if-then statement (which results in difficulties understanding the logic behind the induction step), the necessity to include the base case, and difficulties performing proofs by induction on problems that are not similar to the ones students encountered before.

Educators have been observing students' struggles to understand proofs by induction and have tried for decades to present them to the students in a way that facilitates their learning. Through the years, researchers who looked into students' difficulties with proofs by induction have proposed some solutions to mend the situation (Dubinsky, 1989; Dubinsky, 1986; Ernest, 1984), but little improvement has been achieved.

1.2 My Approach to Solving the Problem

The fact that computer science students have difficulties with proofs by induction impacts the whole course of their studies, since proofs by induction are omnipresent in the field of computer science. More precisely, students do not gain conceptual understanding of proofs by induction early in the curriculum and as a result, they have difficulties understanding and applying them to more advanced areas later on in their studies. For this reason, the major goal of my dissertation is to develop a new approach to teaching proofs by induction for computer science, in a way that facilitates computer science students' learning and helps them overcome their difficulties, which so far has not been done. To be able to do so, it was important to identify and take into consideration the conceptual sources of computer science students' difficulties with proofs by induction.

Study to identify sources of students' difficulties

As part of my dissertation, I conducted a study to identify possible sources of computer science students' difficulties with proofs by induction and in particular, difficulties that students may have *prior* to proving a statement by induction. My focus was on finding whether students' difficulties lie in proofs by induction, or there are other factors preventing them from understanding and appropriately applying the concepts presupposed in such proofs. More precisely, I backtracked students' difficulties to their understanding of the set theoretical concepts involved in and leading up to proofs by induction, including but not limited to structures, closed sets, inductive sets, and inductive definitions. Such concepts are presupposed in the so-called *Induction Principle* (p. 72) that justifies proofs by induction—a fact that is often not made explicit.

Based on my five year experience tutoring students in "Logic for Computer Science," a course in which computer science students are introduced to proofs by structural induction, and grading their exams and quizzes over these years, as well as feedback from and conversations with professors who teach proofs by induction for decades, I came to realize that students' difficulties with proofs by induction may arise from their lack of understanding of the implicit concepts leading up to proofs by induction, which are often ignored in the classroom.

After an extensive review of the literature on students' difficulties with proofs by induction, I concluded that nobody has looked at the role that the set theoretical concepts that are presupposed in proofs by induction (i.e., structures, closed sets, inductive sets, and inductive definitions) play in students understanding of such proofs. The existing literature focuses on difficulties students have *while* performing a proof by induction and

the psychological sources of such difficulties, as well as the discrete mathematics elements necessary for understanding and performing proofs by mathematical induction (Wu Yu, 2000; Baker, 1995; Dubinsky, 1989; Dubinsky 1986; Dubinsky & Lewin, 1986; Ernest, 1984).

For the above reasons, I decided to focus my study on the set theoretical concepts presupposed in proofs by induction and explore whether lack of understanding of them is a source of computer science students' difficulties with proofs by induction, or conversely, whether understanding them leads to better results in using proofs by induction.

My study was contextualized within the undergraduate computer science curriculum. As part of my study, I developed an instrument which I administered to a sample of undergraduate computer science students before and after formal instruction of induction.

The results of my study suggest that there is a close correlation between students' understanding of the set theoretical concepts presupposed in proofs by induction and students' understanding and performance of proofs by induction. Additionally, they suggest that students who have conceptual difficulties with proofs by induction are those who have difficulties understanding such concepts.

The "Conceptual route" of teaching induction

Taking into consideration the results of my study and the fact that the current methodologies for teaching proofs by induction either ignore or pay inadequate attention to the set theoretical foundations of proofs by induction, I propose that the teaching material of proofs by induction be based on what I call the "conceptual route" of teaching

proofs by induction, which is an operationalization of the Induction Principle (see, e.g., Enderton, 2001). The idea behind the conceptual route is to decompose proofs by induction into their fundamental building blocks, and teach them based on these building blocks. Therefore, the teaching material that the conceptual route follows is streamlined towards teaching the set theoretical concepts leading up to proofs by induction such as, structures, closed sets, inductive sets, and inductive definitions, so that students' understanding can emerge through the conceptual building blocks involved.

An electronic book for teaching/learning proofs by induction

Finally, as part of my dissertation I have designed and developed an interactive and multimodal electronic book (e-book) for teaching and learning proofs by induction. My e-book offers a complete educational environment. It is a standalone learning tool for students to learn proofs by induction, as well as a teaching tool for instructors.

Unlike the traditional route of teaching proofs by induction, the teaching approach of my e-book is based on the conceptual route mentioned above. It presents each set theoretical concept presupposed in proofs by induction (i.e., structures, closed sets, inductive sets, and inductive definitions) as well as proofs by induction in a separate chapter, along with several examples and interactive exercises.

I decided to develop an e-book, instead of a traditional textbook, because I wanted to present my teaching material not only in the form of text, but in the form of images, animations, and sound, as well. I wanted to create a multimodal learning environment—an environment where students can use more than one modality (e.g., visual modality, auditory modality, and kinesthetic modality). Even though students have preferred learning modalities, they all learn best when they use multiple modalities simultaneously

(Lutzinger, 1995; Dunn, 1988). According to Dunn (1988), "multiple modalities provide a responsive vehicle for learning" (p. 432). My purpose is to help students learn proofs by induction in a constructive way that will help them get the "big picture" of induction, and overcome their difficulties.

Moreover, there is a variety in students' learning styles (Thomas et al., 2002; Lutzinger, 1991), as well as their ability to learn and their learning speeds (Baldwin & Kuljis, 2000). Some students may need more time and more practice to understand a concept, while others may be fast learners and one example can be enough for them to understand. Considering that the number of students in a classroom keeps increasing, their learning style diversity cannot be easily addressed by an instructor. In contrast, I have designed my e-book to accommodate most of the learning styles. Since students will be able to use my e-book to study and practice on their own time and at their own pace, they will have the opportunity to learn according to their own needs.

In addition, with curricula becoming more and more demanding, instructors are being asked to teach more material during the same amount of time (Dunlap, 2001). This limits the time they have available for teaching each topic of the course and their opportunity to go over more examples and problems. My e-book saves classroom time for the instructors by exposing students to many examples and problems of a great variety. At the same time, it gives students the opportunity to spend an adequate amount of time focusing on the material that is harder for them to understand, and go over more examples.

Furthermore, my e-book includes several *interactive exercises* and examples on each basic concept, so that students can embody it. For each exercise, students are

provided with *hints (clues)* in case they have difficulties solving the exercise, as well as with *feedback* on their performance and detailed explanation of the *solution of the exercise*. Also, some of the exercises in my e-book are contextualized within computer science, so that students can see its relevance and importance in the field, and be motivated to learn it.

The overall purpose of my e-book is to help students understand induction in a way that allows them to apply it to problems that are not similar to the ones they solve in class, to connect it to their current knowledge, to understand its relevance and importance, and to recognize on their own when to apply and correctly apply it whenever appropriate. According to Dubinsky (1986), "the ultimate goal is that proof by induction becomes an integrated part of a student's mathematics repertoire and that the student is more or less able to decide when it might be useful to apply it in a given situation, without having been told beforehand that this is an induction problem" (p. 316).

An Exploratory Study on the effectiveness of the e-book in the classroom

I have conducted an exploratory study on the effectiveness of my e-book in the classroom. Participants of my study were two groups of undergraduate computer science majors. One group of students was taught proofs by induction through the traditional route, whereas the other group through my e-book. Both groups were taught by the same professor and they spent the same amount of time on proofs by induction. The results of my study suggest that indeed, my e-book helped students overcome many of their difficulties and gain conceptual understanding of proofs by induction. They suggest that students who were taught induction through my e-book performed better on inductive definitions and proofs by induction than students who were taught through the traditional

route. Moreover, they suggest that more students of those who were taught through my e-book gained conceptual understanding of proofs by induction, than of those who were taught through the traditional route.

1.3 Dissertation Outline

Proofs by induction are omnipresent in the computer science curriculum and therefore students come across them in many courses of their curricula. At the same time, it is well documented in the literature that in general, students have difficulties understanding and performing such proofs. Unfortunately, this has a broader impact on the whole course of studies of computer science students, since they do not understand proofs by induction early in the curriculum and as a result, face difficulties applying them to more advanced areas later on in their studies.

Chapter 2 presents the theoretical foundations behind proofs by induction and gives a brief overview of previous research done on students' difficulties with proofs by induction, as well as on proposed solutions for helping students overcome such difficulties. It also presents the reasons these solutions did not improve the situation.

To demonstrate the importance of proofs by induction in computer science, Chapter 3 discusses some of their computer science applications. Specifically, it presents examples of how proofs by induction can be applied to the areas of data structures, theory of computation, algorithm correctness, and programming languages.

Because of the importance of proofs by induction in computer science and the fact that the current teaching methodologies do not help students gain conceptual understanding of proofs by induction, I decided to focus my research on developing a new approach to teaching proofs by induction for computer science. My goal was to

facilitate computer science students' conceptual understanding of proofs by induction and help them overcome their difficulties. For developing such an approach, it was important to identify the difficulties that students face understanding and performing proofs by induction, and more importantly, to identify the sources of such difficulties. I have conducted a study to identify possible sources of computer science students' difficulties with proofs by induction, which I discuss in detail in Chapter 4.

Taking into consideration the results of my study and the relevant literature on proofs by induction, I have designed and developed an interactive and multimodal electronic book (e-book) for learning and teaching proofs by induction for computer science. This e-book provides a complete educational environment and it can be used by students to learn proofs by induction as well as by instructors to teach them. Chapter 5 presents the teaching approach I follow in the e-book, and discusses in detail the development and content of the e-book. Furthermore, it presents the results of an exploratory study I have conducted on the effectiveness of the e-book in the classroom, which suggest that the e-book was successful in helping computer science students gain conceptual understanding of proofs by induction and overcome many of their difficulties.

Finally, Chapter 6 discusses the conclusions of my research and some future research directions.

CHAPTER 2
THEORETICAL FOUNDATIONS AND LITERATURE OVERVIEW

2.1 Proofs by Structural Induction

Proof by induction is one of the most powerful proof techniques used in computer science, with a wide variety of applications (see chapter 3, p.24).

We can think of a proof as "a logical argument that one makes to justify a claim and to convince oneself and others" (Blanton & Stylianou, 2003, p.113). More formally, a proof is "a logically rigorous deduction of conclusions from hypotheses" (Dreyfus, 1990, as cited in Steen, 1999, p.273). Proofs by induction are most commonly used to prove a statement asserting that every element of an inductively defined set has a given property. Such a statement is usually of the form "Every element x in S has property P," where S is an inductively defined set.

Before going any further, let us have a closer look at inductive definitions, inductively defined sets, and proofs by induction. Let U be a non empty set (called the universe), B a subset of U (called the base set), I a set of indices, and for each $i \in I$, f_i an n_i-place operation on U, where $n_i \in IN$, and n_i is the arity of f_i (i.e., f_i: $U^{n_i} \rightarrow U$). An inductively defined set S is a subset of U whose elements are generated by the elements in B using the operations on U.

The *general schema of an inductive definition* of a set $S \subseteq U$ is as follows:

1) Base Clause: Every element in B is in S.
2) Inductive Clause: For every $i \in I$ and $x_1,\ldots,x_{n_i} \in U$, if $x_1, \ldots, x_{n_i}$ is in S, then $f_i(x_1, \ldots, x_{n_i})$ is also in S (i.e., S is closed under the operations in U).
3) Final Clause: No element of U is in S unless it has to be one by 1) or 2) above.

Let us look at an example. Let X = {♣, ●,◘, ♦}. Let U = X^* (the set of strings obtained by concatenating zero or more symbols from X). Let I = {1, 2}. Let B = {♣, ●}. Let f_1: $U \to U$ be the unary operation ($n_1 = 1$) defined as follows: $f_1(F)$ = ◘ F ♦, for all $F \in U$. Let f_2: $U^3 \to U$ be a ternary operation ($n_2 = 3$) defined as follows: $f_2(F, G, H)$ = F ◘ G ♦ H, for all F, G, $H \in U$. The following is an example of an inductive definition. It defines some imaginary objects that I will call "bricklings".

1) Base Clause: ♣, ● are bricklings.
2) Inductive Clause: If F is a brickling, then $f_1(F)$ = ◘ F ♦ is a brickling, as well, and if F, G, and H are bricklings, then $f_2(F, G, H)$ = F ◘ G ♦ H is a brickling, as well.
3) Final Clause: No string in X^* is a brickling unless it has to be one by 1) or 2) above.

Some examples of bricklings are ◘ ♣ ♦, ◘ ♣ ♦ ◘ ◘ ♣ ♦ ♦ ◘ ♣ ♦, and ◘ ◘ ● ♦ ♦. Notice that ◘ ♣, and ◘ ♦ ◘ are not bricklings. The set containing all the bricklings is an inductive set.

As I said earlier, proofs by induction are called for whenever we want to prove that every element of an inductively defined set has a certain property.

The *general schema of a proof by (structural) induction* is as follows:

1) Base Case: Prove that every element of $B \subseteq S$ has property P.
2) Induction Step: For every $i \in I$: Choose $x_1,\ldots,x_{n_i} \in S$ arbitrarily.
   Assume that $x_1,\ldots,x_{n_i}$ have property P (*Induction Hypothesis*).
   Prove that $f_i(x_1,\ldots,x_{n_i})$ has property P, as well.

To illustrate this schema, let us go back to our example of bricklings. Assume that we want to prove the statement: "Every brickling F has the following property P: F has an odd length."

For the sake of simplicity, let us denote the number of elements in F by #e(F). Note that F is a sequence of elements of the set {♣, ●,◘, ♦}.

Proof by induction:
Let F be an arbitrary brickling
1) Base case: Assume F is ♣ or F is ●. Obviously, the number of elements in F is odd, since #e(F) = 1.
   So, F has property P.
2) Induction Step:
   2a) Suppose F = f_1(G) = ◘ G ♦ for some brickling G.
   Induction Hypothesis: G has an odd length (#e(G) is odd), i.e., G has property P.
   Then, the number of elements in F is #e(G) + 2, which is also odd. This proves that #e(F) is odd. Therefore, F has property P.
   2b) Suppose F = f_2(K, G, H) = K ◘ G ♦ H for some bricklings K, G, and H.
   Induction Hypothesis: K, G, and H have an odd length (#e(K), #e(G), and #e(H) are odd), i.e., K, G, and H have property P.
   Then, the number of elements in F is #e(K) + #e(G) + #e(H) + 2, which is also odd. This proves that #e(F) is odd. Therefore, F has property P.

Since F was an arbitrary brickling and we covered all possibilities for the structure of F, we proved the proposition.
q.e.d.

The idea of proofs by induction may be new to the students, but it is not at all new to the world of mathematics. Some argue that the roots of proofs by induction reach all the way back to 370 BCE when the first proof by induction appeared in Plato's Parmenides (Kaplan et al., 2004; Wiznia, 2003), while others argue that they reach even further back to Socrates who introduced it into Greek philosophy. Even though there is a debate in the mathematical community over who invented and who formalized proofs by induction, most researchers agree that the first known proof by induction was published by Francesco Maurolico (an Italian mathematician and astronomer) in his book Arithmeticorum Libri Duo, in 1575. Maurolico used a proof by induction to prove that the sum of the first n odd integers is n^2 (Fressola & Krone, 2003; Wiznia, 2003). The

formal definition of proofs by induction as we know it today emerged through the work of great mathematicians such as Pascal (1654), De Morgan (1838), and Dedekind (1878), who picked up on and improved Maurolico's work (Shonk, 2003; Wiznia, 2003; Acerbi, 2000).

The notion of proofs by induction has arisen from the need to prove properties of natural numbers. For this reason, its first formalization was *mathematical induction* (induction on numbers). Through the years, the need to prove properties of other inductively defined structures led to the generalization of mathematical induction to what we call today structural induction. Therefore, mathematical induction is a specific case of structural induction, where the inductively defined set involved in the proof is the set *IN* of all natural numbers. The following inductive definition of natural numbers is an instantiation of the general schema of an inductive definition that I gave earlier (p.11):

> Let U = *IR* (the set of all real numbers), I = {1}, and B = {0}. Let f_1: $U \to U$ be the unary operation ($n_1 = 1$) defined as follows: $f_1(n) = n + 1$, for all $n \in U$.
> 1) Base Clause: 0 is a natural number.
> 2) Inductive Clause: If n is a natural number, then $f_1(n) = n + 1$ is also a natural number.
> 3) Final Clause: An element of U is a natural number if and only if it has to be one by 1) or 2) above.
>
> Note: This is just one possible inductive definition of natural numbers. Natural numbers can also be defined as a subset (up to isomorphism) of {0, 1, +}* generated from {0} by the operation "+1."

The *general schema of a poof by mathematical induction* is as follows:

> Statement: "Prove that every natural number n has property P."
>
> Let n be an arbitrary natural number.
> 1) Base Case: Assume that n = 0 and prove that 0 has property P.
> 2) Induction Step: Assume that $n = f_1(k) = k+1$ for some natural number k.
>
> Induction Hypothesis: k has property P.

Prove that k+1 has property P, as well.

Note: The above schema of a proof by mathematical induction is also known as *weak mathematical induction*.

For example, consider the statement "Every natural number n has the following property P: n^3- n is divisible by 3."

Proof by mathematical induction:

Let n be an arbitrary natural number.

1) Base Case: Assume that n = 0.
   0^3 - 0 = 0 and 0 is divisible by 3.
   Therefore, n has property P.
2) Induction Step: Assume that n = $f_1(k)$ = k + 1, where k is a natural number.
   Induction hypothesis: k^3- k is divisible by 3, i.e., k has property P.
   Show that $(k + 1)^3$ - (k + 1) is divisible by 3.
   $(k + 1)^3$ - (k + 1) =
   $(k^3 + 3k^2 + 3k + 1) – (k + 1)$ =
   $(k^3 – k) + 3(k^2 + k)$.
   Since $(k^3 – k)$ is divisible by 3 (by the Induction Hypothesis) and $3(k^2 + k)$ is divisible by 3 as well,
   we can conclude that $(k^3 – k) + 3(k^2 + k)$ is divisible by 3.
   Therefore, n has property P.

Since n can have only one of these two forms, we completed our proof that every natural number n has property P.

q.e.d.

In some cases, it is more convenient to have a stronger induction hypothesis for proving our claim than the one we have in the above schema of a proof by mathematical induction. There is another form of mathematical induction, which is called *strong mathematical induction*, that can be used in such cases. The only difference between weak mathematical induction and strong mathematical induction is the induction step. While in the induction step of a proof by weak mathematical induction we assume that a natural number k has a property P (induction hypothesis) and we prove that k+1 has property P, as well, in the induction step of a proof by strong mathematical induction we assume that all natural numbers less than or equal to k have property P (induction

hypothesis), and we prove that k+1 has property P, as well. Below is the schema of a proof by strong mathematical induction:

General schema of a poof by strong mathematical induction:

Statement: "Prove that every natural number n has property P."

Let n be an arbitrary natural number.
1) Base Case: Assume that n = 0 and prove that 0 has property P.
2) Induction Step: Assume that n = k+1 for some natural number k.
   Induction Hypothesis: Every $m \leq k$ ($m \in IN$) has property P.
   Prove that k+1 has property P, as well.

Strong mathematical induction is equivalent to weak mathematical induction (i.e., whatever we can prove by one, we can prove by the other). A proof of their equivalence can be found in Schach (1958, p.84).

Even though I did an extensive review of the literature on proofs by induction, I could not find anything on students' difficulties with proofs by structural induction. It seems that the existing literature focuses only on mathematical induction. My dissertation focuses on *structural induction*, instead of mathematical induction. The reason I decided to focus on structural induction is that understanding of structural induction entails understanding of mathematical induction, it justifies recursive definitions of functions (see Recursion Theorem, Enderton, 2001, p.39), and it is at the core of many important areas of computer science (see chapter 3, p.24).

In the literature, the term "induction" refers to either mathematical induction or structural induction, or both. Throughout my dissertation, I will use the term "induction" to refer to both structural and mathematical induction.

2.2 Documented Difficulties that Students have with Proofs by Induction

Even though students are exposed to proofs by induction in many courses of their curricula, they still have difficulties understanding and performing them—a fact well documented in the literature (Polycarpou, 2006; Polycarpou, Pasztor, & Alacaci, 2006; Wu Yu, 2000; Sheard, 1998; Thompson, 1996; Baker, 1995; Movshovitz-Hadar, 1993; Lowenthal & Eisenberg, 1992; Dubinsky, 1989; Dubinsky 1986; Dubinsky & Lewin, 1986; Ernest, 1984; Brumfiel, 1974). "Indeed, if you question students–even those who have had several mathematics courses–although almost all of them will have heard of induction, not many of them will be able to say anything intelligent about what it is, much less actually use it to solve a problem. Some students won't even recall any kind of problem to which induction can be applied." (Dubinsky, 1986, p.305).

In the last few decades various specific difficulties that students have with proofs by induction have been reported in the literature. According to Ernest (1984), there are six difficulties and misconceptions that students have with proofs by mathematical induction:

1) There is an ambiguity in the word "induction," since proofs by induction are based on steps that involve deductive, rather than inductive reasoning. This ambiguity can result in the confusion of the students.
2) Students have difficulties understanding an implication statement (A→B). More precisely, students have difficulties understanding that they have to assume the first part of the implication (antecedent, i.e., A), to prove the second part (consequent, i.e., B). This often leads them to the misconception that they need to assume what they have to prove.

3) Students have difficulties understanding quantifiers and free variables, resulting in their misconception that there is a circular reasoning behind a proof by mathematical induction.

4) Students often believe that the base case of a proof by mathematical induction is not necessary.

5) Students are not exposed to different areas to which proofs by mathematical induction can be applied, leading them to the misconception that proofs by mathematical induction are only used to prove the summing of finite series.

6) Students do not see the relevance of mathematical induction and its usages, which leads them to question why this concept was even adopted.

Moreover, Baker (1995) conducted a study that revealed several difficulties that students have while performing proofs by mathematical induction, as well as factors that affect students' performance with proofs by mathematical induction. He separated students' difficulties into:

1) *Procedural knowledge difficulties*, which include students' difficulties understanding the steps involved in a proof by mathematical induction, as well as lack of strategies required for performing the proof.

2) *Conceptual knowledge difficulties*, which include students' difficulties understanding the substance of the proof and seeing the proof as a convincing argument and not as a procedure to be followed.

3) *Mathematical resources difficulties*, which include students' difficulties understanding that for the induction step to be valid, the induction hypothesis must be true, and that

an example is not enough to prove the argument of the induction step, since it must be true for every case.

In addition to these difficulties, Baker (1995) found that examples play an important role in students' decisions about verifying a statement, and that students use problems they encountered before as templates to guide them through a new problem. The latter was accompanied by the fact that students had difficulties performing proofs by mathematical induction on problems that were not similar to the problems they encountered before. As I will show later (section 4.4, p.55), the results of my study on computer science students' difficulties with proofs by induction strengthen these last two findings, as well as Baker's (1995) findings that students have conceptual difficulties and difficulties understanding that an example is not enough to prove the argument of the induction step.

Not surprisingly, Baker (1995) also found that students with less experience with proofs by mathematical induction had a poor performance compared to students with more experience, and that poor mathematical background was a factor contributing to students' difficulties. Finally, he found no evidence that students' performance was affected by their beliefs, but he did find evidence that negative attitudes and feelings were accompanied by poor performance.

The results of a study conducted by Wu Yu (2000) validate Baker's (1995) findings that students have difficulties with procedural and conceptual understanding of proofs by mathematical induction. They also confirm Ernest's (1984) claim that students have difficulties understanding the implication involved in the induction step and the relevance of the base case. According to Wu Yu (2000), the reason for these two

difficulties is that students do not understand the difference between the validity of an implication statement (A→B) and the truth of the second part of the implication (the consequent, i.e., B), involved in the induction step, and they do not understand that the statement they are trying to prove must be true for all the elements, including those of the base set (base case).

Furthermore, the results of Wu Yu's (2000) study revealed that students have the misconception that the purpose of the induction step is the same as that of the whole proof, and that the induction hypothesis is the same as the result of the whole proof. Finally, she found that students' procedural understanding might be affected by their content knowledge, which, as I will show later (section 4.4, p.55), is also confirmed by the results of my study on computer science students' difficulties with proofs by mathematical induction.

2.3 Literature Overview of Proposed Solutions

The facts that proofs are a "nightmare" for students and students have proof phobia make it difficult for educators to effectively convey the often complex nature of proofs to the students. This is especially the case with proofs by induction. Educators have been observing students' struggles to understand proofs by induction and have tried for decades to present them to the students in a way that facilitates their learning. Through the years, researchers who looked into students' difficulties with proofs by induction have proposed some solutions to mend the situation (Dubinsky, 1989; Dubinsky, 1986; Ernest, 1984), but little improvement has been achieved.

According to Ernest (1984), there are three major behavioral skills that students need to have in order to be able to understand proofs by mathematical induction:

1) *the ability to prove the base case*, which includes students ability to prove that particular numbers have fix numerical properties, which, in turn, depends on their ability to perform substitution into algebraic expressions in a single variable.
2) *the ability to prove the induction step*, which includes students ability to prove implication statements by deducing a conclusion from a hypothesis, which, in turn, involves their ability to make deductions from algebraic identities and the ability to manipulate algebraic identities.
3) *the ability to present a proof by mathematical induction in the correct form*, which includes students ability to communicate their knowledge of the correct form of a proof by mathematical induction.

In Ernest's (1984) opinion, teaching based on these behavioral skills will improve students' understanding. Moreover, he goes one step further and based on his conceptual analysis of mathematical induction, he proposes to strengthen the teaching of concepts that are precursor to proofs by mathematical induction. Such concepts include elementary proofs, properties of natural numbers, implication (which is needed for the induction step), and inductively defined functions.

Dubinsky & Lewin (1986) take a different approach towards students' difficulties with proofs by mathematical induction. They believe that to understand proofs by mathematical induction, students need to construct a cognitive schema of mathematical induction ("a collection of mental objects and mental operations which the subject is able to perform on these objects," Dubinsky [1986], p.306). In their opinion, the reason for students' difficulties understanding proofs by mathematical induction is that the current teaching methodologies do not provide the necessary tools for the students to be able to

construct such a schema. After interviewing students who were learning mathematical induction, Dubinsky & Lewin (1986) developed a general description of the construction of the cognitive schema of mathematical induction. Based on this general description, the subschemas of "method of proof," "function," and "logical necessity" are present when the students begin to learn proofs by mathematical induction, and through the learning process they are expected to construct the subschemas of "Modus Ponens," "Explain Induction," "Apply Induction," and "Solve Problems."

Accordingly, Dubinsky (1986, 1989) proposes a new approach to teaching mathematical induction, one that involves activities to help students construct their own cognitive schema of mathematical induction. Based on his approach, students learn a programming language called SETL (or ISETL—a more interactive version of SETL), which helps the development of a good mathematical knowledge, and then work on different projects (writing programs) with SETL (or ISETL), where each individual project aims at the construction of a specific part of the proposed cognitive schema of mathematical induction. Dubinsky (1986, 1989) conducted two studies to examine the effectiveness of his approach to teaching mathematical induction. The results of his studies showed an improvement of students' understanding of mathematical induction. In addition, after interviewing the students, Dubinsky concluded that students have passed through all the phases of the cognitive schema of mathematical induction proposed by Dubinsky & Lewin (1986).

Even though Ernest (1984) and Dubinsky (1986, 1989) proposed ways to help educators overcome the obstacles that they are facing when teaching proofs by induction,

today's educators are still struggling with conveying the notion of proofs by induction effectively to their students.

Ernest (1984) created a network of interrelated concepts which, in his opinion, students need to master before learning proofs by mathematical induction. He did not, however, propose any new methodology for teaching these concepts. On the other hand, Dubinsky (1986, 1989) proposed a new way of teaching proofs by mathematical induction, but nobody has picked up on it. His approach is not easy to adopt and deploy in the classroom. It requires the use of an outdated programming language, which can only be integrated into the curriculum at great cost in time and effort.

Although Ernest and Dubinsky's suggestions have some drawbacks and so far have not been very successful in solving the problem, I based my approach to teaching proofs by induction in part on their recommendations. By enhancing and combining their approaches I developed a new teaching methodology to improve students' performance with proofs by induction.

CHAPTER 3

SIGNIFICANCE OF INDUCTION IN COMPUTER SCIENCE

Proofs by induction are omnipresent in the field of computer science. There are many important computer science concepts (sets) which are defined inductively, and for which we often need to prove their properties. For example, a number of data structures, such as the List and the Tree data structures, as well as graphs, regular expressions, context-free grammars, and programming languages are usually defined inductively (Weiss, 2006; Fressola & Krone, 2003; Page, 2003; Hopcroft et al., 2001). Regular expressions, context-free grammars, and the Tree data structure have a wide variety of applications in computer science, two of which are compiler construction and text-search applications (Appel, 2002; Hopcroft et al., 2001). Proofs by induction can also be used to prove properties of recursive programs (Manna, Ness, & Vuillemin, 1973).

Furthermore, proofs by induction are central to many areas of computer science, such as graph theory, programming languages (Pierce, 2002), theory of computation (Hopcroft et al., 2001; Barwise & Etchemendy, 1998; Sipser, 1997), program verification (Kaplan et al., 2004; Holland-Minkley, 2002; Bundy et al., 1991), program efficiency-time complexity (Cormen et al., 2001), and correctness of algorithms—be they recursive or non-recursive (Weiss, 2006; Page, 2003; Cormen et al., 2001; Krone & Feil, 2001; Best, 1996; Lynch, 1996).

Moreover, induction can serve as a medium for teaching recursion—a powerful problem solving tool and a programming technique (Fressola & Krone, 2003; Wu, Dale, & Bethel, 1998), and it is important for students' understanding and performance of

software development (Fressola & Krone, 2003; Page, 2003). A study conducted by Page (2003) shows that proofs by induction play a major role on software design and implementation. The results of his study suggest that learning proofs by induction in relation with their applications in software development improves students ability to design and develop algorithms in their data structure course, which according to Page, implies that proofs by induction can increase the effectiveness in the practice of software development.

In what follows, I chose some specific areas of computer science to present in detail how proofs by induction are applied.

3.1 Data Structures

According to Weiss (2006), a data structure is a representation of data along with the operations allowed on the data. In computer science data structures serve as input and output domains of programs. Algorithm designers often face the dilemma of choosing the most appropriate data structure for writing a specific algorithm. To make an effective decision, designers must have a good understanding of the definition and properties of each data structure available to them. Some data structures, such as the List and the Tree data structures are usually defined inductively, and to prove their properties proofs by induction are necessary.

3.1.1 The List Data Structure

Lists are used as building blocks for implementing other data structures such as the Stack and the Queue data structures. The following is an inductive definition of the List data structure (c.f. section 2.1, p.11):

Let U = {[,], |, i: $i \in I$}*, where I is a finite, non empty set, and let B = {[]}.
For all $i \in I$, let f_i: $U \to U$ be a unary operation ($n_i = 1$) defined as follows:
$f_i(L) = [i \mid L]$ for all $L \in U$.

1) Base Clause: [] is a list, called the empty list.
2) Inductive Clause: If L is a list, then for all $i \in I$, $f_i(L) = [i \mid L]$ is also a list.
3) Final Clause: No element of U is a list unless it has to be one by 1) or 2) above.

Let I = *IN*. Some examples of lists are [3 | [8 | []]], [5 | [6 | [4 | []]]], [4 | [3 | [2 | [1 | []]]]], and [24 | [10 | [3 | [67 | [50 | []]]]]].

Note: You can think of the list [3 | [8 | []]] as [3, 8], the list [5 | [6 | [4 | []]]] as [5, 6, 4], the list [4 | [3 | [2 | [1 | []]]]] as [4, 3, 2, 1], and so forth.

We define two functions *Append* and *Length*, so that given two lists L_1 and L_2, *Append*(L_1, L_2) concatenates the two lists, and given a list L, *Length*(L) returns the length of list L, i.e., the number of elements in the list.

1. *Append*:
   Let L be a List.
   1.1) *Append*([], L) = L
   1.2) For every $i \in I$ and a list L_1,
   Append([i | L_1], L) = [i | *Append*(L_1, L)].

2. *Length*:
   2.1) *Length*([]) = 0.
   2.2) For every $i \in I$ and a list L,
   Length([i | L]) = *Length*(L) + 1

Now, consider the following property P of a list L:

"*Length*(*Append*(L,L_2))=*Length*(L)+*Length*(L_2), for any list L_2."

Claim: All lists L have property P.

To prove the above claim, we use proof by induction on the structure of lists.

Proof by structural induction:
Let L be an arbitrary list.
1) Base Case: Suppose that L = []. Let L_2 be an arbitrary list.

Length(*Append*([], L_2)) = (by 1.1)
Length(L_2) =
0 + Length(L_2) = (by 2.1)
Length([]) + *Length*(L_2) =
Length(L)+*Length*(L_2).
Therefore, L has property P.

2) Induction Step: Suppose that L = [i | L_1], for some $i \in$ I, where L_1 is a list. Let L_2 be an arbitrary list.
   Induction hypothesis: For any list L_2', *Length*(*Append*(L_1, L_2')) = *Length*(L_1) + *Length*(L_2'), i.e., list L_1 has property P.
   Length(*Append*([i | L_1], L_2)) = (by 1.2)
   Length([i | *Append*(L_1, L_2)]) = (by 2.2)
   Length(*Append*(L_1, L_2)) + 1 = (by the Induction Hypothesis)
   Length(L_1) + *Length*(L_2) + 1 =
   (*Length*(L_1) + 1) + *Length*(L_2) = (by 2.2)
   Length([i | L_1]) + *Length*(L_2)
   So, *Length*(*Append*([i | L_1], L_2)) = *Length*([i | L_1]) + *Length*(L_2).
   Therefore, L has property P.

Since L was an arbitrary list and we covered all possibilities for the structure of L, we proved our claim.
q.e.d.

3.1.2 The Tree Data Structure

The Tree data structure is used by almost every operating system for storing files, and it is the best choice for compiler construction (abstract syntax tree), and text-search applications (text processing and searching algorithms) (Weiss, 2006). To demonstrate the relation between the Tree data structure and proofs by induction, I will use a specific type of the Tree data structure, called Binary Tree data structure. In a binary tree, no node can have more than two children. The following is an inductive definition of binary trees (c.f. section 2.1, p.11):

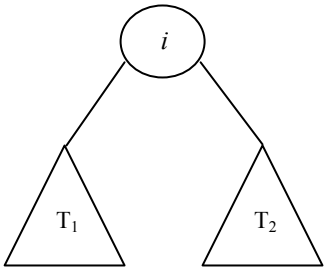
Let A = $IN \cup \{ /, \backslash, (,) \}$, U = A^*, and B = I = IN. For all $i \in$ I, let f_i: $U^2 \rightarrow$ U be the binary operation (n_i = 2) defined as follows: $f_i(T_1, T_2)$ = (T_1 / i \$T_2$) for all T_1, $T_2 \in$ U.

1) Base Clause: Any natural number n is a binary tree, called a node.

   Note: You can think of a node as 

2) Inductive Clause: If T_1 and T_2 are binary trees, then for every $i \in I$, $f_i(T_1, T_2) = (T_1 / i \backslash T_2)$ is also a binary tree.

Note: You can think of the natural numbers as labeled nodes of the tree and $f_i(T_1, T_2) = (T_1 / i \backslash T_2)$ as



3) Final Clause: No element of A^* is a binary tree unless it has to be one by 1) or 2) above.

We define two functions *Leaf* and *MaxLeaf*, where *Leaf*(n, T) is a Boolean function that accepts a number n and a tree T, and returns true if n is a leaf node (a node with no children) of tree T, and *MaxLeaf*(T) returns the leaf node of tree T that is the largest natural number.

1. *Leaf*:
   1.1) For any $n \in IN$ and any tree T, *Leaf*(n, T) returns true if and only if T = n.
   1.2) For any $n \in IN$, for all $i \in I$, and any trees T_1 and T_2, *Leaf*(n, $f_i(T_1, T_2)$) = *Leaf*(n, $(T_1 / i \backslash T_2)$) returns true if and only if *Leaf*(n, T_1) returns true or *Leaf*(n, T_2) returns true.

2. *MaxLeaf*:
   2.1) For any $n \in IN$, *MaxLeaf*(n) = n.
   2.2) For any trees T_1 and T_2 and for all $i \in I$, *MaxLeaf*($f_i(T_1, T_2)$) = *MaxLeaf*($(T_1 / i \backslash T_2)$) = *maximum*(*MaxLeaf*(T_1), *MaxLeaf*(T_2)), where the operation *maximum*($node_1$, $node_2$) returns the node that is the largest natural number of $node_1$ and $node_2$.

Let P be the following property of binary trees: "For every binary tree T, *MaxLeaf*(T) is a leaf node," i.e., *Leaf*(*MaxLeaf*(T), T) returns true. To prove property P, we use induction on the structure of binary trees.

Proof by structural induction:
Let T be an arbitrary binary tree.

1) Base Case: Suppose T = n, for some n $\in$ *IN*.
   Leaf(*MaxLeaf*(n), n) = (by 2.1)
   Leaf(n, n).
   By 1.1, *Leaf*(n, n) returns true.
   Therefore, T has property P.
2) Induction Step: Suppose T = (T_1 / *i* \T_2) for some *i* $\in$ I, where T_1 and T_2 are trees.
   Induction Hypothesis: *Leaf*(*MaxLeaf*(T_1), T_1) and *Leaf*(*MaxLeaf*(T_2), T_2) each return true, i.e., trees T_1 and T_2 have property P.
   Show that *Leaf*(*MaxLeaf*((T_1 / *i* \$T_2$)), ($T_1$ / *i* \$T_2$)) returns true.
   MaxLeaf((T_1 / *i* \T_2)) = (by 2.2)
   maximum(*MaxLeaf*(T_1), *MaxLeaf*(T_2)).
   By definition, the operation *maximum*($node_1$, $node_2$) must return $node_1$, or $node_2$.
   So, *maximum*(*MaxLeaf*(T_1), *MaxLeaf*(T_2)) will return either *MaxLeaf*(T_1) or *MaxLeaf*(T_2).
   So, *Leaf*(*MaxLeaf*((T_1 / *i* \$T_2$)), ($T_1$ / *i* \$T_2$))
   = *Leaf*(*MaxLeaf*(T_1), (T_1 / *i* \$T_2$)) or *Leaf*(*MaxLeaf*($T_2$), ($T_1$ / *i* \$T_2$)).
   Case 1: If *Leaf*(*MaxLeaf*((T_1 / *i* \$T_2$)), ($T_1$ / *i* \$T_2$)) = *Leaf*(*MaxLeaf*($T_1$), ($T_1$ / *i* \$T_2$)), then by 1.2 we get *Leaf*(*MaxLeaf*(T_1), T_1) or *Leaf*(*MaxLeaf*(T_1), T_2).
   By the induction hypothesis, we know that *Leaf*(*MaxLeaf*(T_1), T_1) is true, so *Leaf*(*MaxLeaf*((T_1 / *i* \$T_2$)), ($T_1$ / *i* \$T_2$)) returns true.
   Case 2: If *Leaf*(*MaxLeaf*((T_1 / *i* \$T_2$)), ($T_1$ / *i* \$T_2$)) = *Leaf*(*MaxLeaf*($T_2$), ($T_1$ / *i* \$T_2$)), then by 1.2 we get *Leaf*(*MaxLeaf*(T_2), T_1) or *Leaf*(*MaxLeaf*(T_2), T_2).
   By the induction hypothesis, we know that *Leaf*(*MaxLeaf*(T_2), T_2) is true, so *Leaf*(*MaxLeaf*((T_1 / *i* \$T_2$)), ($T_1$ / *i* \$T_2$)) returns true.
   Therefore, T has property P.

Since T was an arbitrary binary tree and we covered all possibilities for the structure of T, we proved the proposition.
q.e.d.

3.2 Theory of Computation

Theory of computation is an area of computer science that deals with whether a problem can be solved and how efficiently it can be solved on a computer (Hopcroft et al., 2001; Sipser, 1997). One subarea of theory of computation is automata theory. Automata theory is the study of abstract machines and the problems they can solve. Some

topics that are of importance in automata theory are finite automata, formal grammars, regular expressions, and Turing machines. Finite automata are used to model different hardware and software; formal grammars are used in the design of software that processes data with an inductive structure; regular expressions are used to denote the structure of such data (data with an inductive structure); and Turing machines are used to understand what we can expect from a software (Hopcroft et al., 2001). In what follows, I chose some specific areas of theory of computation to present in detail how proofs by induction are applied.

3.2.1 Regular Expressions

Regular expressions denote regular languages which can also be described by finite automata. They are essential in the areas of compiler construction (building a lexical analyzer) and text-search applications (searching for and recognizing patterns in a text), and also serve as the input languages for programs that process strings (Hopcroft et al., 2001). Regular expressions are inductively defined, and they are most commonly used to denote inductively defined languages (Fressola & Krone, 2003; Page, 2003; Hopcroft et al., 2001; Linz, 2001; Sipser, 1997).

The inductive definition of regular expressions R that denote a language L(R), is as follows (c.f. section 2.1, p.11):

> Let $U = \{\varepsilon, \varnothing, \alpha, +, *, (,): \alpha \in A\}^*$, where A is a finite, non-empty set of characters. Let $I = \{1, 2, 3, 4\}$, and $B = \{\varepsilon, \varnothing, \alpha: \alpha \in A\}$. Let $f_1: U^2 \to U$ be the binary operation ($n_1 = 2$) defined as follows: $f_1(R_1, R_2) = R_1 + R_2$, for all $R_1, R_2 \in U$. Let $f_2: U^2 \to U$ be the binary operation ($n_2 = 2$) defined as follows: $f_2(R_1, R_2) = R_1R_2$, for all $R_1, R_2 \in U$. Let $f_3: U \to U$ be the unary operation ($n_3 = 1$) defined as follows: $f_3(R) = R^*$, for all $R \in U$. Let $f_4: U \to U$ be the unary operation ($n_4 = 1$) defined as follows: $f_4(R) = (R)$, for all $R \in U$.

1) Base Clause: ε, $\emptyset$, and any symbol $\alpha \in A$, are regular expressions denoting the languages $\{\varepsilon\}$, $\emptyset$, and $\{\alpha\}$, respectively. That is, $L(\varepsilon) = \{\varepsilon\}$, $L(\emptyset) = \emptyset$, and $L(\alpha) = \{\alpha\}$.
2) Inductive Clause: If R_1 and R_2 are regular expressions denoting $L(R_1)$ and $L(R_2)$, respectively, then R_1+R_2 is a regular expression denoting the union of $L(R_1)$ and $L(R_2)$, that is, $L(R_1 + R_2) = L(R_1) \cup L(R_2)$; and R_1R_2 is a regular expression denoting the concatenation of $L(R_1)$ and $L(R_2)$, that is, $L(R_1R_2) = L(R_1)L(R_2)$.
If R is a regular expression denoting L(R), then R^* is a regular expression denoting the closure of L(R), that is, $L(R^*) = (L(R))^*$; and (R) is a regular expression denoting the same language as R, that is, L((R)) = L(R).
3) Final Clause: No element of U is a regular expression unless it has to be one by 1) or 2) above.

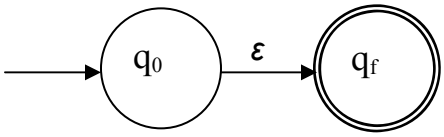
(Linz, 2001, p.72)

To prove properties of regular expressions denoting languages we use proofs by structural induction. For example, to prove that for every language L that is denoted by a regular expression R there is a finite automaton E that accepts L, we use induction on the structure of regular expressions.

Let P be the following property: "Every language L that is denoted by a regular expression R is also accepted by a finite automaton E."

Proof by structural induction:
Let R be an arbitrary regular expression.
Suppose L = L(R). We need to show that L = L(E), where E is an ε-NFA (a non deterministic finite automaton that allows ε transitions) with exactly one accepting state, no edges into the initial state, and no edges out of the accepting state.

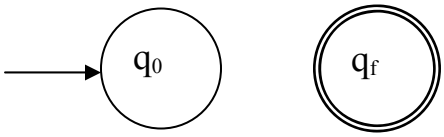
1) Base Case:
There are three cases we need to show in the base case:
1a) Suppose R = ε. The language denoted by ε is $\{\varepsilon\}$ ($L(\varepsilon) = \{\varepsilon\}$).
We need to construct a finite automaton E that accepts $L(\varepsilon)$.
Let E be the finite automaton below:



Finite automaton E

E consist of an initial state (q_0), a final state (q_f) and a transition $\delta(q_0, \varepsilon) = q_f$. In other words, from the initial sate with an ε-transition we move to the final state. E accepts only ε, therefore, E accepts L(R).
So, R has property P.

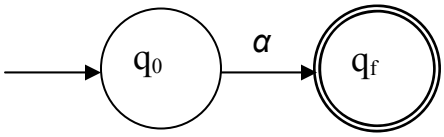
1b) Suppose R = $\varnothing$. The language denoted by $\varnothing$ is $\varnothing$ ($L(\varnothing) = \varnothing$).
We need to construct a finite automaton E that accepts $L(\varnothing)$.
Let E be the finite automaton below:



Finite automaton E

E consists of an initial state (q_0) and a final state (q_f), but it has no transitions from the initial state to the final state. E accepts nothing ($\varnothing$), therefore E accepts L(R).
So, R has property P.

1c) Suppose R = α for some symbol $\alpha \in S$. The language denoted by α is $\{\alpha\}$ ($L(\alpha) = \{\alpha\}$).
We need to construct a finite automaton E that accepts $L(\alpha)$.
Let E be the finite automaton below.

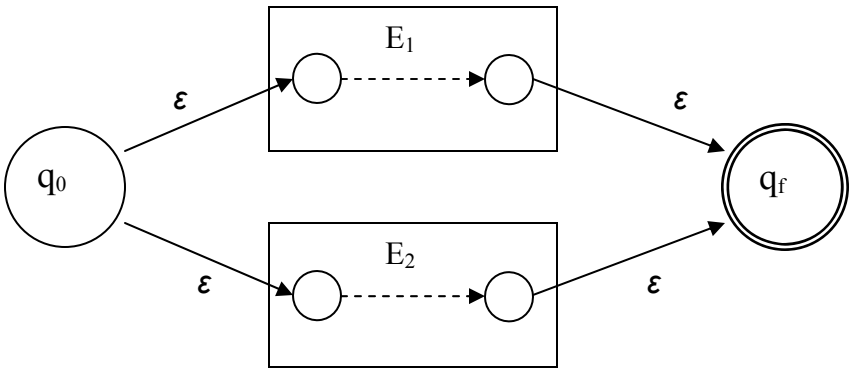


Finite automaton E

E consist of an initial state (q_0), a final state (q_f) and a transition $\delta(q_0, \alpha) = q_f$. In other words, from the initial sate we move to the final state with symbol α. E only accepts the string α, therefore E accepts L(R).
So, R has property P.

2) Induction Step:
There are four cases we need to show in the induction step:
2a) Suppose $R = R_1 + R_2$, for some regular expressions R_1 and R_2. The language denoted by $R_1 + R_2$ is the union of the languages denoted by R_1 and R_2, respectively ($L(R_1+R_2) = L(R_1) \cup L(R_2)$).
Induction Hypothesis: The languages denoted by R_1 and R_2 are accepted by finite automata, say E_1 and E_2, respectively, i.e., R_1 and R_2 have property P.

We need to construct a finite automaton E that accepts $L(R_1+R_2)$.
Let E be the finite automaton below:



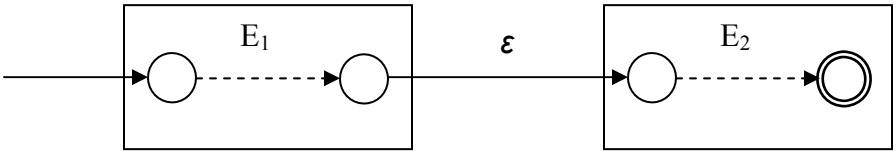
Finite automaton E

E is constructed from E_1 and E_2. E starts at initial state q_0 and with an ε transition can go to the initial state of E_1 or E_2. It can reach the final state of E_1 or E_2 through a path labeled by a string in $L(R_1)$ or $L(R_2)$, respectively. Once it reaches the accepting state of either of E_1 and E_2, it can reach the final state q_f with an ε transition.
Since E accepts $L(R_1 \cup R_2)$, we can conclude that R has property P.

2b) Suppose $R = R_1R_2$, for some regular expressions R_1 and R_2. The language denoted by R_1R_2 is the concatenation of the languages denoted by R_1 and R_2, respectively ($L(R_1R_2) = L(R_1)L(R_2)$).

Induction Hypothesis: The languages denoted by R_1 and R_2 are accepted by finite automata, say E_1 and E_2, respectively, i.e., R_1 and R_2 have property P.

We need to construct a finite automaton E that accepts $L(R_1R_2)$.
Let E be the finite automaton below:



Finite automaton E

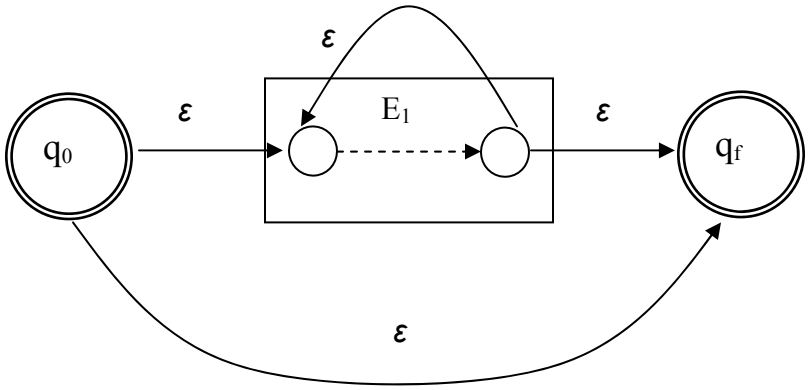
E is constructed from E_1 and E_2. The initial state of E is the initial state of E_1, and the final state of E is the final state of E_2. The only possible paths from the initial state to the final state is through E_1 (a path which is labeled by a string in $L(R_1)$) and then E_2 (a path which is labeled by a string in $L(R_2)$). So, the only paths in E from initial state to final state are the ones labeled by strings in $L(R_1)L(R_2)$.
Since E accepts $L(R_1R_2)$, we can conclude that R has property P.

2c) Suppose $R = R_1^*$ for some regular expression R_1. The language denoted by R_1^* denotes the closure of $L(R_1)$ $(L(R_1^*) = (L(R_1))^*)$.

Induction Hypothesis: The language denoted by R_1 is accepted by a finite automaton, say E_1, i.e., R_1 has property P.

We need to construct a finite automaton E that accepts $L(R_1^*)$.

Let E be the finite automaton below:



Finite automaton E

E is constructed from E_1. From its initial state (q_0), E has two possible moves. It can either go to its final state (q_f) through an ε transition, or go to the initial state of E_1. The fact that E can reach the final state with ε lets E accept the empty string (ε), which is in $L(R_1^*)$ no matter what R_1 is. In the second case, from the initial state, E can go to the initial state of E_1 from where it can go through the automaton E_1 one or more times and then go to the final state. This allows E to accept strings in $L(R_1)$, $L(R_1)L(R_1)$, $L(R_1)L(R_1)L(R_1)$, etc., which covers all the strings in $L(R_1^*)$.

Since E accepts $L(R_1^*)$, we can conclude that R has property P.

2d) Suppose $R = (R_1)$, for some regular expression R_1. The language denoted by (R_1) is the same language as R_1 $(L((R_1)) = L(R_1))$.

Induction Hypothesis: The language denoted by R_1 is accepted by a finite automaton, say E_1, i.e., R_1 has property P.

We need to construct a finite automaton E that accepts $L((R_1))$.

Since (R_1) denotes the same language as R_1 $(L(R_1) = L((R_1)))$, we can define our automaton E to be the same automaton as E_1.

Since E accepts $L((R_1))$, we can conclude that R has property P.

Since R was an arbitrary regular expression and we covered all possibilities of the structure of R, we proved the proposition.

q.e.d

(Hopcorft et al., 2001, p.102)

3.2.2 Context-Free Grammars

Context-free grammars are a more powerful tool for defining languages. In addition to the regular languages, they can also formally express recursive definitions of languages that cannot be denoted by regular expressions (Hopcroft et al., 2001). Such languages are called "context-free languages." Some of the most important applications of context-free grammars are in describing programming languages and data formats through document-type-definition (most commonly used in XML), as well as in compiler construction (implementing the parser) (ibid). In what follows, I demonstrate through an example how structural induction can be used in relation to context-free grammars.

Consider the language L_{pal} of palindromes (strings that read the same forward and backward, e.g., madam) over the alphabet {0, 1}.

So, if w is a string and w^r is the reverse of w (for example, if w = abc, then w^r = cba), then w is a palindrome if $w = w^r$. So, $L_{pal} = \{w \in \{0, 1\}^* : w = w^r\}$.

Consider the following inductive definition (c.f. section 2.1, p.11):

> Let A = {0, 1}, U = A^*, I = {1, 2}, and B = {ε, 0, 1}, where ε denotes the empty string. Let $f_1: U \to U$ be the unary operation ($n_1 = 1$) defined as follows: $f_1(w) = 0w0$, for all $w \in U$. Let $f_2: U \to U$ be the unary operation ($n_2 = 1$) defined as follows: $f_2(w) = 1w1$, for all $w \in U$.
> 1) Base Clause: ε, 0, and 1 are palindromes.
> 2) Inductive Clause: If w is a palindrome, 0w0 and 1w1 are palindromes, as well.
> 3) Final Clause: No element of A^* is a palindrome, unless it has to be one by 1) or 2) above.

We call the inductive set defined by the above rules Pal.

Note: Pal = L_{pal}.

To prove that Pal = L_{pal} we need to prove that Pal $\subseteq$ L_{pal} and that $L_{pal} \subseteq$ Pal.

Statement: Pal $\subseteq$ L_{pal}.

Proof by structural induction:
Let w be an arbitrary element in Pal.
To show that w is in L_{pal}, we perform a proof by induction on the structure of w.

1) Base Case: Assume that w is either ε, 0, or 1.
   By the definition of L_{pal} ε, 0, 1, are in L_{pal}.
   Therefore, w is in L_{pal}.
2) Induction Step: Assume that w = 0x0, or w = 1x1, where x is in Pal.
   Induction Hypothesis: x is in L_{pal}.
   By the induction hypothesis, we know that x is in L_{pal}. Since x is in L_{pal}, $x = x^r$. If $x = x^r$, then $0x0 = 0x^r0$ and $1x1 = 1x^r1$.
   So, $w = 0x0 = 0x^r0 = w^r$ $((0x0)^r = 0x^r0)$ and $w = 1x1 = w^r$ $((1x1)^r = 1x^r1)$.
   Therefore, w is in L_{pal}.

Since w was an arbitrary element in Pal and we covered all possible forms of w, we proved the proposition.

Statement: $L_{pal} \subseteq$ Pal.
Proof by strong mathematical induction:
Let w be an arbitrary string in L_{pal}.
To show that w is in Pal, we perform a proof by strong mathematical induction on the length of w.

1) Base Case: Assume that |w| = 0 or 1.
   If the length of w is 0 or 1, then w must be of the form ε, 0, or 1 (since ε denotes the empty string, |ε| = 0).
   Since ε, 0, and 1 are in Pal by the base clause of the inductive definition of Pal, we can conclude that w is in Pal.
2) Induction Step: Assume that |w| > 1.
   Induction Hypothesis: For all x in L_{pal}, if |x| < |w|, x is in Pal.
   Since w is in L_{pal}, it must begin and end with the same symbol ($w = w^r$), and since its length is greater than one, w must be of the form 0x0 or 1x1, where x is in L_{pal}.
   By the induction hypothesis x is in Pal. Since w is of the form 0x0 or 1x1 and x is in Pal, 0x0 and 1x1 are also in Pal (by the inductive clause of the inductive definition of Pal).Therefore, w is in Pal.

Since, w was an arbitrary string in L_{pal} and we covered all the possible lengths of w, we proved the proposition.
q.e.d

Now consider the following grammar G_{pal}, which expresses the above recursive definition of palindromes over {0, 1}:

G_{pal}(V, T, P, S}: the set of variables V = {0, 1}, the set of terminals T = {ε, 0,1}, the start symbol S = X, and the set of productions P = {X}, where X is defined as follows:

1. $X \rightarrow \varepsilon$
2. $X \rightarrow 0$
3. $X \rightarrow 1$
4. $X \rightarrow 0X0$
5. $X \rightarrow 1X1$

(Hopcroft et al., 2001, p. 171)

For a grammar G, the language L(G) (the language defined by G) is the set of all terminal strings that are derived from its start symbol:

$L(G) = \{w \text{ in } T^* \mid S \Rightarrow^* w\}$, where S is the start symbol, T the set of terminal symbols, and $\Rightarrow^*$ is the derivation relationship that represents zero or more steps ($\Rightarrow$ is the derivation relationship for a single step).

For example, consider the context-free grammar G_{pal} defined above and the language it defines $L(G_{pal})$. Let w = 0110110. We can derive w starting with X, therefore, w is a string in $L(G_{pal})$: $X \Rightarrow 0X0 \Rightarrow 01X10 \Rightarrow 011X110 \Rightarrow 0110110$ ($X \Rightarrow^* 0110110$).

If L is defined by some context-free grammar G, then L is said to be a context-free language.

For example, we said that G_{pal} defines the language of palindromes over {0, 1}. So, the set of all palindromes over {0, 1} is a context-free language.

To prove that $L(G_{pal})$ is the set of all palindromes over {0, 1} (i.e., $L(G_{pal})$ = Pal), we use a proof by induction. We need to prove that $Pal \subseteq L(G_{pal})$ and $L(G_{pal}) \subseteq Pal$.

> Statement: $Pal \subseteq L(G_{pal})$.
> Proof by structural induction:
> Let w be an arbitrary element in Pal.
> To show that w is in $L(G_{pal})$, we perform a proof by induction on the structure of w.
> 1) Base Case: Assume that w is either ε, 0, or 1.

Since there are productions $X \to \varepsilon$, $X \to 0$, and $X \to 1$, we can conclude that $X \Rightarrow^* w$ (from X we can derive w) in any of these cases.
Therefore, w is in $L(G_{pal})$.

2) Induction Step: Assume that w = 0x0, or w = 1x1, where x is in Pal.
   Induction Hypothesis: x is in $L(G_{pal})$, i.e., $X \Rightarrow^* x$ (we can derive x starting from X).

2a) w = 0x0.
   There is a derivation of w from X: $X \Rightarrow 0X0 \Rightarrow^* 0x0$ (by induction hypothesis we know that $X \Rightarrow^* x$).
   Therefore, w is in $L(G_{pal})$.

2b) w = 1x1
   There is a derivation of w from X: $X \Rightarrow 1X1 \Rightarrow^* 1x1$ (by induction hypothesis we know that $X \Rightarrow^* x$).
   Therefore, w is in $L(G_{pal})$.

Since w was an arbitrary element in Pal and we covered all possible forms of w, we proved the proposition.

Statement: $L(G_{pal}) \subseteq$ Pal.
To prove the above statement we need to perform a proof by induction on the number of steps in a derivation of a string w in $L(G_{pal})$ from X ($X \Rightarrow^* w$) and show that w is in Pal.

Since there is at least one step in a derivation of w from X, we need to perform a proof by induction on *positive natural numbers* ($IN^+ = \{n \in IN: n>0\}$), rather than on natural numbers (*IN*). The set of all positive natural numbers is also an inductive set. The inductive definition of positive natural numbers is as follows:

Let U = *IR*, I = {1}, and B = {1}. Let $f_1: U \to U$ be the unary operation ($n_1 = 1$) defined as follows: $f_1(n) = n + 1$, for all $n \in U$.

1) Base Clause: 1 is a positive natural number.
2) Inductive Clause: If n is a positive natural number, then $f_1(n) = n + 1$ is also a positive natural number.
4) Final Clause: An element of U is a positive natural number if and only if it has to be one by 1) or 2) above.

Now, let us prove our statement:

Proof by mathematical induction (on positive natural numbers):
Let w be an arbitrary string in $L(G_{pal})$, i.e., $X \Rightarrow^* w$.
As I have mentioned above, to show that w is in Pal, we perform a proof by mathematical induction on the number n of steps in a derivation of w from X.

1) Base Case: Assume that n = 1, i.e., from X we can derive w in one step.

If the derivation consists of one step, then it must use one of the productions $X \to \varepsilon$, $X \to 0$, or $X \to 1$. Since ε, 0, and 1 are in Pal, we can conclude that w is in Pal.

2) Induction Step: Assume that n = k+1 for some natural number k (k≥1), i.e., from X we can derive w in k+1 steps.

Induction Hypothesis: If $X \Rightarrow^* x$ in k steps (if we can derive x from X in k steps), then x is in Pal.

For the derivation of w to take k + 1 steps, it must be at least two steps. The only productions that allow that are $X \to 0X0$ and $X \to 1X1$. So, the derivation must be of the form $X \Rightarrow 0X0 \Rightarrow^* 0x0 = w$ or $X \Rightarrow 1X1 \Rightarrow^* 1x1 = w$.

We can derive x from X in k steps ($X \Rightarrow^* x$ in k steps), so by the induction hypothesis, we can conclude that x is in Pal. If x is in Pal, then 0x0 and 1x1 are also in Pal (by the inductive clause of the inductive definition of Pal).

Therefore, w is in Pal.

Since, w was an arbitrary string in $L(G_{pal})$ and we covered all the possible number of steps in a derivation of w from X, we proved the proposition. **q.e.d**

Note: Hopcroft et al. (2001) also proves this (p. 177), but in a different way. The difference is that they take it for granted that L_{pal} = Pal and therefore, they do not use structural induction. They instead use mathematical induction on the length of a string w over $\{0, 1\}^*$.

3.2.3 Context-Free Languages

As I mentioned in the previous section, context-free grammars are the most commonly used vehicle to define context-free languages. Moreover, context-free languages can also be defined inductively (although sometimes it is by mutual induction, which we do not discuss here), and to prove their properties we can use structural induction. One important category of languages that can be defined inductively is programming languages—a fact that is often not made explicit. An inductive definition of a programming language along with proofs of its properties by structural induction can be found in Pierce (2002).

In what follows, I present an example of how a simple context-free language can be defined inductively (rather than by a context-free grammar) and how structural induction can be used to prove its properties.

Consider the following inductive definition of a context-free language L (c.f. section 2.1, p.11):

> Let U = {a, b}*. Let B = ε, where ε is the empty string. Let f_1, f_2: $U^2 \rightarrow U$ be binary operations ($n_1 = n_2 = 2$) defined as follows: $f_1(F, G) = aFbG$ and $f_2(F, G) = bFaG$ for all $F, G \in U$.
> 1) Base Clause: $\varepsilon \in L$.
> 2) Inductive Clause: If strings $S_1, S_2 \in L$, then string $aS_1bS_2 \in L$ and string $bS_1aS_2 \in L$.
> 3) Final Clause: No element of U is in L unless it has to be one by 1) or 2) above.

Now, consider the following statement: "Every string $S \in L$ has the following property P: S has an equal number of a's and b's." For the sake of simplicity, we denote the number of a's and b's in S by #a(S) and #b(S), respectively. To prove the above statement, we use a proof by induction on the structure of strings in L.

> Proof by structural induction:
> Let S be an arbitrary string in L
> 1) Base Case:
> Suppose $S = \varepsilon$.
> #a(S) = #b(S) = 0. Therefore, S has property P.
> 2) Inductive Step:
> a) Suppose $S = aS_1bS_2$, where $S_1, S_2 \in L$.
> Induction Hypothesis: #a(S_1) = #b(S_1) and #a(S_2) = #b(S_2) (i.e., S_1 and S_2 have property P).
> #a(S) = #a(S_1) + #a(S_2) + 1.
> #b(S) = #b(S_1) + #b(S_2) + 1.
> So, by the induction hypothesis, #a(S) = #b(S). Therefore, S has property P.
> b) Suppose $S = bS_1aS_2$, where $S_1, S_2 \in L$. This case is the same as 2a).
> Since S was an arbitrary string in L and we covered all possibilities of the structure of S, we proved the statement.
> **q.e.d.**

3.3 Algorithm Correctness

An important area of computer science where proofs by induction are also used is algorithm correctness (Weiss, 2006; Page, 2003; Cormen et al., 2001; Krone & Feil, 2001; Best, 1996; Lynch, 1996). An algorithm is *correct* if it terminates and produces the correct output for every possible input (Cormen et al., 2001), meaning that it does what it is supposed to do according to its specifications. In what follows, I chose some specific algorithms to demonstrate how proofs by induction can be applied to prove their correctness. To prove the correctness of an algorithm, we have to prove that the algorithm terminates and that it produces the correct output for every possible input. Termination of an algorithm is not of interest in my dissertation, so I will only focus on proving that an algorithm produces the correct output for every possible input. One can think that we can do this by testing the output of the algorithm for all possible inputs, but, it cannot really help us prove the correctness of the algorithm for all inputs, since the set of all possible inputs can be infinite and we will not be able to test the algorithm with every possible input. That is where proofs by induction come into place.

3.3.1 A Simple Algorithm: The Factorial

Let us consider the following recursive algorithm which calculates the factorial of a natural number n:

```
Factorial (n)
1. if n = 0
2.    return   1
3. else
4.    return   n*Factorial (n-1)
```

Statement: For every natural number n (every possible input), Factorial(n) returns n! (produces the correct output).

Proof by mathematical induction:
Let n be an arbitrary natural number (arbitrary input).

1) Base Case: Assume that n = 0.
   According to lines 1 and 2 of the algorithm, Factorial(0) returns 1. We also know by the definition of factorial that 0! = 1.
   Therefore, Factorial(0) returns 0!
2) Induction Step: Assume that n = k+1, where k is a natural number.
   Induction Hypothesis: Factorial(k) returns k!
   We need to show that Factorial(k+1) returns (k+1)!
   Since n ≠ 0, lines 3 and 4 of the algorithm will be executed.
   According to line 4, Factorial(k+1) returns (k+1)*Factorial(k).
   By the induction hypothesis, we know that Factorial(k) returns k!.
   So, Factorial(k+1) returns (k+1)*k!
   By the definition of factorial, (k+1)! = (k+1)*k!
   Therefore, Factorial(k+1) returns (k+1)!

Since the possible inputs to Factorial(n) can have only one of these two forms, we have completed our proof that for every possible input $n \in IN$, Factorial(n) produces the correct output n!.
q.e.d

3.3.2 A More Complex Algorithm: The Quicksort

The Quicksort algorithm is used to sort a non-empty list of elements. The algorithm is as follows:

```
Quicksort ( A, p, r)
1. if p < r
2.     then pivot := Partition(A, p, r)
3.          Quicksort (A, p, pivot-1)
4.          Quicksort (A, pivot+1, r)

Partition (A, p, r)
x := A[p]
i := p - 1
j := r + 1
while True
  do repeat  j := j - 1
       until A[j] <= x
     repeat  i := i + 1
       until A[i] >= x
    if i < j
      then exchange A[i] <-> A[j]
   else return j
```

(Cormen et al., 2001, p.146)

Note: A is the list we want to sort, p is the first index of the list (or sublist), and r is the last index. The first call to our algorithm is Quicksort(A, 1, A.length()).We consider that the first index of a list is one and that the last index of a list is the length of the list, where the length of a list refers to the number of elements in the list. For example, if a list contains only two elements, then the index of the first element will be one and of the second (and last) element will be two.

To sort a list A, Quicksort(A, p, r) is called recursively with two sublists A[p…*pivot*-1] and A[*pivot*+1…r] as input, where *pivot* is an index chosen by the Partition(A, p, r) algorithm.

The Partition(A, p, r) algorithm accepts a list A[p…r], chooses an index that serves as the *pivot*, and rearranges the elements of A according to the element at the *pivot* position. The elements of A are rearranged into two sublists A[p…*pivot*-1] and A[*pivot*+1…r], where each element of A[p…*pivot*-1] is less than or equal to A[*pivot*], which is less than or equal to each element in A[*pivot*+1…r]. Partition(A, p, r) also calculates and returns the new index (after the rearrangement of the elements) of the element originally at *pivot* index.

For the sake of simplicity, I will omit the proof of the Partition(A, p, r) and assume that we already proved that it is correct (the proof of correctness of the Partition(A, p, r) algorithm can be found in Cormen et al., 2001, p.147-148).

To prove that Quicksort(A, p, r) correctly sorts list A, we use a proof by induction on the length of A (proof by induction on numbers). To be more precise, we use strong induction on numbers, which allows us to perform a "cleaner" proof.

Consider the following statement: For every list A[p…r] (every possible input), Quicksort(A, p, r) correctly sorts A[p…r] (produces the correct output).

Note: since A is a non-empty list (it contains at least one element), to prove the above statement we perform a proof by strong induction on positive natural numbers (IN^+), rather than on natural numbers (IN).

Proof by strong mathematical induction (on positive natural numbers):
Let A be an arbitrary list of length n.

1) Base Case: Assume that n = 1 (the length of A is 1, i.e., A contains only one element).
   This means that the call to the algorithm is Quicksort(A, 1, 1).
   Then, p = 1 and r = 1. So, p is not less than r (actually, p = r) and the statement in line 1 will fail (the algorithm will terminate).
   Since there was only one element in A, A is sorted.
   Therefore, Quicksort(A, p, r) correctly sorts list A.
2) Induction Step: Assume that n = m+1, where m is a positive natural number greater than 0 (A is of length m+1, i.e., A contains more than one element).
   Induction Hypothesis: Quicksort(A, p, r) correctly sorts any list A of length r, for all $1 < r \leq m$.
   We want to prove that Quicksort(A, p, m+1) correctly sorts list A, where A is of length m+1.
   The first call to the algorithm is Quicksort(A, 1, m+1). Then, the statement in line 1 is true, since p < r (1 < m+1). So, lines 2, 3, and 4 will be executed.
   By line 2, *pivot* = Partition(A, p, r).
   Since we assumed that we proved Partition(A, p, r) to be correct, Partition(A, p, r) will correctly rearrange the elements in A and will return the index *pivot*, on which the recursive call will be made on the sublists A[p…*pivot*-1], which is of length *pivot*-1, and A[*pivot*+1…m+1], which is of length m+1-*pivot* (lines 3 and 4).
   By induction hypothesis, every call to Quicksort(A, p, *pivot*-1) and Quicksort(A, *pivot*+1, m+1) will correctly sort A[p…*pivot*-1] and A[*pivot*+1…m+1], since both sublists have length less than or equal to m. (for the first sublist we have *pivot* – 1 < m+1 and for the second sublist we have m+1-*pivot* < m+1).

   In the last recursive calls to Quicksort(A, p, *pivot*-1) and Quicksort(A, *pivot*+1, r) p will not be less than r (actually p = r) and line 1 will fail (the algorithm will terminate), since there will be only one element in the sublists A[p…*pivot*-1] and A[*pivot*+1, r] (p = *pivot*-1 and *pivot*+1 = r).

Since the sorting of the sublists of A is performed directly on A, upon termination of the algorithm, A will be sorted.

Since A was an arbitrary list and we covered all the possible lengths of A, we proved the proposition.

q.e.d

3.3.3 An Algorithm that Manipulates Inductively Defined Objects

When we want to prove the correctness of an algorithm that manipulates inductively defined elements (objects), a proof by structural induction is the best approach, since it can provide a streamlined proof that follows directly from the structure of the algorithm. In what follows, we consider an algorithm on well-formed formulas (*wffs*), which are inductively defined.

The inductive definition of *wffs* is as follows (c.f. section 2.1, p.11):

Let $U = \{(,), \neg, \vee, A_i: i \in IN\}^*$, where the A_i's ($i \in IN$) are atomic sentences, $I = \{1,2\}$, and $B = \{A_i: i \in IN\}$. Let $f_1: U \rightarrow U$ be the unary operation ($n_1 = 1$) defined as follows: $f_1(x) = neg(x) = \neg x$, for all $x \in U$.
Let $f_2(x): U^2 \rightarrow U$ be the binary operation ($n_2 = 2$) defined as follows: $f_2(x,y) = or(x, y) = (x \vee y)$, for all $x,y \in U$.

1) Base Clause: All atomic sentences A_i ($i \in IN$) are *wffs*, called atomic formulas.
2) Inductive Clause: If F is a *wff*, then $neg(F) = \neg F$ is also a *wff*, called "negation of F." If G and H are *wffs*, then $or(G, H) = (G \vee H)$ is also a *wff*, called "disjunction of G and H."
3) Final Clause: No element of U is a *wff* unless it has to be one by 1) or 2) above.

Now, consider the following recursive algorithm on *wffs*, which returns true if the input string is a *wff* and false otherwise. Note: For the sake of simplicity (and because programs are finite), in our algorithm we consider A and B to be the only atomic sentences.

```
is_wff(String S)
1.   if (S.equals('A') || S.equals('B'))
2.            return true;
3.   else if (S.stratsWith('¬'))
4.            return is_wff(S.substring(1));
5.   else if (S.startsWith('(') && S.endsWith(')'))
```

```
6.          int pos = S.indexOf(' ∨ ');
7.          if (pos > -1)
8.               return(is_wff(S.substring(1,pos)) &&
                      is_wff(S.substring(pos + 1, S.length() - 1)));
9.     return false;
```

Also for simplicity, we will only consider the following claim on our algorithm:

Claim: For any input string S that is a *wff*, our algorithm (i.e., is_wff(S)) will return *true*.

For example, given the input string S = ($\neg$ A $\vee$ $\neg$ $\neg$ B), our algorithm will return *true*.

If the input string to the algorithm is a *wff*, then it can only be of one of three types: A or B, $\neg$ G, where G is a *wff*, or (G $\vee$ H), where G and H are *wffs*.

To prove our claim, we use induction on the structure of *wffs*.

Proof by structural induction:
Let F be an arbitrary *wff*.
1) Base Case:
   a) Suppose F = A. When we call our algorithm with F as the input string (i.e., is_wff(A)), line 1 will be satisfied and our algorithm will return *true* (line 2).
   b) The case F = B is the same as the above.

2) Induction Step:
   a) Suppose F = $\neg$ G, where G is a *wff*.
      Induction Hypothesis: For the input string G, our algorithm returns *true* (i.e., is_wff(G) returns *true*).
      When we call our algorithm with F as the input string (i.e., is_wff($\neg$ G)), line 3 will be satisfied and our algorithm will return the result of a call to itself with G as the input string (line 4).
      By the induction hypothesis, we know that our algorithm will return *true* for the input string G.
      Therefore, for *wff* F, our algorithm returns *true*.
   b) Suppose F = (G $\vee$ H), where G and H are *wffs*.
      Induction Hypothesis: For the input strings G and H, our algorithm returns *true* (i.e., is_wff(G) and is_wff(H) return *true*).
      When we call our algorithm with F as the input string (i.e., is_wff((G $\vee$ H))), line 5 will be satisfied, and since $\vee$ is included in F, variable *pos* will get the index position of $\vee$ in F (line 6). Then, our algorithm will return the result of a call to itself with G and H as the input strings (line 8).

By the induction hypothesis, we know that our algorithm will return *true* for both input strings G and H.
Therefore, for *wff* F, our algorithm returns *true*.
Since F was an arbitrary *wff* and we covered all possibilities for the structure of F, we proved the claim.
q.e.d.

3.4 Programming Languages

Programming languages are defined through syntactic and semantic rules that determine their structure and meaning, respectively. The purpose of the syntactic rules (syntax) of a programming language is to describe all the possible combinations of symbols that form a program, and the purpose of the semantic rules (semantics) is to give meaning to these combinations of symbols. There are different ways of defining the syntax of a programming language. One of them is through an inductive definition of the terms of the language (Pierce, 2002).

For the sake of simplicity, in the next example I will use a rudimentary programming language over the alphabet {*true*, *false*, 0, *succ*, *pred*, *iszero*, *if*, *then*, *else*}. The following is the inductive definition of the syntax of the terms of our programming language (c.f. section 2.1, p.11):

Let U = {*true*, *false*, 0, *succ*, *pred*, *iszero*, *if*, *then*, *else*}*, I = {1, 2, 3, 4}, and B = {*true*, *false*, 0}. Let f_1: U → U be the unary operation ($n_1 = 1$) defined as follows: $f_1(t)$ = *succ* t, for all t ∈ U. Let f_2: U → U be the unary operation ($n_2 = 1$) defined as follows: $f_2(t)$ = *pred* t, for all t ∈ U. Let f_3: U → U be the unary operation ($n_3 = 1$) defined as follows: $f_3(t)$ = *iszero* t, for all t ∈ U. Let f_4: U → U be the ternary operation (n_4=3) defined as follows: $f_4(t_1, t_2, t_3)$= *if* t_1 *then* t_2 *else* t_3, for all $t_1, t_2, t_3 \in$ U.

1) Base Clause: *true*, *false*, and 0 are terms.
2) Inductive Clause: If t_1 is a term, then $f_1(t_1)$ = *succ* t_1, $f_2(t_1)$ = *pred* t_1, and $f_3(t_1)$ = *iszero* t_1 are terms, as well. If t_1, t_2, and t_3 are terms, then $f_4(t_1, t_2, t_3)$ = *if* t_1 *then* t_2 *else* t_3 is a term, as well.
3) Final Clause: No element of U is a term, unless it has to be one by 1) or 2) above.

(Pierce, 2002, p. 26)

According to the above definition, some examples of terms are *succ succ* 0, *if iszero then true else false*, *succ pred succ* 0, *is zero true*, and *succ pred iszero true*. Notice that the above definition of terms only defines the syntax of our programming language and not the semantics. Therefore, some terms may be syntactically correct in our programming language, but semantically incorrect. For example, let *true* and *false* be Boolean constants, 0 a numeric constant, and *succ* an arithmetic operation (the successor operation). Then, *succ succ* 0 (the successor of the successor of 0) is syntactically as well as semantically correct. On the other hand, *succ succ true* (the successor of the successor of *true*) is syntactically correct, but semantically incorrect, since *succ* is an arithmetic operation and not a Boolean operation.

Now, consider the following property P of terms: "The number of distinct constants in t is no greater than the size of t (i.e., $|\text{Consts}(t)| \leq \text{size}(t)$)."

Consts(t) is the set containing all the constants in a term t, and size(t) is the size of t. A constant is any of the elements *true*, *false*, and 0, and the size of a term t is defined as follows:

size(t):

size(*true*) = size(*false*) = size(0) = 1.

size(*succ*(t)) = size(*pred*(t)) = size(*iszero*(t)) = size(t) + 1.

size(*if* t_1 *then* t_2 *else* t_3) = size(t_1) + size(t_2) + size(t_3) + 1.

Claim: Every term t has property P.

To prove the above claim, we use proof by induction on the structure of terms.

Proof by structural induction:
Let t be an arbitrary term.
1) Base Case: Assume that t is either *true*, *false*, or 0.

true, *false*, and 0 are all constants. So, $|Consts(t)| = |\{t\}| = 1$. Also, $size(t) = 1$. Therefore, t has property P ($1 \leq 1$).

2) Induction Step:

2a) Assume that t is of the form *succ* t_1, *pred* t_1, or *iszero* t_1, where t_1 is a term.

Induction hypothesis: Term t_1 has property P, i.e., $|Consts(t_1)| \leq size(t_1)$.

$Consts(t) = Consts(succ\ t_1) = Consts(t_1)$. So, $|Consts(t)| = |Consts(t_1)|$.
$size(t) = size(succ\ t_1) = size(t_1) + 1$.
By the induction hypothesis, $|Consts(t_1)| \leq size(t_1)$.
We also know that $size(t_1) < size(t_1) + 1$. So, $|Consts(t_1)| < size(t_1) + 1$, which implies that $|Consts(t)| < size(t)$.
Therefore, term $t = succ\ t_1$ has property P.
The same is true for terms *pred* t_1 and *iszero* t_1.
$Consts(t) = Consts(pred\ t_1) = Consts(t_1)$. So, $|Consts(t)| = |Consts(t_1)|$.
$size(t) = size(pred\ t_1) = size(t_1) + 1$.
By the induction hypothesis, $|Consts(t_1)| \leq size(t_1)$.
We also know that $size(t1) < size(t_1) + 1$. So, $|Consts(t_1)| < size(t_1) + 1$, which implies that $|Consts(t)| < size(t)$.
Therefore, term $t = pred\ t_1$ has property P.
$Consts(t) = Consts(iszero\ t_1) = Consts(t_1)$. So, $|Consts(t)| = |Consts(t_1)|$.
$size(t) = size(iszero\ t_1) = size(t_1)+1$.
By the induction hypothesis, $|Consts(t_1)| \leq size(t_1)$
We also know that $size(t1) < size(t_1) + 1$. So, $|Consts(t_1)| < size(t_1) + 1$, which implies that $|Consts(t)| < size(t)$.
Therefore, term $t = iszero\ t_1$ has property P.

2b) Assume that t is of the form *if* t_1 *then* t_2 *else* t_3, where t_1, t_2, and t_3 are terms.

Induction hypothesis: The terms t_1, t_2, and t_3 have property P, i.e., $|Consts(t_1)| \leq size(t_1)$, $|Consts(t_2)| \leq size(t_2)$, and $|Consts(t_3)| \leq size(t_3)$.

$Consts(t) = Consts(if\ t_1\ then\ t_2\ else\ t_3) = Consts(t_1) \cup Consts(t_2) \cup Consts(t_3)$.
$size(t) = size(if\ t_1\ then\ t_2\ else\ t_3) = size(t_1) + size(t_2) + size(t_3) + 1$.
So, $|Consts(t)| = |Consts(t_1) \cup Consts(t_2) \cup Consts(t_3)| \leq |Consts(t_1)| + |Consts(t_2)| + |Consts(t_3)|$.
By the induction hypothesis, $|Consts(t_1)| + |Consts(t_2)| + |Consts(t_3)| \leq size(t_1) + size(t_2) + size(t_3)$, since $|Consts(t_1)| \leq size(t_1)$, $|Consts(t_2)| \leq size(t_2)$, and $|Consts(t_3)| \leq size(t_3)$.
We also know that $size(t_1) + size(t_2) + size(t_3) < size(t_1) + size(t_2) + size(t_3) + 1$.
So, $|Consts(t_1)| + |Consts(t_2)| + |Consts(t_3)| < size(t_1) + size(t_2) + size(t_3) + 1$, which implies that $|Consts(t)| < size(t)$.
Therefore, term t has property P.

Since t was an arbitrary term and we covered all the possibilities of the structure of t, we proved the proposition.
(Pierce, 2002, p. 30)

CHAPTER 4

SOURCES OF COMPUTER SCIENCE STUDENTS' DIFFICULTIES WITH PROOFS BY INDUCTION: A STUDY

As I mentioned earlier (section 1.2), as part of my dissertation I conducted a study to identify possible sources of computer science students' difficulties with proofs by induction, and more precisely, to find whether students' lack of understanding of the set theoretical concepts presupposed in proofs by induction is a source of such difficulties (see also Polycarpou, Pasztor, & Alacaci, 2006 and Polycarpou, 2006). My study was contextualized within the undergraduate computer science curriculum.

4.1 Questions and Hypotheses

Question #1: How does students' understanding of an inductive definition and their performance on proofs by induction change after formal instruction?

Question #2: Do students have difficulties with the procedural or rather the conceptual aspects of proofs by induction? In other words, do students have difficulties understanding the steps involved in a proof by induction or do they have difficulties understanding the concepts involved in and leading up to proofs by induction?

Question #3: Do students who successfully perform proofs by induction have a deep and general understanding of induction, or do they perform them mechanically, and if so, to what extent?

Hypothesis #1: Students' performance with proofs by induction will improve after class instruction but still not be as it is desirable.

Hypothesis #2: There is a close correlation between students' understanding of inductive definitions and their performance of proofs by induction.

Hypothesis #3: Students will perform proofs by induction without a conceptual understanding of the process involved.

Hypothesis #4: Students' performance will be affected by their previous encounters with similar problems, as well as their everyday life experiences.

4.2 Participants and Procedures

As part of my study, I developed an instrument which I administered to a sample of computer science students twice: once before (*pretest*), and once after (*posttest*) formal instruction of induction. To minimize the possibility of students remembering the problem presented in the instrument, I administered the pretest two months prior to formal instruction of induction.

Participants of my study were 66 undergraduate (mostly junior) computer science majors taking a "Logic for Computer Science" course. I chose this course because it introduces students to proofs by structural induction. In addition, a prerequisite of this course is "Discrete Math," which introduces students to proofs by mathematical induction. In a "Logic for Computer Science" course, usually four class periods are spent on structural induction. My study was done as part of the course. Students were given participation or extra credit points for the class, contingent on making an honest effort to answer all questions in the instrument to the best of their knowledge. To ensure diversity of my population, I administered the instrument to two sections of the course taught by different professors.

Between pretest and posttest, students were first introduced to the inductive definition of well-formed formulas (*wffs*) and proofs by induction of their properties, followed by a number of similar inductive definitions and proofs. The students were then introduced to the general procedure involved in proofs by structural induction. This was

backed up by examples and homework problems, to amplify students' mastering of induction.

4.3 Instrument

I designed my instrument to check students' understanding of inductive definitions and its correlation with students' performance on proofs by induction. In the instrument, I defined inductively variable names (IPO-words) for a fictive programming language (IPO). The following text was presented at the beginning of the instrument:

> "*As most of you know by now, most programming languages have their own requirements for defining variable names. There is a new programming language out on the market, called IPO. In IPO, variable names are called IPO-words, and they are constructed from the characters of the set {a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z,* ***"****,* ***"****, _}, using the following rules:*
>
> *1. Any member of the set {a, …, z} is an IPO-word.*
> *2. If W is an IPO-word, then* ***"****W****"*** *is also an IPO-word (in other words, an IPO-word in quotes is also an IPO-word).*
> *3. If W 1 and W 2 are IPO-words, then W 1_W 2 is also an IPO-word (in other words, two IPO-words concatenated by underscore form a new IPO-word).*
> *4. Nothing is an IPO-word unless it has to be one by 1., 2., or 3. above.*
>
> *The length of an IPO-word refers to the number of characters in an IPO-word, where a character is any lower case letter* ***a*** *through* ***z****, an opening or closing quotation mark, or an underscore*."

Following this text and definition were six questions concerning the structure of IPO-words. I designed each question to test students' conceptual understanding of the definition of IPO-words.

> 1) First question: "*Identify which of the following strings are and which are not IPO-words and explain why. Mark the appropriate box.*
>
> *S1: Apple* □ *No* □ *Yes* _____
> *S2: A_p_p_l_e* □ *No* □ *Yes* _____
> *S3: a_p"_p_l"_e* □ *No* □ *Yes* _____
> *S4: "a_p"_"p_l_e"* □ *No* □ *Yes* _____

S5: a_"p_p"_l_e □ *No* □ *Yes* _____
S6: a_p""_p_l_e □ *No* □ *Yes* _____
S7: a_p"_"_p_l_e □ *No* □ *Yes* _____
S8: a_p_p_L_e □ *No* □ *Yes* _____
S9: "r_e_d"_"a_p_p_l_e" □ *No* □ *Yes* _____
S10: green_apple □ *No* □ *Yes* _____."

In order to correctly identify the above strings, students must have a good understanding of the inductive definition of IPO-words. Strings S4 (*"a_p"_"p_l_e"*), S5 (*a_"p_p"_l_e*), and S9 (*"r_e_d"_"a_p_p_l_e"*) are the only IPO-words.

I included strings S1 (*Apple*), S2 (*A_p_p_l_e*), and S10 (*green_apple*) to test the robustness of students' understanding of the definition, specifically, whether familiarity from earlier encounters with similar strings in other programming languages or just everyday English affects students' understanding. I expected more students to misidentify strings S1, S2, and S10 than any other string. I also included strings S2 (*A_p_p_l_e*) and S8 (*a_p_p_L_e*) to differentiate students' biases in cases where the first letter is uppercase (S2) from the cases where an uppercase letter appears in the middle of the string (S8). I expected more students to misidentify string S2 as an IPO-word than string S8.

> 2) Second question: "*What is the minimum length of an IPO-word? Give an example of an IPO-word of that length*."

To answer this question, students must understand the base clause in the definition of an IPO-word. The smallest length of an IPO-word is one—the length of an element of the set {a, …, z}.

> 3) Third question: "*Can you also give the maximum length of an IPO-word? Explain*."

To answer this question, students need to understand that the set of IPO-words, which is an inductive set, is infinite. There is no limit on the length of an IPO-word.

> 4) Fourth question: "*Try to construct an IPO-word of length greater than 6*."

Constructing an IPO-word of length greater than 6 forces students to use more than one rule of the definition of IPO-words, each more than once.

> *5)* Fifth question: "*Can you also construct an IPO-word of length equal to 8?*"

Here, I expected students to realize that they cannot construct such an IPO-word, because IPO-words can only have odd length. This is exactly what I asked them to prove in the next question.

> 6) Sixth question: "*Statement: For every IPO-word W, the following property P(W) holds: The length of W is odd (that is, W has an odd number of characters, where a character is any member of the set {a, …, z, "**,** "**,** _}).*
> *Do you think the above statement is correct? Whatever your answer is, try to prove it (Hint: use the rules for constructing an IPO-word).*"

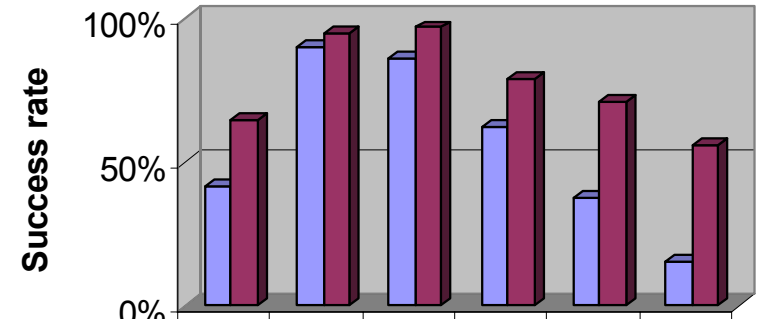
If a student correctly answers question five, I expect her/him to also realize that the given statement is correct. The aim of this question is to test whether a) students recognize that the proof of the statement calls for induction, and b) they are able to correctly perform the proof.

At the end of each question, I asked students to report the thoughts and feelings they experienced while answering the question. In Alacaci & Pasztor (2002), this information has provided powerful qualitative insights into students' thinking.

The full instrument along with the answers to its questions can be found in Appendix A, p.105.

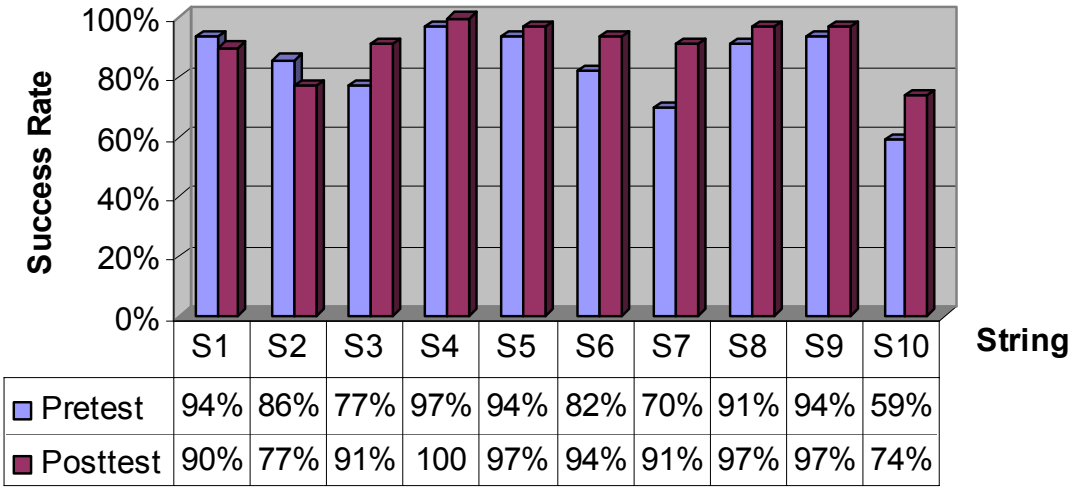
4.4 Results

I analyzed data from both pretest and posttest quantitatively, as well as qualitatively. My quantitative analysis was only based on students' answers, and its results are summarized in Graphs 1 and 2. The percentages in each graph are rounded to the nearest whole number. Graph 1 reports the percentages of students' correct answers in each question for the pretest and posttest, respectively. The percentage for the first question (Q1) represents the students who correctly identified all strings. The percentage for the sixth question (Q6) represents the students who agreed with the statement, recognized that they can prove it by induction, and did so correctly. Graph 2 shows a more detailed analysis of the first question, where the students' success rate for each individual string is presented for the pretest and posttest, respectively. My qualitative analysis was based on students' answers and comments, as well as their thoughts and feelings documented at the end of each question.



	Q1	Q2	Q3	Q4	Q5	Q6
Pretest	42%	90%	86%	62%	38%	15%
Posttest	65%	95%	97%	79%	71%	56%

Graph 1: Percentages of students' correct answers for each question.



	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Pretest	94%	86%	77%	97%	94%	82%	70%	91%	94%	59%
Posttest	90%	77%	91%	100	97%	94%	91%	97%	97%	74%

Graph 2: Percentages of students' correct answers for each string in the first question.

As we can see in Graph 1, there was an increase in the students' success rate in all of the questions in the posttest compared to the pretest. This suggests that *students' understanding of inductive definitions and their performance on proofs by induction improved after instruction*. However, this was not the case with some individual strings. According to Graph 2, my hypothesis for strings S2 (A_p_p_l_e) and S8 (a_p_p_L_e) was confirmed. In the posttest, string S2 had a 77%, while string S8 had a 97% success rate. Neither string is an IPO-word: each includes an "illegal" character (*A* and *L*, respectively). Moreover, the success rate for strings S1 (Apple) and S2 actually decreased after instruction, and string S10 (green_apple) had the lowest success rate. This suggests that even after formal instruction students had difficulties identifying these strings, which is again what I expected. Based on these results, I concluded that *students were influenced by their current (background) knowledge* on variable naming in other programming languages and/or on strings in their everyday language.

Furthermore, the results of my qualitative analysis suggest that students were also influenced by the *context in which the problem is presented*. For example, one student commented, "it is confusing to know that a variable name or IPO-word can have opening (") and closing (") quotes within it, although it is defined by rule 2." On average, students who made similar comments misidentified three out of nine strings in the first question, and incorrectly answered questions four, five, and six. The same was true for students who commented that the maximum length of an IPO-word depends on the computer's capacity and performance. Although these students realized that there is no length limit on IPO-words, they were influenced by the fact that there is a limit on the computer's ability to process variable names. Due to their context bias, they overlooked the fact that this question was purely theoretical, and they approached the solution from the implementation point of view, which was irrelevant to the question.

In addition, my results suggest that there is a close correlation between students' understanding of inductive definitions and their performance on proofs by induction. In the posttest 73% of the students understood the definition of IPO-words (i.e., correctly answered most of the questions), while 27% did not (i.e., misidentified at least two out of nine strings in the first question, and incorrectly answered at least two more questions). Furthermore, 71% of the students who understood the definition of IPO-words successfully proved the statement by induction (i.e., recognized that they can prove the statement by induction, and did so correctly). Finally, 83% of the students who did not understand the definition were not able to successfully perform the proof.

I further analyzed qualitatively the answers of the students who were not able to successfully prove the statement in the sixth question in the posttest (29 students). I

divided students into the following four categories (*A*-*D*), according to their conceptual level of understanding:

Category *A* (eight students): students who had conceptual understanding of both the process involved in the proof and the definition of IPO-words, but had difficulties articulating their thoughts, especially on the induction step.

The following is a typical example of a student's proof in this category:

> "*The minimum length of an IPO-word is one, for example* a *or* b.
> *In order to create a larger IPO-word I can either concatenate another length one IPO-word using underscore or use quotes, for example,* a_b *or* "a," *which both have length three.*
> *So, the length is going to be always odd, for example, |W| = 1, or 3, or 5, or 7, or 9, and so on*."

Students in this category correctly answered all of the previous questions.

Category *B* (four students): students who used problems encountered earlier in class as templates for the proof, had a conceptual understanding of the process involved in the proof, but did not have a conceptual understanding of the definition of IPO-words. At each step of the proof, they were able to make the necessary adaptations to the structure of IPO-words, but they had difficulties proving the induction step, which involves a conceptual understanding of the definition of IPO-words.

For example, one student wrote:

> "*Let F be an arbitrary IPO-word.*
> *1) Base Case: Assume that F is an IPO-word and we want to prove that P(W) holds.*
> *P(W) holds because an IPO-word can be a character, which is of length one. 2) Induction Step:*
> *2a) Suppose that F = "G" where G is an IPO-word.*
> *Induction Hypothesis – The IPO-word G has the property P(W).*
> *Then assume G holds P(W) and the addition of quotes, then it still holds the property P(W).*
> *So P(W) holds.*

2b) Suppose that F=W1_W2.
Induction Hypothesis – The IPO-word W1 has the property P(W) and the IPO-word W2 has the property P(W).
Since F was an arbitrary formula (underline mine) *and we have covered all possibilities for the structure of F, we have proved the proposition*."

Two students in this category answered question five incorrectly, while the other two answered all questions correctly.

Category C (four students): students who used problems encountered earlier in class as templates for the proof, but had no conceptual understanding of the process involved or the definition of IPO-words. Although they were able to retrieve a similar problem from their memory, they were not able to adapt it to the problem at hand.

For example, one student used the structure of *wffs* instead of that of IPO-words to prove the statement:

"*Let W be an arbitrary IPO-word.*
1) Let W = a*, where* a *is an IPO-word.*
Then the length of a *is one, which is odd. So P(W) holds.*
2a) Let W = ¬G, where G is an IPO-word. P(G) holds by rule 1.
Since P(G) holds, then P(¬G) means that the length of G is even. However, IPO-words must be constructed be rule 1, 2, or 3, so the length of "G" is odd.
a_b *is odd and* a_b_ *is even but it is not an IPO-word.*
2b) Let W = G∨H where G and H are IPO-words. Since G and H are IPO- words, P(G) and P(H) holds. By induction step 2a there is no IPO-word of even length. So, P(G∨H) holds."

This student answered all questions correctly. The other three incorrectly answered questions four and five and misidentified strings S2 and S10.

Category D (six students): students who just agreed with the statement, without even trying to prove it. Three students answered all previous questions correctly and three answered questions two and five incorrectly and misidentified strings S2 and S7.

Of the remaining seven students, three did not answer the question and four disagreed with the statement.

Finally, according to Graph 1 (Q5), in the pretest 62% of all students incorrectly answered question five (by constructing a string of length eight which they thought was an IPO-word). I expected these students to disagree with the statement that all IPO-words have odd length, but to my surprise, only 53% of these did so. This implies that 47% of these students contradicted themselves in those two questions. The situation did not improve much in the posttest. 29% of all students incorrectly answered question five, but only 63% of these students disagreed with the statement in the sixth question. Moreover, in the posttest 28% of the students who contradicted themselves in questions five and six (37%) not only agreed with the statement in the sixth question, but also successfully proved it by induction. Based on my qualitative analysis, I concluded that students who incorrectly answered the fifth question did not have a good understanding of the definition of IPO-words. This suggests that these students performed the proof without having a conceptual understanding of the process involved.

4.5 Summary

To sum up, the results of my study suggest that there is a close correlation between students' understanding of inductive definitions and their performance on proofs by induction. In addition, they suggest that some students perform proofs by induction without a conceptual understanding of the process involved. This latter finding refines the results of Baker (1995) and the claim of Lowenthal & Eisenberg (1992) and Thompson (1996) that students tend to apply induction in a "mechanical way". According to Thompson (1996), sometimes "students appear to be approaching the proof process very

algorithmically, having memorized a process instead of understanding its origin" (p. 479). Specifically, I found that students who had conceptual difficulties with proofs by induction were those who had difficulties understanding inductive definitions. This suggests that the latter is a major source of student's difficulties with proofs by induction.

According to Wu Yu (2000), students' performance with induction can be influenced by their current content knowledge. Based on my results, students' performance is additionally influenced by their previous experiences and the context in which the problem is presented. Although some students performed well on their class quizzes and homework problems on induction, they did not do as well on the problem presented in the instrument. Despite of the fact that the latter was very similar to the problems students encountered in class, it was situated in a different context. This suggests that students have difficulties generalizing proofs to new contexts.

In addition, the results of my study reinforce Baker's (1995) findings that examples play an important role for students' decisions about verifying a statement, and that students do not understand that an example is not enough to prove their arguments. As I mention in the previous section, students in category $\mathcal{A}$ used specific examples of IPO-words to prove their arguments of the induction step, but they did not prove them to be true for every IPO-word. Baker (1995) also pointed out that students use problems they encountered before as templates to guide them through a new problem, which is accompanied by the fact that students have difficulties performing proofs on problems that are not similar to the ones they encountered before. This was also the case with the students participating in my study, specifically students in categories $\mathcal{B}$ and $\mathcal{C}$. These students used problems they encountered earlier in class and more precisely, problems on

well-formed formulas, as templates for the problem at hand. They created analogies between well-formed formulas and IPO-words and used solutions to problems on well-formed formulas without correctly adapting them to IPO-words—a phenomenon that was observed often during the quizzes. According to Thagard (1998), analogical thinking can be very beneficial for problem solving and learning, if it is used appropriately. In my study, students were given a new problem (IPO-words) and they were able to retrieve a similar problem (well-formed formulas) from their memory, but they were not able to make the correct mapping between the relevant elements or they were not able to make the correct adaptations to the new problem. This suggests that students did not have a conceptual understanding of proofs by induction.

CHAPTER 5

A NEW APROACH TO TEACHING PROOFS BY INDUCTION: AN ELECTRONIC BOOK

As I mentioned previously, the major goal of my dissertation is to develop a new methodology for teaching proofs by induction for computer science in a way that will help computer science students overcome their difficulties and gain conceptual understanding of proofs by induction. I have designed and developed an interactive and multimodal electronic book (e-book) for learning and teaching proofs by induction for computer science. For developing my e-book as well as the teaching methodology I follow in my e-book, I took into consideration three things: 1) the results of my study on finding possible sources of computer science students' difficulties with proofs by induction, 2) the documented difficulties that students have with proofs by induction, and 3) the drawbacks of the current methodologies of teaching proofs by induction for computer science.

5.1 Current Approaches to Teaching Proofs by Induction.

It is a well known secret that students are afraid of proofs and they try to avoid them whenever possible. They perceive proofs as "a ritual without meaning" (Ball et al., 2002) and as something they have to learn as part of a course. Students' struggle to understand proofs, coupled with instructors being so "put off" with students' performances of proofs, leads instructors to often omit proofs of theorems or avoid asking students to perform any kind of proofs during their course (Epp, 2003), and to move

quickly to more "interesting" topics (Epp, 1996). Currently, many instructors teach proofs in a mechanical way, based on some underlying pattern. This way of teaching proofs has no effect on students' reasoning abilities, and in addition, it leads students to believe that proofs are all about memorizing some steps (Epp, 2003; Steen, 1999). Moreover, by asking students to perform a proof according to a certain pattern, instructors may be unwillingly reinforcing students' belief that there is no meaning behind proofs (Ball et al., 2002).

Proofs by induction are no exception. It has been documented in the literature that students have difficulties understanding and performing such proofs, even after repeated instruction in different courses of their curricula (Polycarpou, 2006; Polycarpou, Pasztor, & Alacaci, 2006; Wu Yu, 2000; Sheard, 1998; Thompson, 1996; Baker, 1995; Movshovitz-Hadar, 1993; Lowenthal & Eisenberg, 1992; Dubinsky, 1989; Dubinsky 1986; Dubinsky & Lewin, 1986; Ernest, 1984; Brumfiel, 1974). Even though induction is omnipresent in the field of computer science, and the idea of proofs by induction is reinforced in many courses of the computer science curriculum, it seems that the seeds are not there for students' understanding to grow.

Teaching of proofs by induction takes up too little of the current computer science curriculum. In most undergraduate computer science programs proofs by structural induction are taught as part of a "Logic for Computer Science" course. This is a one semester course in which on average three to four class periods are spent on proofs by structural induction. Reviewing some of the best textbooks which are being used to teach "Logic for Computer Science" (e.g., Gallier, 2003 ; Reeves & Clarke, 2003; Ben-Ari, 2001; Huth & Ryan, 2001; Burris, 1998; Mendelson, 1997; Schöning, 1989, to mention

just a few), I found that the current practices of teaching proofs by induction do not present the conceptual foundations of proofs by induction. They present proofs by induction as a "recipe" to be followed, without explaining the purpose and the role of each "ingredient". To be more precise, they do not connect proofs by induction to the Induction Principle that justifies proofs by induction, and they either ignore or pay inadequate attention to the set theoretical concepts, such as structures, closed sets, inductive sets, and inductive definitions, presupposed in proofs by induction. As a result, students accept proofs by induction as a valid proof technique without having any knowledge of the Induction Principle, and they memorize the steps involved in proofs by induction without understanding the rationale behind them (Thompson, 1996). According to Epp (2003), simply memorizing abstract concepts and learning to apply them mechanically has no impact on students' broader reasoning powers.

In addition, some textbooks present proofs by structural induction only in a specific context. For example, they present them only in relation to well-formed formulas (Burris, 1998; Schöning, 1989), which is another topic covered in a "Logic for Computer Science" course. According to Hatano (1996), when learners construct their knowledge, it is usually domain specific to the context in which the learning occurs, which leads learners to not be able to apply their knowledge in other domains (Hatano, 1996, as cited in Henderson et al., 2001).

Moreover, according to the results of the Third International Mathematics and Science Study, instructors focus more on teaching students how to do mathematics, rather than on understanding what they are doing (NCES 96, as cited in Steen, 1999). Additionally, most educators share what Gibbs (1994) calls the "disseminating

knowledge" view. According to this view, instructors are more interested in covering the appropriate material in class, than ensuring students' conceptual understanding of the material. This is also the view that most instructors who teach proofs by induction share today. With such a small time frame allocated to the teaching of proofs by structural induction (usually three to four class periods), and with limited time available through the whole course and a lot of material to be covered, instructors choose to move towards "easier" concepts, rather than struggle to convey the idea of proofs by induction to the students.

According to Baldwin & Kuljis (2000), many authors criticize formal instruction for leading students to develop misconceptions and misunderstandings. Regarding proofs by induction, students have the following misconceptions and misunderstandings:

1) Students have the misconception that there is no need to prove the base case and that in the induction step they have to assume what they have to prove (Wu Yu, 2000; Cuoco & Goldenberg, 1992; Ernest, 1984). This is a result of the current teaching practices, which do not accommodate students' conceptual understanding of the steps involved in proofs by induction. If students do not understand the rationale behind the base case, then they cannot see why they have to prove it. The same applies to the induction step. If students do not understand the rationale behind the induction step, and more precisely, the induction hypothesis, then they cannot see the difference between the assumption of the induction hypothesis and the claim of the whole proof.
2) Students have the misconception that an example or several examples are sufficient to prove an argument (specifically, the argument of the induction step) (Epp, 2003; Ross, 1998; Baker, 1995). According to Epp (2003), this may be due to the fact that

often instructors omit a proof of a statement and they rely on examples to justify it to the students. She also argues that instructors may be unwillingly conveying to the students the impression that empirical evidence is sufficient to prove a statement.

3) Students have the misconception that proofs by induction are only used to prove properties of certain structures (Ernest, 1984), and they have difficulties performing proofs by induction on problems that are not similar to problems they encountered before (Baker, 1995). This misconception arises from the current teaching practices of presenting proofs by induction only in relation to specific structures (e.g., well-formed formulas), and not exposing students to several examples and problems from different domains.

4) Students do not understand that to prove the implication involved in the induction step, it suffices to assume the first part of the implication (antecedent) and prove the second part (consequent) (Epp, 2003; Hoyles & Küchemann, 2002; Raffalli & David, 2002; Wu Yu, 2000; Ernest, 1984). According to Wu Yu (2000), this is the result of students' misunderstanding of the difference between the validity of an implication statement and the truth of the consequent. This is a more general problem that also applies to areas other than computer science, as well as to students' everyday life. For students to fully understand and appreciate the use of implication Hoyles & Küchemann (2002) recommend that its teaching includes activities that "focus on developing meanings for the structural properties of logical implication in mathematics" and "sustains and develops these meanings over time" (p. 219).

5.2 A New, Conceptual Approach to Teaching Proofs by Induction: "The Conceptual Route."

According to Dubinsky (1989), "Induction itself presents specific cognitive obstacles and students will continue to be unsuccessful with induction as long as teaching methodology continues to ignore these difficulties" (p. 285).

As I have said in the previous section, neither the ones who theorize about teaching, nor the ones who write textbooks for teaching "Logic For Computer Science" are concerned with conveying to students the conceptual foundations of proofs by induction that are needed for students to get the "big picture" and develop a conceptual understanding of proofs by induction. This impacts the whole course of studies of computer science students, since proofs by induction are omnipresent in the field of computer science. More precisely, computer science students do not gain conceptual understanding of induction early in the curriculum and as a result, they have difficulties understanding and applying it to more advanced areas later on in their studies. Presenting proofs by induction to computer science students as a step-by-step procedure to be followed is not enough for them to gain conceptual understanding and get the "big picture" of induction.

Moreover, the results of my study on finding possible sources of computer science students' difficulties with proofs by induction (see section 4.4, p.55) suggest that there is a close correlation between students' understanding of the set theoretical concepts presupposed in proofs by induction and students' understanding and performance of proofs by induction. Additionally, they suggest that students who have conceptual

difficulties with proofs by induction are those who have difficulties understanding such concepts.

Accordingly, I recommend that the teaching material for proofs by induction be based on what I call the "conceptual route" of teaching proofs by induction (Polycarpou, Pasztor, & Adjouadi, 2008), which is an operationalization of the Induction Principle (see, e.g., Enderton, 2001). The aim of the conceptual route is to shift students' focus from the syntactic form of proofs by induction to their substance, and not to just impose proofs by induction on the student like the current methodologies do. Specifically, I propose that the teaching material on proofs by induction be streamlined towards teaching the set theoretical concepts presupposed in proofs by induction, so that students' understanding can emerge through the conceptual building blocks involved. I propose that educators introduce students to the concepts of structures, closed sets, inductive sets, and inductive definitions, then connect these concepts to the Induction Principle, and finally connect the Induction Principle to proofs by induction. The set theoretical "story" behind the conceptual route of teaching proofs by induction that needs to be told for the students to get the "big picture" of proofs by induction unfolds as follows (c.f. Enderton, 2001):

> Note: Even though the literature uses the term "inductive set," here I use the term "B-inductive set," because I wish to make clear for the students that there is a base set (B) from which the inductive set is build up.
>
> A *structure* is a sequence $\alpha = \langle U, f_i \rangle_{i \in I}$, where U is a non empty set, I a set of indices, and for each $i \in I$, f_i an n_i-place operation on U, where $n_i \in IN$, and n_i is the arity of f_i (i.e., f_i: $U^{n_i} \rightarrow U$).
>
> If α is a structure and $S \subseteq U$, then we say that S is *closed* in α, if for all $i \in I$ and for all $x_1, \ldots, x_{n_i} \in S$, $f_i(x_1, \ldots, x_{n_i}) \in S$. If $B \subseteq U$, then we say that S is *B-inductive* in α, if $B \subseteq S$ and S is closed in α.

Let α be a structure, $B \subseteq U$, and $C(B) = \cap \{S: S \subseteq U$ and S is B-inductive in α$\}$. Note that C(B) is a non-empty intersection.

Lemma: *C(B) is the smallest B-inductive set in* α.

Proof of the above Lemma:
To show that C(B) is the smallest B-inductive set in α, we first need to show that C(B) is B-inductive in α, and then show that it is the smallest such set.

1) Show that C(B) is B-inductive in α:
To show that C(B) is B-inductive in α, we need to show that $B \subseteq C(B)$ and C(B) is closed in α.
1a) Show that $B \subseteq C(B)$.
C(B) is the intersection of all subsets S of U ($S \subseteq U$) that are B-inductive in α. Since C(B) is a non-empty intersection and B is a subset of every B-inductive subset of U, it is also a subset of their intersection. Therefore, B is a subset of C(B) ($B \subseteq C(B)$).
1b) Show that C(B) is closed in α.
To show that C(B) is closed in α, we need to show that for every $i \in I$ and every n_i-tuple $(x_1, \ldots, x_{n_i})$, if $x_1, \ldots, x_{n_i} \in C(B)$, then $f_i(x_1, \ldots, x_{n_i}) \in C(B)$.
Let $i \in I$ and $x_1, \ldots, x_{n_i} \in C(B)$ be arbitrary.
Since C(B) is the intersection of all S that are B-inductive in α, $x_1, \ldots, x_{n_i} \in S$ for all such S (by the definition of intersection).
Knowing that $x_1, \ldots, x_{n_i} \in S$ for all S that are B-inductive in α, and that every such S is closed in α, we can conclude that $f_i(x_1, \ldots, x_{n_i}) \in S$ for all such S (by the definition of closed sets).
Finally, knowing that $f_i(x_1, \ldots, x_{n_i}) \in S$ for all S that are B-inductive in α, we can conclude that $f_i(x_1, \ldots, x_{n_i}) \in C(B)$, by the definition of intersection.
Since we chose $i \in I$ and $x_1, \ldots, x_{n_i} \in C(B)$ arbitrarily, and we have shown that $f_i(x_1, \ldots, x_{n_i}) \in C(B)$, we can conclude that C(B) is closed in α.
By steps 1a) and 1b) we know that $B \subseteq C(B)$ and C(B) is closed in α, therefore, C(B) is B-inductive in α.

2) Show that C(B) is the smallest B-inductive set in α:
Since we already proved that C(B) is B-inductive in α, the only thing left is to show that it is a subset of every B-inductive set in α, which is a direct consequence of the fact that it is the intersection of all S that are B-inductive in α.

Let $\alpha = \langle U, f_i \rangle_{i \in I}$ be a structure and let $B \subseteq U$.
The set of all elements generated from B in α is defined by the following inductive definition:

1) Base Clause: For all $x \in B$, x is generated from B in α.
2) Inductive Clause: For all $i \in I$ and $x_1, \ldots, x_{n_i} \in U$, if $x_1, \ldots, x_{n_i}$ are generated from B in α, then $f_i(x_1, \ldots, x_{n_i})$ is also generated from B in α.
3) Final Clause: Any element of U is generated from B in α *only if* it is so by 1) or 2) above.

We denote by $C_*(B)$ the set of all elements generated from B in α.

Statement: The set of all elements generated from B in α ($C_*(B)$) is a B-inductive set in α. Moreover, it is the smallest B-inductive set in α.

Proof of the above statement:

1) Prove that $C_*(B)$ is *B*-inductive in α.
To prove that $C_*(B)$ is *B*-inductive in α, we need to show that $B \subseteq C_*(B)$ and that $C_*(B)$ is closed in α.

1a) Show that $B \subseteq C_*(B)$.
According to the base clause of the definition of $C_*(B)$, all the elements of B are in $C_*(B)$. Therefore, $B \subseteq C_*(B)$.

1b) Show that $C_*(B)$ is closed in α.
According to the inductive clause of the definition of $C_*(B)$, for each $i \in I$ and $x_1, \ldots, x_{n_i} \in U$, if $x_1, \ldots, x_{n_i}$ are generated from B in α, then $f_i(x_1, \ldots, x_{n_i})$ is also generated from B in α. So, $C_*(B)$ is closed in α, by the definition of closed sets.

By steps 1a) and 1b) we know that $B \subseteq C_*(B)$ and $C_*(B)$ is closed in α, therefore, $C_*(B)$ is B-inductive in α.

2) Prove that $C_*(B)$ is the smallest B-inductive set in α.
To prove that $C_*(B)$ is the smallest B-inductive set in α, it suffices to show that $C_*(B) \subseteq C(B)$.
To prove that $C_*(B) \subseteq C(B)$, we need to show that for all $x \in C_*(B)$, $x \in C(B)$.
Proof:
Let x be an arbitrary element of $C_*(B)$, i.e., let x be an arbitrary element generated from B in α.

By the final clause of the definition of $C_*(B)$, any element generated from B in α is either in B (base clause) or is generated from B by repeated applications of the inductive clause.

Case 1: $x \in B$.

This implies that $x \in C(B)$, since $B \subseteq C(B)$.

Case 2: $x \notin B$.

This implies that for some $n \geq 1$, x is generated from B by n applications of the inductive clause.

Lemma: For any $n \geq 1$ and any element y that is generated from B by m applications of the inductive clause, where $1 \leq m \leq n$, y is in C(B).

Proof by strong mathematical induction on n:

1) Base Case: Let $n = 1$ (y is generated from B by one application of the inductive clause).

Then, $y = f_i(x_1,\ldots, x_{n_i})$ for some $i \in I$ and some $x_1, \ldots, x_{n_i} \in B$.

Since $x_1,\ldots, x_{n_i} \in B$ and $B \subseteq C(B)$, $x_1,\ldots, x_{n_i} \in C(B)$.

Knowing that $x_1,\ldots, x_{n_i} \in C(B)$ and C(B) is closed in α, we can conclude that $f_i(x_1,\ldots, x_{n_i}) \in C(B)$.

2) Induction Step: Let $n = k+1$ for some $k \geq 1$.

Then, $y = f_i(x_1,\ldots, x_{n_i})$ for some $i \in I$ and some $x_1,\ldots, x_{n_i}$ that are generated from B by m applications of the inductive clause, where $1 \leq m \leq k$.

Induction Hypothesis: Any element y that is generated from B by m applications of the inductive clause, where $1 \leq m \leq k$, is in C(B).

By the induction hypothesis, $x_1,\ldots, x_{n_i} \in C(B)$.

Knowing that $x_1,\ldots, x_{n_i} \in C(B)$ and C(B) is closed in α, we can conclude that $f_i(x_1,\ldots, x_{n_i}) \in C(B)$.

By the above Lemma, we can conclude that $x \in C(B)$.

Since we proved that for all $x \in C_*(B)$, $x \in C(B)$, we can conclude that $C_*(B) \subseteq C(B)$, and therefore, $C_*(B)$ is the smallest B-inductive set in α.

The set of all B-legal elements in α ($C_*(B)$) is the smallest B-inductive set in α, therefore $C_*(B) = C(B)$.

The Induction Principle is a direct consequence of the fact that $C_*(B) = C(B)$.
Let α be a structure and $B \subseteq U$.

Induction Principle:

> *If S is a B-inductive subset of $C_*(B)$, that is, $B \subseteq S$ and S is closed in α, then $S = C_*(B)$.*

Proofs by induction are about proving properties of inductively defined sets, specifically of the elements of $C_*(B)$.
Let P be a property.
To prove that every element $x \in C_*(B)$ has property P, we use the Induction Principle as follows.
Theorem T: *Every $x \in C_*(B)$, has property P*.
Schema of a Proof by Induction:
Let $S_p = \{x: x \in C_*(B) \text{ and } x \text{ has property P}\}$. Then, $S_p \subseteq C_*(B)$.
Theorem T claims that $S_p = C_*(B)$. How do we prove it?
By the Induction Principle, we only need to show that S_p is B-inductive:
a) Base Case: Show that $B \subseteq S_p$. In other words, all elements in B have property P.
b) Inductive Step: Show that S_p is closed in α, i.e., for all $i \in I$ and $x_1,\ldots, x_{n_i} \in U$, if $x_1, \ldots, x_{n_i} \in S_p$, then $f_i(x_1, \ldots, x_{n_i}) \in S_p$.
In other words, let $i \in I$ and $x_1, \ldots, x_{n_i} \in C_*(B)$ be arbitrary.
Induction Hypothesis: $x_1, \ldots, x_{n_i} \in S_p$ (i.e., $x_1, \ldots, x_{n_i}$ have property P.)
Show that $f_i(x_1, \ldots, x_{n_i}) \in S_p$ (i.e., $f_i(x_1, \ldots, x_{n_i})$ has property P.)

5.3 A Multimodal, Interactive Electronic Book for Teaching and Learning Proofs by Induction.

As I mentioned at the beginning of this chapter, I have designed and developed an electronic book (e-book) for learning and teaching proofs by induction for computer science by taking into consideration the drawbacks of the current methodologies of teaching proofs by induction (section 5.1, p.63), as well as the documented difficulties that students have with proofs by induction (section 2.2, p.17), and the results of my study on finding the sources of computer science students' difficulties with proofs by induction (section 4.4 , p.55). I decided to present my teaching material on proofs by induction in an e-book, instead of a regular textbook, because I wanted to move away from the exclusively verbal and "passive world" of traditional textbooks towards a more active and multimodal learning environment, by taking advantage of what multimedia has to offer. For this reason, to develop my e-book, I chose to use "ToolBook Instructor,"

which is a tool for creating interactive e-learning content, and it allows easy integration of multimedia into it.

5.3.1 Educational Systems in the Classroom

In the last few decades, there has been a rapid growth of the use of technology in education, and more precisely, there has been a dramatic increase in the use of instructional software available for teaching and learning different mathematical concepts (Abraham et al., 2001; Jones et al., 1999; Reiser & Kegelmann, 1994). Such software provide an environment where the students can learn and practice, as well as have access to tutorial programs, different views on the topics at hand and different approaches to solving problems, explanations and feedback on their performance, and much more. Currently, in the field of computer science there is a number of such software available for the instructors to use in the classroom and for the students to use on their own time, including software for the areas of algorithms (algorithm visualization/animation systems) (Naps, 2005; Puntambekar et al., 2002; Boroni et al., 1998), programming (program visualization/animation systems) (Dann, Cooper, & Pausch, 2001; Carlisle, 2000; Boroni et al., 1998), automata theory (Rodger & Finley, 2006), Turing machines (Barwise & Etchemendy, 1998), and predicate logic (Barwise & Etchemendy, 1998). Some universities have already integrated such systems into their curricula (Byrne, Catrambone, & Stasko, 2000).

Even though educational software, especially interactive multimedia systems, can be very effective for learning when appropriately designed, recent surveys show a significant disconnect between instructors' belief that such systems will enhance students understanding and their willingness to integrate them into their teaching (Naps et al.,

2003). Their reluctance to use educational systems in the classroom is a result of the problems that they are currently facing with some of them. Such problems include: platform dependence, access to the software, time and complexity to download and install the software, time it takes to learn the software, difficulties to integrate the software into their course material, time it takes to create examples and problems based on the software, and difficulties to maintain and upgrade the software (Naps et al., 2006; Naps et al., 2003; Boroni et al., 1998).

To overcome some of these obstacles, my e-book is bundled on a CD, which eliminates the problem of downloading and installing it; it is a complete educational environment, which makes it easy to integrate into most instructors' material; and it contains a large amount of examples and problems, which saves time for the instructors, since they do not have to create their own.

5.3.2 The Teaching Approach

The teaching approach in my e-book follows the conceptual route of teaching proofs by induction described in the previous section (p.68). Consequently, the teaching material of the e-book strictly follows the whole set theoretical "story" of proofs by induction. The e-book presents each basic concept involved in this "story" in its own chapter, therefore, the outline of the chapters in the e-book is as follows:

Chapter 1: Structures
Chapter 2: Closed Sets
Chapter 3: Inductive Sets
Chapter 4: Inductive Definitions
Chapter 5: Proofs by Induction
(see Appendix B, section B.1, p.109)

In addition, all chapters in my e-book are organized in the same way: each chapter starts with an informal introduction of the concept at hand, followed by a number of examples, followed by a number of interactive exercises, and finally, a brief summary of the current chapter and a brief introduction to what the next chapter will be about. An example of how a typical chapter in my e-book is organized can be found in Appendix B, section B.3, p.111.

Furthermore, for each chapter in my e-book I present a large amount of examples and exercises of a great variety to demonstrate each basic concept, so that students can embody it. For the students to be able to recognize on their own when to apply and correctly apply proofs by induction, as well as to generalize them and apply them to problems that are not similar to the ones they encountered before, they need to solve many problems. Such level of understanding can only be achieved through a lot of experience and different encounters with different types of problems. According to Boroni (1998), "repetition is one of the keys to learning" (p. 146), and according to Bjork & Druckman (1994), "real competence comes only with extensive practice" (as cited in Steen 1999, p. 273). Furthermore, presenting a variety of problems to the students to choose the ones that are more interesting for them serves as a motivation and increases the interest of the students (Dunlap, 2001).

5.3.3 A Multimodal Learning Environment

It is documented in the literature that different people learn in different ways (Thomas et al., 2002; Lutzinger, 1991). In the last few decades, researchers have developed many theories on how people learn best (learning styles) (Burgess, 2005;

Thomas et al., 2002; Bohlen & Ferratt, 1993). The one of interest for my work is the Felder-Silverman learning style model.

According to Thomas et al. (2002), the Felder-Silverman learning style model classifies students as:

1) *Active or reflective learners*: active learners learn best when they are actively involved in the learning process and by trying things out, whereas reflective learners learn best by thinking things through and when the material is presented in an abstract form.
2) *Sensing or intuitive learners*: sensing learners learn best when the material is presented as facts and methods to be followed, and it is connected to the real world, whereas intuitive learners learn best when the material is presented in an abstract form, and by discovering relationships between concepts on their own.
3) *Visual or verbal learners*: visual learners learn best when the material is presented in the form of pictures, diagrams, and other external graphical representations, whereas verbal learners learn best when the material is presented linguistically, in the form of words (written or spoken).
4) *Sequential or global learners*: sequential learners learn best when the material is presented in an incremental order (in sequential, logical steps), whereas global learners learn best when the whole material is presented at once.
5) *Inductive or deductive learners*: inductive learners learn best when the material is presented from its specific to its general form, whereas deductive learners learn best when the material is presented from its general to its specific form.

Moreover, based on the sensory modality that people are using while learning, they are classified as visual, auditory, and kinesthetic learners (Thomas et al., 2002;

"Overview of Learning Styles", n.d.). As I mentioned above, visual learners learn best when the material is presented in the form of external graphical representations, whereas auditory learners learn best when the material is presented in the form of sound, and kinesthetic learners learn best when the material is presented through hands on experiences. Most people learn best when they use a combination of sensory modalities simultaneously (multimodal learning) ("Overview of Learning Styles", n.d), and they may use different combination of sensory modalities for learning different concepts (Lutzinger, 1995; Dunn, 1988).

Considering that the number of students in a classroom keeps increasing, their learning style diversity cannot be easily addressed by an instructor. In contrast, I have designed my e-book in a way that will accommodate the different learners classified by the Felder-Silverman learning style model and provide a multimodal learning environment for the students.

To present the material of my e-book, I added to the traditional text form different external graphical representations, such as images and animations. For example, when students are presented with a specific structure, they can view an animation of the operations in the structure. Also, when students are presented with an inductive definition and some specific elements which are defined by it, they can view an animation of how each given element can be constructed by following the rules of the inductive definition. (Fig. 1). Graphical representations have been proven to be especially helpful for learning complex concepts (White & Fresrickson, 1998, as cited in Puntambbekar, 2002). They are currently used as learning tools in many disciplines, from math, chemistry, and physics to computer science. In computer science we represent network protocols with

Petri nets, model resource allocation (deadlock) with directed graphs, represent the structure of a software system with graphs (flow of control, data flow, etc), and let us not forget Turing machines, DFAs, and basic Data Structures, such as Trees, Lists, and Arrays, which are also represented graphically (Henderson et al. 2001, Barwise & Etchemendy, 1998). Without the graphical representations of these concepts, it will be nearly impossible for instructors to explain them to the students on an intuitive level. According to Raeder (1985), people acquire knowledge faster by discovering relationships between images than by reading a text (as cited in Baldwin & Kuljis, 2000), and according to Grissom, McNally, & Naps (2000), graphical representations serve as a motivational factor for the students. Furthermore, learning experience with graphical representations becomes more enjoyable and fun (Naps et al., 2003). The use of graphical representations also benefits visual learners, whose needs are totally ignored by the current teaching methodologies of proofs by induction, and at the same time, helps auditory and kinesthetic learners develop their visualization skills. This is essential, since mathematics is intrinsically visual in nature (Sierpinska, 2003). According to Sierpinska (2003), in learning and doing mathematics "visualization is not a matter of didactic choice; it is a cognitive necessity" (p. 174).

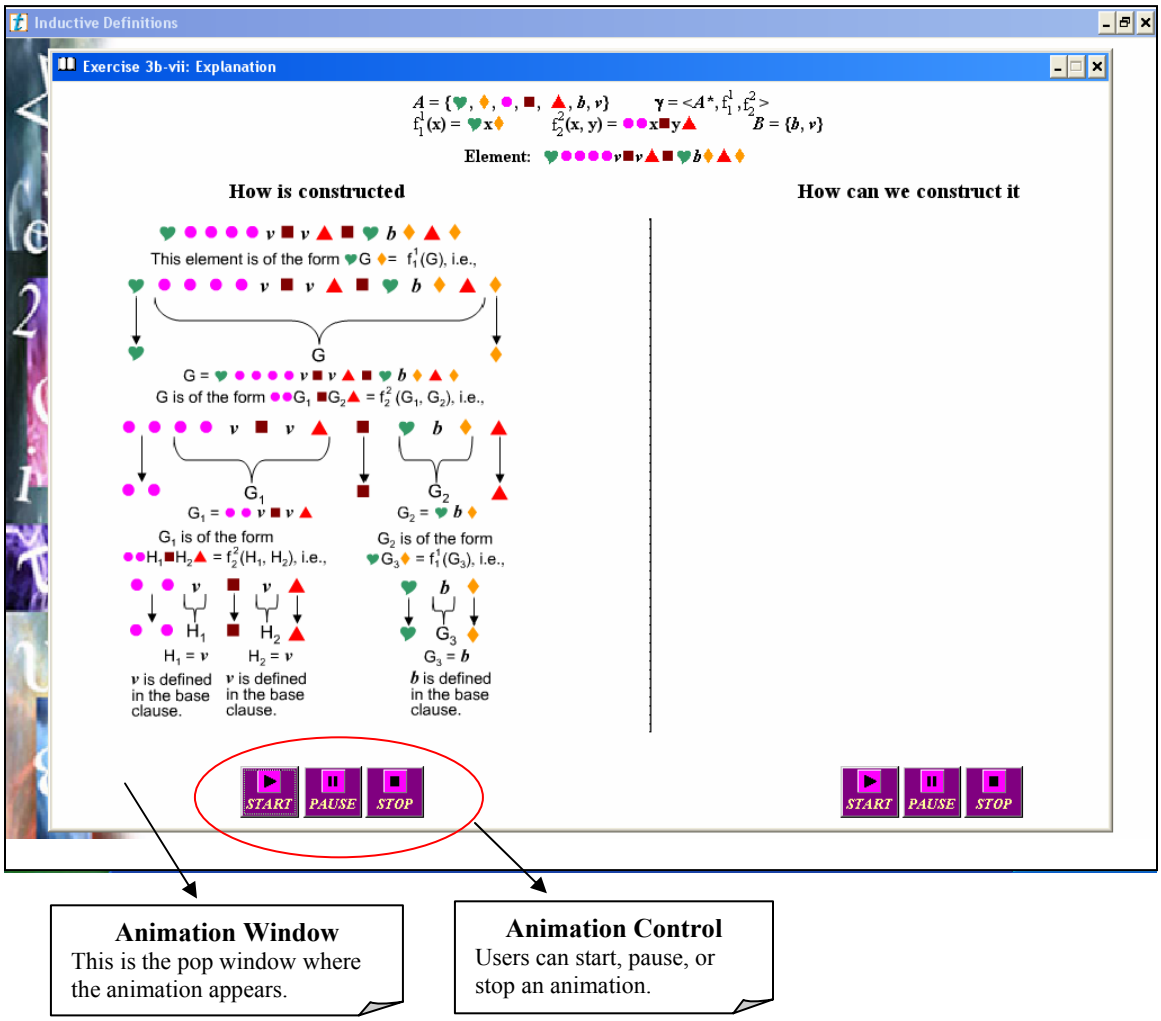


Fig. 1: This figure shows a sample animation where given an inductive definition, the animation shows whether the structure of a given element follows the definition.

Moreover, in my e-book I use different learning techniques that have been shown to be effective for students' learning. For example, for each exercise in my e-book, I provide students with a hint, as well as with feedback on their performance and detailed explanation of the solution of the exercise:

- *Hints (clues)*: For each exercise, I provide students with a hint (clue), in case they have difficulties solving it. The hint guides and leads students through questions that

will help them find solutions on their own. This guiding questions method goes all the way back to Socrates ("Socratic seminar"), who used it to help his students discover the knowledge they sought (Tsovaltzi, 2003; Dunlap, 2001). On each exercise page in my e-book, there is a hint icon available, and when students click on it, a pop up text field appears with the hint information (Fig.2).

- *Feedback*: For each exercise, I provide visual and auditory *feedback* to the students on their performance. On each exercise page in my e-book, there is a "*Submit*" button for students to submit their answer. After the students click the "*Submit*" button, the systems responds by letting them know whether their answer was correct or incorrect. The system displays the message "*Congratulations!*" for a correct answer and the message "*Incorrect Answer*" otherwise (Fig.2). Both messages are accompanied by the appropriate sound.
- *Explanation*: For each exercise, I provide students with the solution of the exercise along with a detailed explanation. According to Abraham et al. (2001), exercises are only beneficial with the appropriate feedback. Students need to know their mistakes and the reason they are wrong. Actively learning, making mistakes, and learning from them, is more effective than passively listening to a lecture (Turban, 2003). With the increasing number of students in every classroom, it is impossible for the instructors to devote time and attention to every single student, and concentrate on his/her individual mistakes. In every exercise page in my e-book, when students click the "*Submit*" button and after they are presented with the appropriate exercise feedback, a "*Solution*" button appears on the screen next to the "*Submit*" button (Fig.2). Students can click the "*Solution*" button to view a detailed explanation of the solution of the

exercise. The explanation appears on a pop up window (see Appendix B, section B.5.1, p.121).

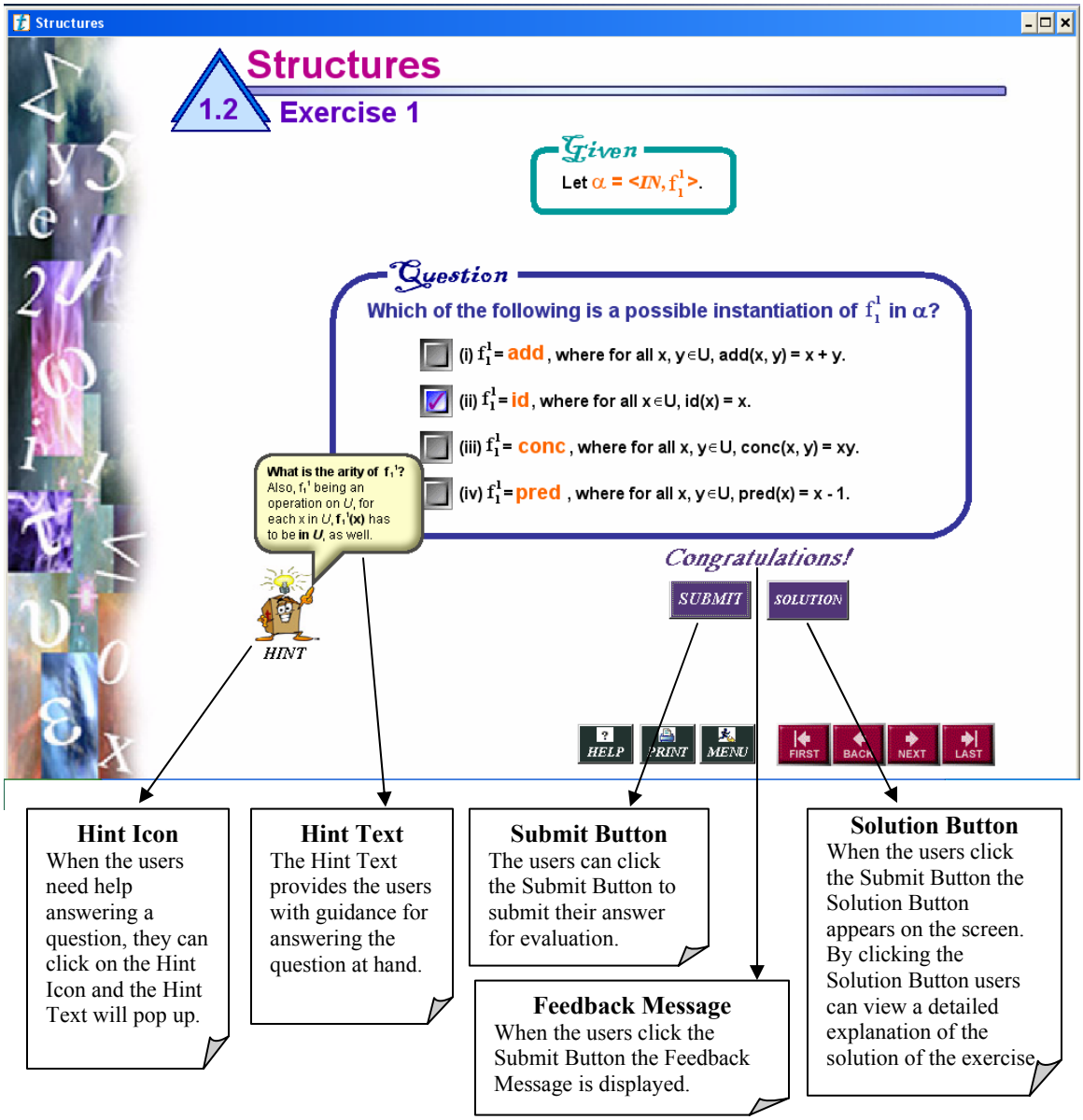


Fig. 2: This figure shows a sample exercise page form the e-book.

In addition, I have designed some of the exercises (questions) in my e-book not only to ensure students understanding of the concept at hand, but also to gently lead to concepts that students need to conquer in the chapter right after. This way, students get a

chance to think about and come to an intuitive understanding of what is coming next. For example, when students are introduced to inductive definitions, they are presented with exercises in which, given an inductive definition, they are asked questions such as, "What is the minimum length of an element of the set defined by the inductive definition at hand? Give an example for an element of that length.". Such a question aims at students' understanding of the base clause of the inductive definition, which later on will serve as the base case of a proof by induction. Also, students are presented with the following kind of exercises: given a statement claiming that all the elements of the set defined by the inductive definition at hand have a certain property, they are asked whether the statement is correct. This type of question aims at developing students' thinking on how to prove such statements and prepares them for the next chapter on proofs by induction.

Furthermore, some of the exercises in my e-book are contextualized within computer science. According to Hatano (1996) and Cox (1994), exposing students to a variety of problems in relation to different contexts of students' interest (in our case computer science), can help students broaden their knowledge and apply it in different domains (of computer science) (as cited in Henderson et al., 2001). In addition, connecting the material to computer science serves as a motivation for the students, since they can realize its different usages. According to Krone (2001), we should teach computer science concepts such as induction side-by-side with their applications in the field, so that students can experience directly their relevance and importance.

Finally, my e-book offers a flexible navigation through the chapters, which makes it easy for students to follow the learning path most appropriate for them. Throughout my e-book, students also have the option to print the content of a page and call upon a help

menu by clicking on the "*Print*" and "*Help*" buttons, respectively. Both buttons are available on every page of the e-book (see Appendix B, section B.2, p110).

- *Help*: Once in the help menu, students can get information on how to navigate through the e-book, how to respond to a question, how to submit their answer to a question, how to view the solution of an exercise, and how to view and play animations available in the e-book (see Appendix B, section B.6, p.130). In the help menu, students also have the option to view a list of all the symbols used in the e-book, along with their corresponding meaning.

Some of the functionalities of my e-book are a result of students' suggestions and comments. I gave students who were using my e-book a questionnaire aiming to check whether they liked my e-book, whether it helped them in any way, and whether there was anything else that they would like to see in a future version of the e-book that they think can improve it (the questionnaire can be found in Appendix C, p.131). I also interviewed the students who used my e-book to get additional feedback. As a result of students' responses in the questionnaire and their comments during the interview, I improved some of the functionality and design of my e-book: 1) I added the Help menu and Print option; 2) I added an animation icon in every page of the e-book that contains an animation, so that students know in advance that they can view an animation on that page; and 3) in addition to running the e-book from a CD, I added an installation feature on the CD, so that students can install the e-book on their personal computers. The questionnaire and interviews also helped me get some insights into students' thoughts about the e-book. My e-book was very well received by the students. Most of them enjoyed working with it and found it very helpful. Below are some of the students' general comments on my e-book:

- "*I thought that the e-book was a very good idea. Great Job!*"
- "*I prefer the e-book than any other textbook I have seen so far.*"
- "*The e-book made things very easy for me.*"
- "*The e-book was very easy to understand and I can use it on my own pace. I prefer the e-book than any other regular textbook because it is very interactive. I love to push buttons.*"
- "*If you read induction from a regular book you do not understand why you have to do things. But, the e-book makes the connections for you, using B-inductive sets. This was very helpful.*"
- "*This e-book is exactly what we need. It helped me a lot. The fact that is structured the same (every chapter) helped me a lot.* "
- "*The e-book is very nice and easy to understand. I like the way that the chapters are organized. It is really helpful.*"

Students also commented on how helpful the animations as well as the examples and exercises in the e-book were for them:

- "*For someone who has difficulties, the animations can be very helpful.*"
- "*The animations helped me a lot. They were very easy to follow.*"
- "*Usually professors may skip some steps of the solution/explanation in class because of time. But, with the e-book I have all the details and I can take as much time as I want.*"
- "*The fact that the e-book had a lot of examples was very helpful. If I had a question on the first one, by the time I was at the third one, my question was answered.*"
- "*Exercises where attractive, so I worked on more than what I would if it was a regular book.*"
- "*I am motivated to solve more exercises in the s/w than in a regular book.*"
- "*Having explanations of the exercises was very helpful to understand my mistakes.*"

I am not suggesting that my e-book is a comprehensive treatment of the problem at hand (completely overcoming students' difficulties with proofs by induction), but rather that it is a big step toward a solution. My intention was to provide an educational environment to help students learn and create a mental model of proofs by induction. Once students are exposed to a variety of problems on proofs by induction, as well as the basic concepts involved in and leading up to proofs by induction, they will embody the whole concept of

induction, and it will become part of their vocabulary. Then, they will be able to concentrate on correctly applying proofs by induction when appropriate, rather than struggling on how to perform them. As I will show in the next section, my e-book can help students overcome their difficulties and gain conceptual understanding of proofs by induction, which makes it a good starting point from which better and more comprehensive approaches can be build on.

5.4 Effectiveness of the E-book in the Classroom: An Exploratory Study

In addition to the questionnaire and the interview, I conducted an exploratory study to check the effectiveness of my e-book in the classroom (see also Polycarpou & Pasztor, 2008). For my study, I used the same instrument and procedures as in my study on identifying possible sources of computer science students' difficulties with proofs by induction (section 4.4, p.55). The instrument can be found in Appendix A, p.105.

5.4.1 Participants and Procedures

Participants of my study were two groups of undergraduate (mostly junior) computer science majors—although the two groups did not consist of the same number of students. Group A consisted of 44 students and group B of 26 students. Each group took a different section of a "Logic for Computer Science" course. Even though both sections were taught by the same professor, students in group A were taught induction through the traditional route, and students in group B were taught through the e-book. In both sections of the course, four class periods were spent on induction.

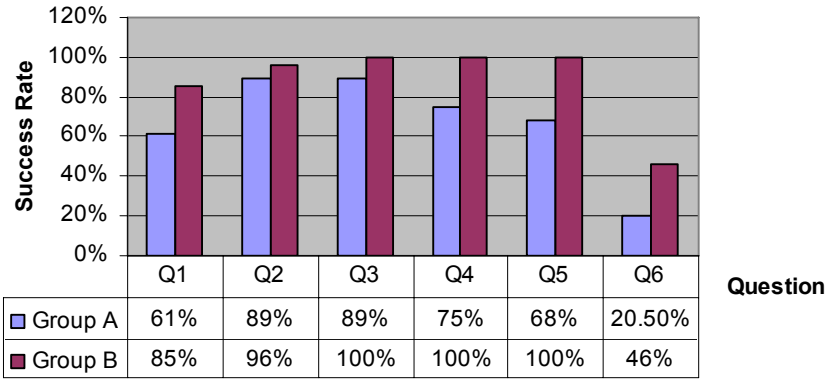
I administered the instrument to both groups after formal instruction of proofs by induction. The study was done as part of the course, and students were given participation or extra credit points for the class, contingent upon making an honest effort to answer all

questions in the instrument to the best of their knowledge. It is also worth mentioning that according to the results of their class quizzes, on average, students in the two groups performed similarly in most of the topics covered in the course.

5.4.2 Results

As with my previous study, I analyzed the data quantitatively, as well as qualitatively. My quantitative analysis was based on students' answers, and my qualitative analysis was based on students' answers and comments, as well as their thoughts and feelings documented at the end of each question.

The results of my analysis are summarized in Graphs 1, 2, and 3. The percentages in each graph are rounded to the nearest whole number. Graph 1 reports the percentages of students' correct answers in each question for group A (students who were taught through the traditional route) and B (students who were taught through the e-book), respectively. The percentage for the first question (Q1) represents the students who correctly identified all strings. The percentage for the sixth question (Q6) represents the students who successfully proved the statement by induction (they agreed with the statement, recognized that they can prove the statement by induction, and did so correctly).



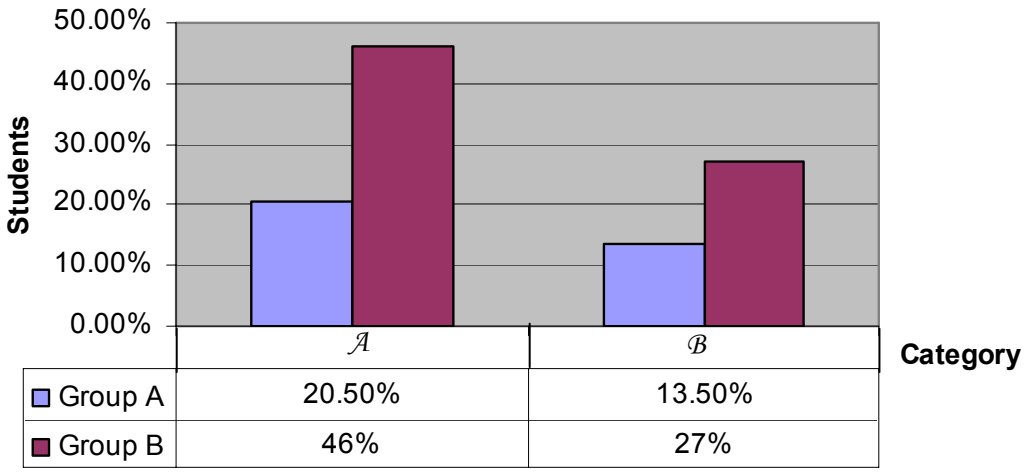
	Q1	Q2	Q3	Q4	Q5	Q6
Group A	61%	89%	89%	75%	68%	20.50%
Group B	85%	96%	100%	100%	100%	46%

Graph 1: Percentages of students' correct answers for each question.

I further analyzed students' answers for the sixth question and I divided their responses into six categories ($\mathcal{A}$ - $\mathcal{D}$, $\mathcal{N}$, and $\mathcal{I}$). Graphs 2 and 3, present the percentage of students in each category for group A and B, respectively.

Category $\mathcal{A}$ represents the students who agreed with the statement, recognized that they can prove the statement by induction, and did so correctly.

Category $\mathcal{B}$ represents the students who agreed with the statement, recognized that they can prove the statement by induction, and made valid arguments to prove the statement correct for each rule of the definition of IPO-words, but without following the usual format of a proof by induction.



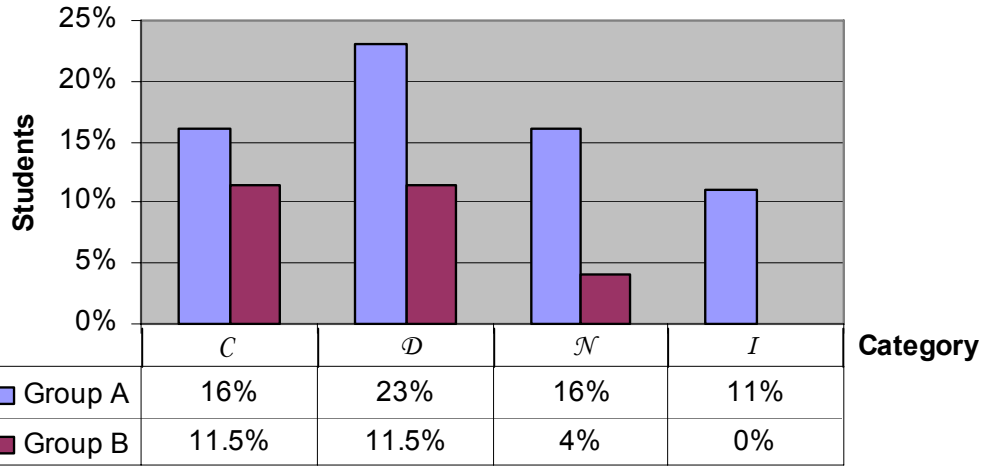
Graph 2: Percentages of students in categories $\mathcal{A}$ and $\mathcal{B}$ for question six.

Category $\mathcal{C}$ represents the students who agreed with the statement and recognized that they can prove the statement by induction, but made some mistakes in their proofs, or their proofs were incomplete.

Category $\mathcal{D}$ represents the students who agreed with the statement and recognized that they can prove the statement by induction, but used incorrect arguments to prove it.

Category $\mathcal{N}$ represents the students who agreed with the statement, but did not recognize that they can prove the statement by induction or they did not prove it at all.

Category I represents the students who disagreed with the statement.



	C	$\mathcal{D}$	$\mathcal{N}$	I
Group A	16%	23%	16%	11%
Group B	11.5%	11.5%	4%	0%

Graph 3: Percentages of students in categories C through I for question six.

As we can see in Graph 1, overall, group B had a higher success rate in all of the questions compared to group A. This suggests that the e-book enhanced students' understanding of inductive definitions and their performance on proofs by induction.

Based on the results of my study on identifying possible sources of computer science students' difficulties with proofs by induction (section 4.4, p.55), I expected students in group A to be influenced by the context in which the problem was presented and to apply proofs by induction without a conceptual understanding of the process involved (mechanically apply proofs by induction). I did not expect the same from students in group B. Indeed, the results of my qualitative analysis suggest that students in group A were influenced by the context in which the problem was presented, whereas

students in group B were not. A number of students from group A commented that the maximum length of an IPO-word depends on the computer's capacity and performance, but there was no similar comment by any student in group B. My second expectation was also met. Some students in group A applied proofs by induction mechanically, whereas there was no indication that any student in group B did so. According to Graph 1 (Q5), 32% of the students in group A incorrectly answered question five (by constructing a string of length eight which they thought was an IPO-word), and only half of them disagreed with the statement in question six that all IPO-words have odd length. This implies that 16% of the students in group A contradicted themselves in those two questions. Additionally, almost half of the students who contradicted themselves in the two questions, correctly proved the statement in question six using proof by induction. Based on these students' performance in previous questions and their reported thoughts and feelings, I concluded that they did not have a clear understanding of the inductive definition of an IPO-word, which suggests that they applied proof by induction "mechanically." In contrast, there was no such contradiction among the results of students in group B, since all of them answered question five correctly and all of them agreed with the statement in question six. This suggests that my e-book helped students overcome their biases and understand the substance of proofs by induction.

Finally, according to Graph 2, the percentage of students in both categories $\mathcal{A}$ and $\mathcal{B}$ together was significantly higher for group B (73%) than for group A (34%). This implies that students who were taught through my e-book performed significantly better on the proof by induction on the sixth question than the students who were taught through the traditional route. I further analyzed students' answers for question six qualitatively

and I found that students in categories $\mathcal{A}$ and $\mathcal{B}$ had conceptual understanding of both the process involved in a proof by induction and the definition of IPO-words. The only difference between the students in these two categories was that students in category $\mathcal{B}$ had some difficulty articulating their thoughts, compared to students in category $\mathcal{A}$. This suggests that my e-book not only improved students' performance, but it also helped students gain conceptual understanding of proofs by induction, which was one of its main goals.

Even though the two sample populations of my study did not consist of the same number of students (44 students in group A and 26 students in group B) and this may weaken my study, the results are very promising and inducing to further research. Moreover, since my study is not focused on teaching but rather on learning, I chose to administer my instrument to students taking a class with one specific professor who has been teaching this course for decades. I believe that I can strengthen my findings by administering the instrument to students taking classes taught by different professors.

CHAPTER 6

CONCLUSION AND FUTURE RESEARCH

Induction is a core concept in the field of computer science, and yet, students have difficulties understanding it. Even though this has been well documented in the literature for decades and different solutions have been proposed, little improvement has been made. The purpose of my dissertation was to find a new approach to teach proofs by induction to computer science students in way that facilitates their conceptual understanding of proofs by induction and help them overcome their difficulties. For developing such an approach, it was important to not only identify what students' difficulties with proofs by induction are, but also to identify possible sources of such difficulties.

As part of my dissertation, I have conducted a study to identify possible sources of computer science students' difficulties with proofs by induction, which I have presented in Chapter 3. My study was contextualized within the undergraduate computer science curriculum. Its results suggest that understanding of the set theoretical concepts presupposed in proofs by induction (i.e., structures, closed sets, inductive sets, and inductive definitions) is a major source of computer science students' difficulties with proofs by induction. More precisely, the results of my study show that there is a close correlation between students' understanding of the set theoretical concepts presupposed in proofs by induction and students' understanding and performance of proofs by induction. In addition, the results of my study suggest that students who have conceptual

difficulties with proofs by induction are those who have difficulties understanding such concepts.

Taking into consideration the results of my study, as well as the fact that the current methodologies of teaching proofs by induction pay inadequate attention to the set theoretical concepts presupposed in proofs by induction, I proposed that proofs by induction are taught through what I call the "conceptual route" of teaching proofs by induction. In contrast to the traditional route, which presents proofs by induction as a step by step "recipe" to be followed without explaining the rationale behind each step, the conceptual route is streamlined towards teaching the set theoretical concepts presupposed in proofs by induction. Specifically, it introduces students to the concepts of structures, closed sets, inductive sets, and inductive definitions, then it connects these concepts to the Induction Principle that justifies proofs by induction, and finally, connects the Induction Principle to proofs by induction. I have presented the whole set theoretical "story" behind the conceptual route in section 5.2.

In addition, as part of my dissertation, I have designed and developed a standalone electronic book (e-book), which I have presented in section 5.3. My e-book provides an interactive and multimodal educational environment for learning/teaching proofs by induction for computer science. Since my aim with this e-book was to help computer science students gain conceptual understanding of proofs by induction, I have designed its teaching material based on the conceptual route of teaching proofs by induction. Moreover, my e-book contains numerous images, animations, examples, and interactive exercises on each individual concept, so that students can truly embody it. It can also accommodate many different learning styles, something that is hard to address in

the classroom. Additionally, my e-book eliminates many of the problems that generally make educators reluctant to use educational software in their classrooms. It is bundled on a CD, which eliminates the problem of downloading and installing it; it is a complete educational environment, which makes it easy to integrate into most instructors' material; and it contains a large amount of examples and problems, which saves time for the instructors, since they do not have to create their own.

Last but not least, as part of my dissertation, I have conducted an exploratory study on the effectiveness of my e-book in the classroom, which I have presented in section 5.4. The e-book was very well received by the students, but more importantly, the results of the exploratory study suggest that students who were taught proofs by induction through my e-book performed better on inductive definitions and proofs by induction than students who were taught through the traditional route. Furthermore, the results of my qualitative analysis suggest that more students of those who were taught through the e-book gained conceptual understanding of proofs by induction than of those who were taught through the traditional route.

The sample population of my exploratory study was not consistent, since I had 44 students who were taught proofs by induction through the traditional route and 26 students who were taught through my e-book. At the same time, all the participants of my study were taught by the same professor. In the future, I am planning to conduct another study with a more consistent and larger sample population and with different professors teaching proofs by induction through my e-book, so that I can validate and strengthen the results of my exploratory study. I am also working towards improving the accessibility of

my e-book, so that more instructors and students can have access to it. I am planning to have an on-line version of the e-book.

Even though the results of my exploratory study suggest that the e-book helped computer science students gain conceptual understanding of and improved students' performance with proofs by induction, I am not claiming that my e-book is a comprehensive "treatment" of the problem. Currently, there is no unified framework used for proofs by induction throughout the computer science curriculum that would help students realize that all the applications of proofs by induction are instantiations of the same schema (Polycarpou, 2008). Every time computer science students come across proofs by induction in different courses of their curricula, they think that they have to learn something new. This does not allow them to focus on the substance of the concepts at hand and instead, they are trying to memorize the different ways that proofs by induction can be applied in computer science.

"Induction itself presents specific cognitive obstacles and students will continue to be unsuccessful with induction as long as teaching methodology continues to ignore these difficulties." (Dubinsky, 1989, p.285). Even though there is a pressure on university educators to skip some of the "less useful" theory to focus on the latest trends (Yurcik and Doss, 2001), I hope that they will take into consideration the relevant research and will see the "big picture" and the importance of enhancing the undergraduate computer science curriculum by updating the teaching material on proofs by induction. Hopefully, they will not consider proofs by induction as one of the "less useful" topics.

I agree with Tucker et al. (1996) that there is no unique curriculum that applies to all universities/computer science schools, since different universities/computer science

schools have different educational priorities. However, no matter what these priorities are, teaching proofs by induction in a way that will facilitate computer science students' conceptual understanding is vital, since it will provide students with an essential tool for effective learning in many courses of their curricula.

LIST OF REFERENCES

Abraham, D., Crawford, L., Lesta, L., Merceron, A. & Yacef, K. (2001, October). The Logic Tutor: A multimedia Presentation. Interactive Multimedia Electronic Journal of Computer-Enhanced Learning, 3, (2). Available World Wide Web: http://imej.wfu.edu/articles/2001/2/03/index.asp.

Acerbi, F. (2000). Plato: Parmenides 149a7-c3. A Proof by Complete Induction?. *Archive for history of exact sciences, 55*, 57-76.

Alacaci, C. & Pasztor, A. (2002). Effects of flawed state assessment preparation materials on students' mathematical reasoning: A study. The Journal of Mathematical Behavior, 21, 225-253.

Appel, A. (2002). Modern Compiler Implementation In Java (2nd ed.). New York, NY: Cambridge University Press.

Baker, D. (1995). Characterizing students' difficulty with proof by mathematical induction. Doctoral Dissertation, School of Education, Indiana University, Bloomington.

Baldwin, P. L. & Kuljis, J. (2000). Visualization Techniques for Learning and Teaching Programming. In the Proceedings of the 22nd International Conference in Information Technology Interfaces (ITI 2000), 83-90.

Ball L. D., Hoyles, C., Jahnke, N. H., & Movshovitz-Hadar, N. (2002). The teaching of Proof. In the Proceedings of the ICM, Beijing, vol. 3, 907-920.

Barker, W., Bressoud, D., Epp, S., Ganter, S., Haver, B., & Pollatsek, H. (2004). Undergraduate Programs and Courses in the Mathematical Science: CUPM Curriculum Guide 2004. The Mathematical Association of America.

Barwise, J., & Etchemendy, J. (1998). Computers, visualization, and the nature of reasoning. In Ward, B., Ward, T, Moor, H. J. (Eds), The Digital Phoenix: How Computers are Changing Philosophy (pp. 93-116). Oxford, UK and Malden, MA: Blackwell Publishing, Inc.

Ben-Ari, M. (2001). Mathematical Logic for Computer Science (2nd ed.). London, U.K.: Springer-Verlag.

Best, E. (1996). Semantics of Sequential and Parallel Programs. New York, NY: Prentice Hall.

Blanton, L. M. & Stylianou, A. D. (2003). The nature of Scaffolding in undergraduate students' transition to mathematical Proof. In the Proceedings of the 27th conference of the International group for the Psychology of Mathematics Education held jointly with the 25th Conference of PME, 113-120.

Bohlen, A. G. & Ferratt, W. T. (1993). The Effect of Learning Style and Method of Instruction on the Achievement, Efficiency and Satisfaction of End-users Learning Computer Software. In the Proceedings of the 1993 conference on Computer personnel research, 273-283.

Boroni, M. C., Goosey, W. F., Grinder, T. M., & Ross, J. R. (1998). A Paradigm Shift! The Internet, the Web, Browsers, Java, and the Future of Computer Science Education. ACM SIGCSE Bulletin , Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education SIGCSE '98, 30 (1), 145-152.

Bruce, B. K., Drysdale, L. S. R., & Kelemen, C. (September, 2003). Why Math? Communications of the ACM, 46, (9), 40-44.

Brumfiel, C. (1974). A note on mathematical Induction. Mathematics Teacher, 67 (7), 616-618.

Bundy, A., Harmelen, F., Hesketh, J., & Smaill, A. (1991). Experiments with proof plans for induction. Journal of Automated reasoning, 7, 303-324.

Burgess, A. G. & Hanshaw, C. (2005). Application of Learning Styles and Approaches in Computing Sciences Classes. Consortium for Computing Sciences in Colleges (CCSC), 60-68.

Burris, S. (1998). Logic for Mathematics and Computer Science. Upper Saddle River, NJ: Prentice Hall.

Byrne, D. M., Catrambone, R., & Stasko, T. J. (2000). Evaluating Animations as Student Aids in Learning Computer Algorithms. Computers & Education, 33, 253-278.

Carlisie, E. G. (2000). EROSI – Visualising Recursion and Discovering New Errors. In the Proceedings of the ACM SIGCSE 2000 Conference, 305 – 309.

Cormen, T., Leiserson, C., Rivest, R., & Stein, C. Introduction to algorithms (2^{nd} ed.) Cambridge, Massachusetts London, England: The MIT press.

Cuoco, A. A. & Goldenberg, E. P. (1992). Mathematical Induction in a Visual Context. Interactive Learning Environments, 2 (3/4), 181-204.

Dann, W., Cooper, S., & Pausch, R. (2001). Using Visualization To Teach Novices Recursion. ACM SIGCSE Bulletin, 33 (3), 109-112.

Dubinsky, E. (1986). Teaching Mathematical Induction I. Journal of Mathematical Behavior, 5, 305-317.

Dubinsky, E. (1989). Teaching Mathematical Induction II. Journal of Mathematical Behavior, 8, 285-304.

Dubinsky, E. & Lewin, P. (1986). Reflective abstraction in mathematics education: The genetic decomposition of induction and compactness. Journal of Mathematical Behavior, 5, 55-92.

Dunlap, J. (2001, November). Mathematical Thinking. Available: http://www.mste.uiuc.edu/courses/ci431sp02/students/jdunlap/WhitePaperII.doc

Dunn, R. (1988). Capitalizing on Students' Perceptual Strengths to Ensure Literacy While Engaging in Conventional Lecture/Discussion. Reading Psychology: An International Quarterly, 9, 431-453.

Enderton, H. (2001). A Mathematical Introduction to Logic. Burlington, MA: Harcourt Academic press.

Engel, G. & Roberts, E. (Eds.) (2001). Computing Curricula 2001–Computer Science. IEEE, ACM. Final Report.

Epp, S. S. (1996). A Cognitive Approach to Teaching Logic and Proof. In DIMACS Symposium on Teaching Logic and Reasoning in an Illogical World. Piscataway, New Jersey: Rutgers University.

Epp, S. S. (2003). The role of Logic in Teaching Proof. American Mathematical Monthly 110 (10), 886-899.

Ernest, P. (1984). Mathematical Induction: A pedagogical Discussion. Educational Studies in Mathematics, 15, 173-189.

Fressola, R. A. & Krone, J. (2003, October). Integer Construction by Induction. Available: www.denison.edu/mathsci/mcurcsm2003/papers/Induction.pdf

Gallier, H., J. (2003). Logic For Computer Science. Foundations of Automatic Theorem Proving. John Wiley & Sons Inc.

Gibbs, G. (1994). Improving Student Learning. Oxford, UK: The Oxford Center for Staff Development.

Grissom, S., McNally, F. M., & Naps, T. (2003). Algorithm Visualization in CS Education: Comparing Levels of Students Engagement. In the Proceedings of the 2003 ACM symposium on Software visualization, 87-94.

Henderson, B. P., Hamer, J., Baldwin, D., Hitchner, L., Dasigi, V., Lloyd, W., Dupras, M., Marion, B., Fritz, J., Riedesel, C., Ginant, D., Walker, H., & Goelman, D. (2001). Striving for Mathematical Thinking. ACM SIGCSE Bulletin, 33 (4), 114-124.

Holland-Minkley, A. (2002). Planning Proof Content for Communicating Induction. In the Proceedings of Second International Natural Language Generation Conference, 167-172.

Hopcroft, J., Motwani, R., & Ullman, J. (2001). Introduction to automata theory (3rd ed.) Boston, MA: Addison-Wesley.

Hoyles, C. & Küchemann, D. (2002). Students' Understanding of Logical Implication. Educational Studies in Mathematics, 51, 193-223.

Huth, M. & Ryan, M. (2000). Logic In Computer Science: modeling and reasoning about systems. Cambridge, UK: University Press.

Jones, A., Scanlon, E., Tosunoglu, C., Morris, E., Ross, S., Butcher, P., & Greenberg, J. (1999). Contextx for Evaluating Educational Software. Interacting with Computers, 11, 499-516.

Kaplan, M. K., Burge, L., Garuba, M., & Kaplan, J.J. (2004). Mathematical Induction: The Basis Step of Verification and Validation in a Modeling and Simulation Course. In the Proceedings of the 2004 American Society for Engineering Education Annual Conference & Exposition, session number: 1465.

Krone, J. & Feil T. (2001). Incorporating Mathematics Into the First Year CS Program: A new Approach to CS2. Consortium for Computing in Small Colleges, 17 (1), 44-51.

Linz, P. (2001). An Introduction to Formal Languages and Automata. Sudbury, MA: Jones and Bartlett Publishers, Inc.

Lowenthal, F. & Eisenberg, T. (1992). Mathematical Induction in School: An illusion of Rigor? School Science and Mathematics, 92, 233-238.

Lutzinger, F. D. (1991, September). Smart Stuff: Learning Styles – A never ending story. Anchor Point Magazine, 19-23.

Lutzinger, F. D. (1995, August). What is your not learning style?. Anchor Point Magazine, 3-6.

Lynch, N. (1996). Distributed Algorithms. San Francisco, CA: Morgan Kaufmann Publishers, Inc.

Manber, U. (1989). Introduction to Algorithms. A creative Approach. USA: Addison-Wesley Publishing Company, Inc.

Manna Z., Ness, S., & Vuillemin, J. (1973). Inductive Methods for Proving Properties of Programs. Communications of the ACM, 16 (8), 491-502.

Mendelson, E. (1997). Introduction to Mathematical Logic. Great Britain: Chapman & Hall.

Movshovitz-Hadar, N. (1993). Mathematical Induction: A Focus on the Conceptual Framework. School Science and Mathematics, 93, 408-417.

Naps, T., Rodger, S., Rössling, G., & Ross, R. (2006). Animation and Visualization in the Curriculum: Opportunities, Challenges, and Success. ACM SIGCSE Bulletin , Proceedings of the 37th SIGCSE technical symposium on Computer science education SIGCSE '06, 38 (1), 328-329.

Naps, T. (2005). JHAVE – Addressing the Need to Support Algorithm Visualization with Tools for Active Engagement. IEEE Computer Graphics and Applications.

Naps, T., Röβling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., & Velázquez-Iturbide, J. A. (2003). Exploring the Role of Visualization and Engagement in Computer Science Education. ACM SIGCSE Bulletin, 35 (2), 131-152.

Naps, T., Röβling, G., Anderson, J., Stephen, C., Dann, W., Fleischer, R., Koldehofe, B., Korhonen, A., Kuittnen, M., Leska, C., Malmi, L., McNally, Rantakokko, J., & Ross, J. R. (2003). Evaluating the Educational Impact of Visualization. Report of the Working Group from ITiCSE 2003 on "Evaluating the Educational Impact of Visualization."

Niederhauser, S. Dale & Stoddart, T. (2000). Teachers' instructional perspective and use of educational software. Teaching and Teacher Education, 17, 15-31.

Overview of Learning Styles. Retrieved November 27, 2006 from http://www.learning-styles-online.com/overview/

Page, L. R. (2003). Software is Discrete Mathematics. In the Proceedings of the eighth ACM SIGPLAN international conference on Functional programming, 79-86.

Pierce, C. B. (2002). Types and Programming Languages. Cambridge, Massachusetts London, England: The MIT press.

Polycarpou, I. & Pasztor, A. (2008). An Interactive and Multimodal Software for Teaching Induction for Computer Science. To appear in the Proceedings of the 2008 International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS'08).

Polycarpou, I. (2008). Induction as a Tool for Conceptual Coherence in Computer Science. In the Proceedings of the 46th ACM Southeast conference.

Polycarpou, I, Pasztor, A., & Adjouadi, M. (2008). A Conceptual Approach to Teaching Induction for Computer Science. In the Proceedings of the 39th ACM Technical Symposium on Computer Science Education (SIGCSE'08), 9-13.

Polycarpou, I., Pasztor, A., & Alacaci, G. (2006). Sources of Students' Difficulties with Proofs by Induction: A Study. In the Proceedings of the Third International Conference in Teaching Mathematics (ICTM3) in Turkey, paper 446.

Polycarpou, I. (2006). Computer Science students' difficulties with proofs by induction: An exploratory study. In the Proceedings of the 44th ACM Southeast conference, 601-606.

Puntambekar, S., Stylianou, A, Suthers, D., Hundhausen, C., & Hübscher-Younger, T. (2002). External Representations for Collaborative Learning and Assessment. In the Proceedings of the CSCL conference.

Raffali, C. & David, R. (2002). Computer Assisted Teaching in Mathematics. In the Proceedings of the Workshop on 35 years of Automath (Edingurgh, France).

Reeves, S. & Clarke, M. (reprinted in 2003). Logic For Computer Science. England: Wokingham; Reading, MA: Addison-Wesley.

Reiser, A. R. & Kegelmann, W. H. (1994). Evaluating Instructional Software: A review and Critique of Current Methods. Educational Technology Research and Development (ETR&D), 42 (3), 63-69.

Rodger, H. S. & Finley, W. T. (2006). JFLAP: An interactive Formal Languages and Automata Package. Sudbury, MA: Jones and Bartlett Publishers.

Ross, K. (1998). Doing and Proving: The Place of Algorithms and Proofs in School Mathematics. The American Mathematical Monthly, 105 (3), 252-255.

Schach, A. (1958). Two Forms of Mathematical Induction. Mathematics Magazine, 32 (2), 83-85.

Schöning, U. (1989). Logic for Computer Science. Boston, MA: Birkhäuser.

Sheard, M. (1998). Induction the Hard Way. The American Mathematical Monthly, 150 (4), 348-353.

Shonk (2003). I proved all odds are prime – With Inductive Reasoning. Selling Waves. Available: http://www.sellingwaves.com/archives/2003/11/

Sierpinska, A. (2003). Visualization is in the mind of the beholder. New Zealand Journal of Mathematics, 32 (1), 173-194.

Sipser, M. (1997). Introduction on the Theory of Computation. Boston, MA: PWS Publishing Company.

Steen A. L. (1999). Twenty Questions about Mathematical Reasoning. In L. Stiff (ed), Developing Mathematical Reasoning in Grades K-12. (pp. 270 - 285). Reston, VA: National Council of Teachers of Mathematics.

Thagard, P. (1998). Mind. Introduction to Cognitive Science. Cambridge, MA: MIT Press.

Thomas, L., Ratcliffe, M., Woodbury, J., & Jarman, E. (2002). Learning Styles and Performance in the Introductory Programming Sequence. ACM SIGCSE Bulletin, Proceedings of the 33rd SIGCSE technical symposium on Computer science education SIGCSE '02, 34 (1), 33-37.

Thompson, D. (1996). Learning and Teaching Indirect Proof. The Mathematics Teacher, 89, 474-481.

Tsovaltzi, D. & Fiedler, A.(2003). An Approach to Facilitating Reflection in a Mathematics Tutoring System. In Vincent Aleven, Ulrich Hoppe, Judy Kay, Riichiro Mizoguchi, Helen Pain, Felisa Verdejo, and Kalina Yacef (eds), AIED2003 - Supplementary Proceedings of the 11th International Conference on Artificial Intelligence in Education, V (pp. 278-287).

Tucker, B.A., et al. (1996). Strategic Directions in Computer Science Education. ACM Computing Surveys, 28 (4), 836-845.

Turban, R. (2003). Approaches to Implementing and Teaching Human Computer Interaction. In the Proceedings of the International Conference on Information Technology: Computers and Communications (ITCC'03), 81-84.

Weiss, M. (2006). Data Structures & Problem Solving Using Java.Boston, MA: Pearson Education, Inc.

Wu, C., Dale, B. N., & Bethel, J. L. (1998). Conceptual Models and Cognitive Learning Styles in Teaching Recursion. ACM SIGCSE Bulletin, Proceedings of the twenty-ninth SIGCSE technical symposium on Computer Science Education SIGCSE '98, 30 (1), 292-296.

Wu Yu, J. (2000). Reasoning Difficulty related to validity of arguments and truth of assertions in mathematics. Doctoral Dissertation, School of Education, University of Pittsburgh, Pittsburgh, PA.

Wiznia, M. (2003, November). A Brief History of Proof Theory. Available: http://info.med.yale.edu/therarad/summers/abstract.htm

Yurcik, W. & Doss, D. (2001). Different Approaches in the Teaching of Information Systems Security. In the Proceedings of the Information Systems Education Conference (ISECON), paper 04a.

APPENDIX A

STUDY INSTRUMENT

Class: COT 3420, Logic for Computer Science
This in **NOT** for a grade, however, you will get 5 participation points for the course.

Your name:

IPO-WORDS

As most of you know by now, most programming languages have their own requirements for defining variable names. The reason for this is for the compiler of the programming language to be able to identify a specific string of characters as a variable name. There is a new programming language out on the market, called IPO. In IPO, variable names are called IPO-words, and they are constructed from the characters of the set {a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, **“**, **”**, _}, using the following *rules*:

1. Any member of the set {a, …, z} is an IPO-word.
2. If *W* is an IPO-word, then **“***W***”** is also an IPO-word (in other words, an IPO-word in quotes is also an IPO-word).
3. If *W*1 and *W*2 are IPO-words, then *W*1_*W*2 is also an IPO-word (in other words, two IPO-words concatenated by underscore form a new IPO-word).
4. No sequence of characters is an IPO-word unless it *has* to be one by 1., 2., or 3. above.

The length of an IPO-word refers to the number of characters in an IPO-word, where a character is any letter **a** through **z**, an opening or closing quotation mark, or an underscore.

Questions:

1. Identify which of the following strings are and which are not IPO-words and explain why. Mark the appropriate box.

   Apple □ No □ Yes _____

   _____

   A_p_p_l_e □ No □ Yes _____

   _____

   a_p“_p_l”_e □ No □ Yes _____

   _____

   “a_p”_“p_l_e” □ No □ Yes _____

a_“p_p”_l_e □ No □ Yes _____

a_p“”_p_l_e □ No □ Yes _____

a_p“_”_p_l_e □ No □ Yes _____

a_p_p_L_e □ No □ Yes _____

“r_e_d”_“a_p_p_l_e” □ No □ Yes _____

green_apple □ No □ Yes _____

Describe in a few sentences your experience (thoughts-feelings) with this question:

2. What is the minimum length of an IPO-word? Give an example of an IPO-word of that length.

_____

Describe in a few sentences your experience (thoughts-feelings) with this question:

3. Can you also give the maximum length of an IPO-word? Explain.

_____

Describe in a few sentences your experience (thoughts-feelings) with this question:

4. Create an IPO-word of length **greater** than 6.

_____

Describe in a few sentences your experience (thoughts-feelings) with this question:

5. Create an IPO-word of length **equal** to 8.

_____

Describe in a few sentences your experience (thoughts-feelings) with this question:

6. Statement:
   For every IPO-word *W*, the following property P(*W*) holds: The length of *W* is odd (that is, *W* has an odd number of characters, where a character is any member of the set {a, …, z, **“**, **”**, _}).
   Do you think the above statement is correct? Whatever your answer is, try to prove it (Hint: use the rules for constructing an IPO-word).

Describe in a few sentences your experience (thoughts-feelings) with this question:

SOLUTIONS TO THE QUESTIONS IN THE INSTRUMENT

Question 1

Apple	□ No
A_p_p_l_e	□ No
a_p“_p_l”_e	□ No
“a_p”_“p_l_e”	□ Yes
a_“p_p”_l_e	□ Yes
a_p“”_p_l_e	□ No
a_p“_”_p_l_e	□ No
a_p_p_L_e	□ No
“r_e_d”_“a_p_p_l_e”	□ Yes
green_apple	□ No

Question 2
The minimum length of an IPO-word is one: any character between lowercase *a* through lowercase *z*. This is given by the Base Clause of the inductive definition of IPO-words.

Question 3
There is no limit on the length of an IPO-word. The set of all IPO-words is infinite.

Question 4
You can follow the rules of the inductive definition of IPO-words to construct an IPO-word of length greater than six. You can start with any element defined in the Base Clause ({a, …, z}) and apply the operations defined in the Inductive Clause.
For example, you can choose a, and then apply the first rule of the induction clause repeatedly: e.g., “ “ “ a ” ” ” (length is 7). You can also choose more elements and use the second rule: e.g., a_b_c_d_e (length is 9). Finally, you can have a combination of the rules: e.g., “a”_ “b”_c (length is 9).

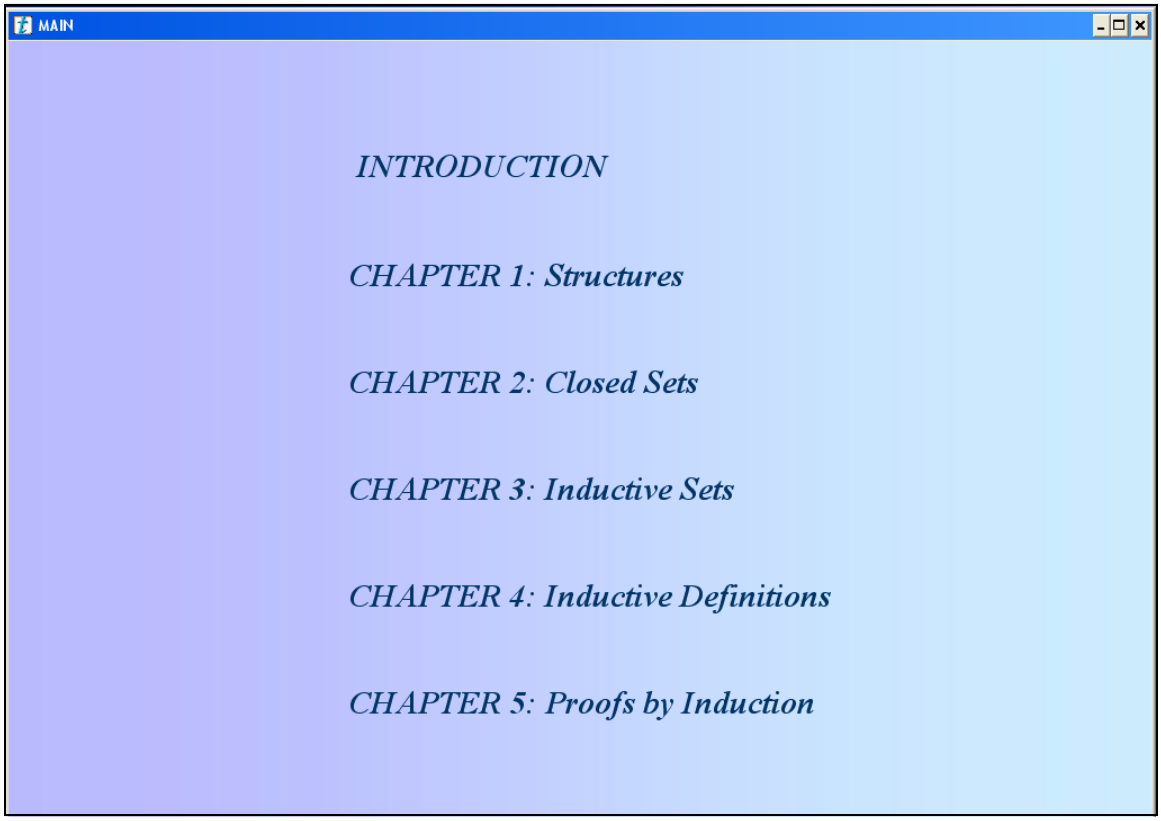
Question 5
This is not possible, because IPO-words can only have odd length.

Question 6
Let F be an arbitrary IPO-word.
1) Base Case: Assume that F is any member of the set {a, …, z}.
Then the length of F is one, and therefore odd.
So, F has property P.
2) Induction Step:
2a) Assume that F = “G”, where G is an IPO-word.
Induction Hypothesis: G has odd length, i.e., G has property P.
The length of F is equal to the length of G plus two (two additional element, namely “, and ”). By the induction hypothesis we know that G has odd length. Adding two to an odd number results to another odd number.
So, F has an odd length and therefore, F has property P.
2b) Assume that F = G_H, where G and H are IPO-words.
Induction Hypothesis: G and H both have odd length, i.e., G and H have property P.
The length of F is equal to the length of G plus the length of H plus one (one additional element, namely _). By the induction hypothesis we know that G and H both have odd length, therefore the length of F is the result of the addition of two odd numbers (even number) and one, which results to an odd number (even number plus one results to another odd number).
So, F has an odd length, and therefore, F has property P.
Since F was an arbitrary IPO-word and we covered all the possibilities for the structure of F, we proved the proposition.

APPENDIX B

THE E-BOOK

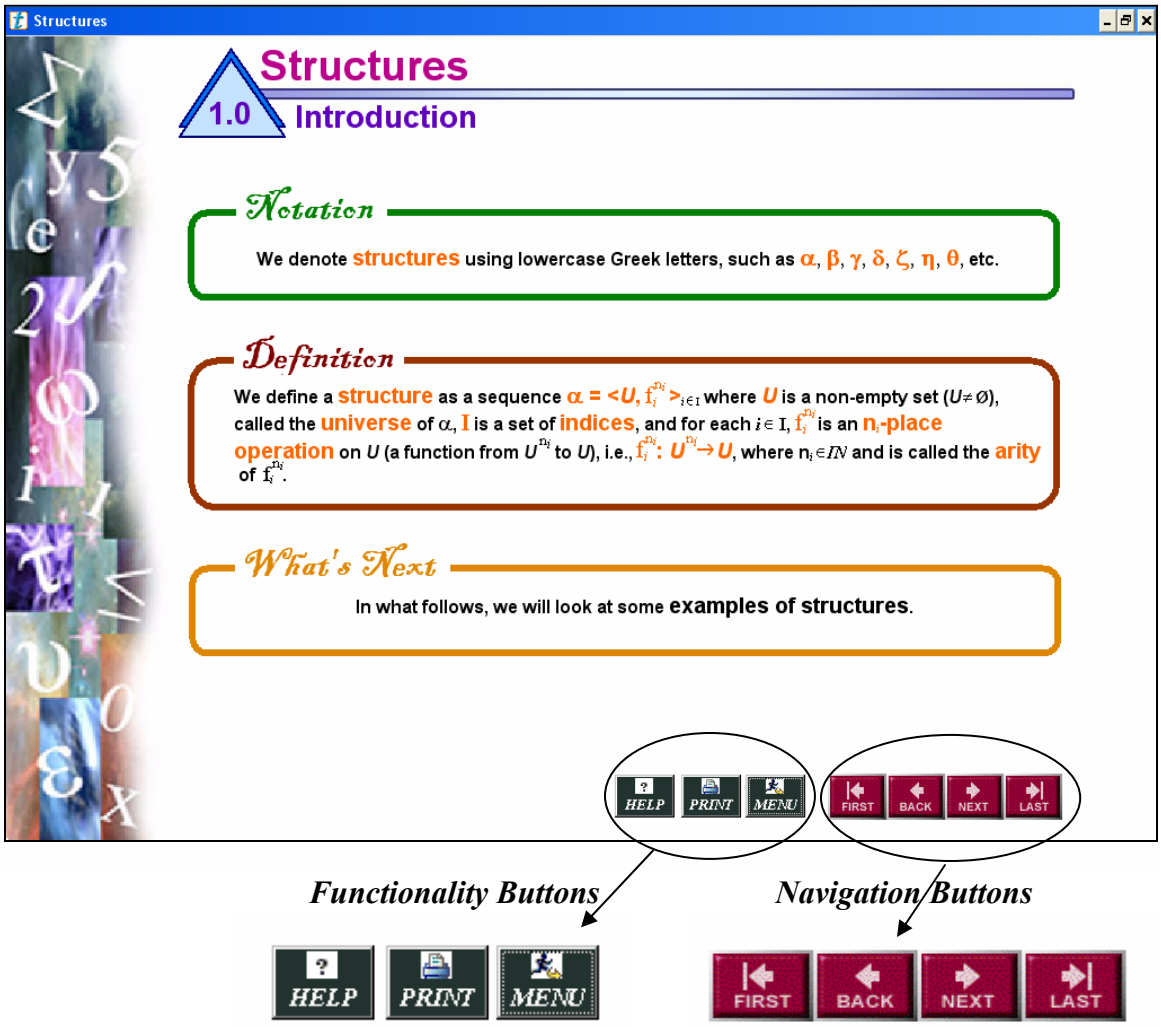
B.1 Main Menu



Main Menu – Outline of the chapters

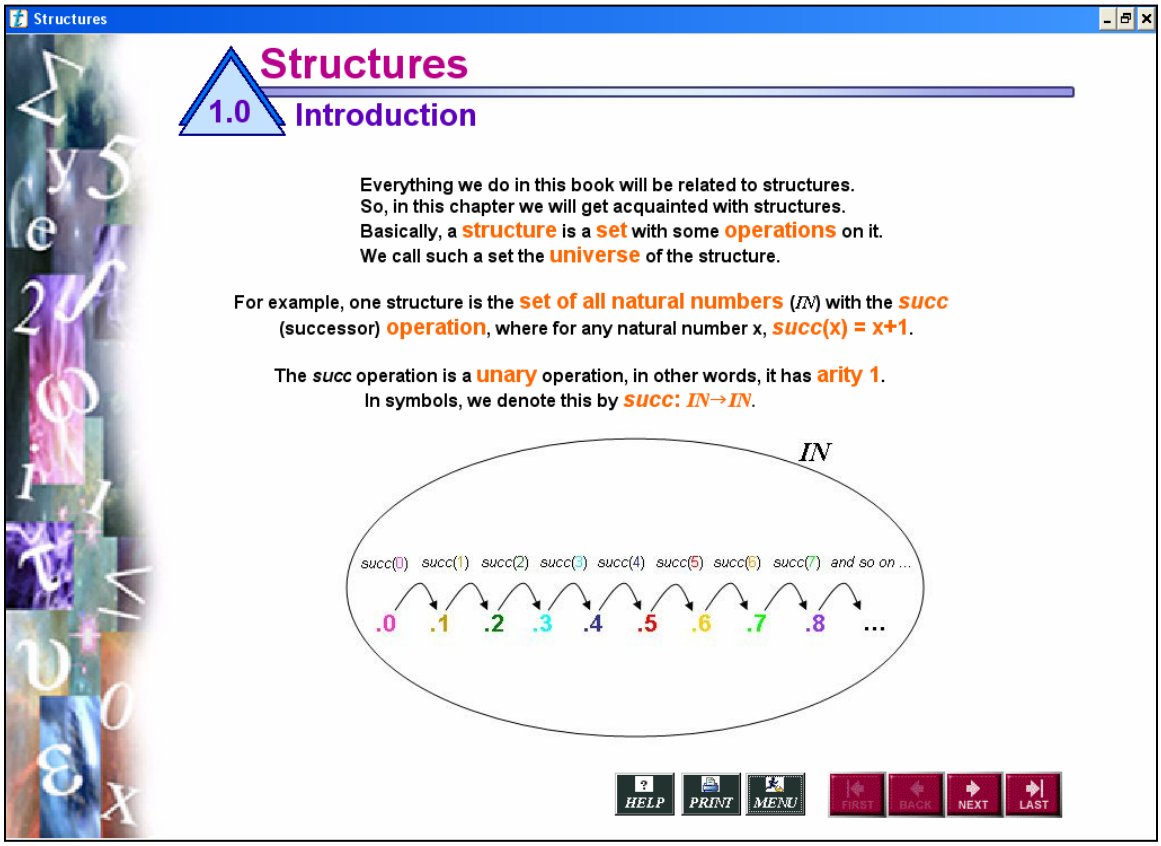
B.2 Navigation and Functionality

Throughout the e-book users have the choice to print the content of a page, access a help menu, go to the main menu from which they can move from one chapter to another, and they have a navigation bar available to be able to move from one page to the next and back.

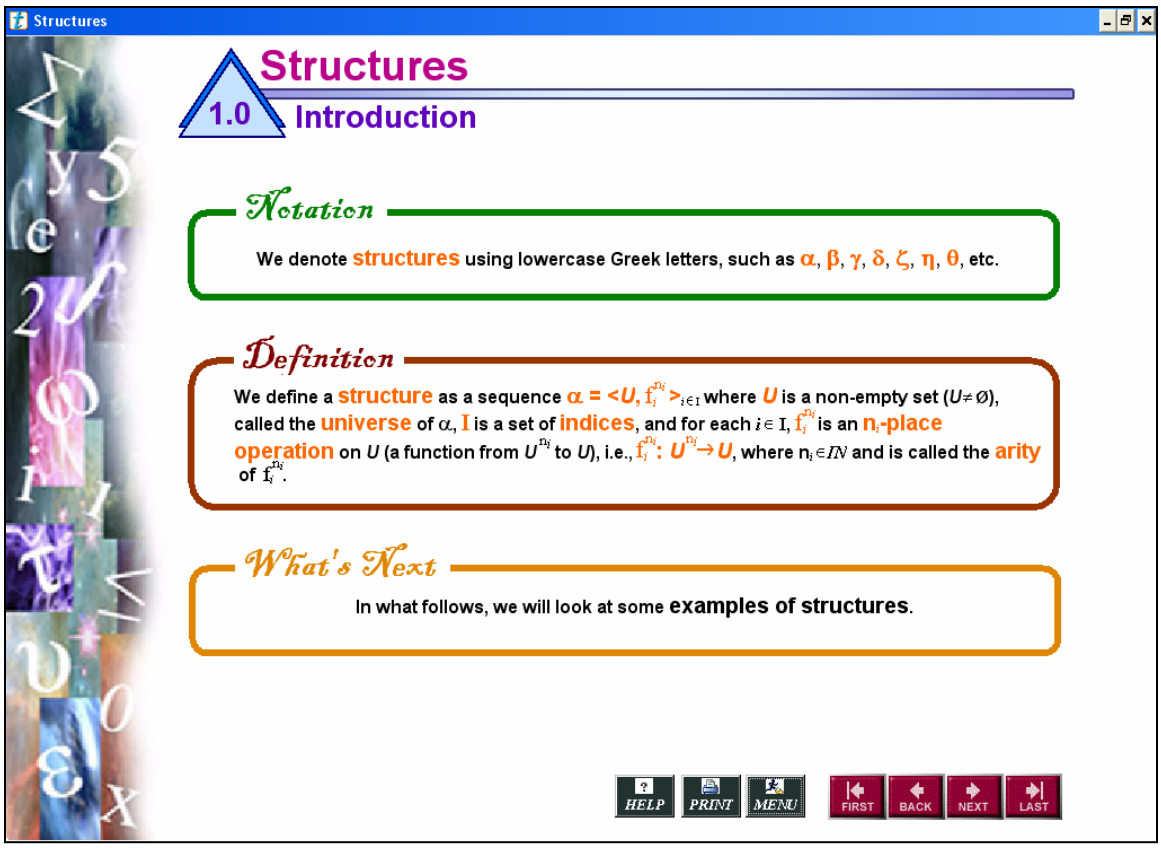


B.3 Chapter Organization

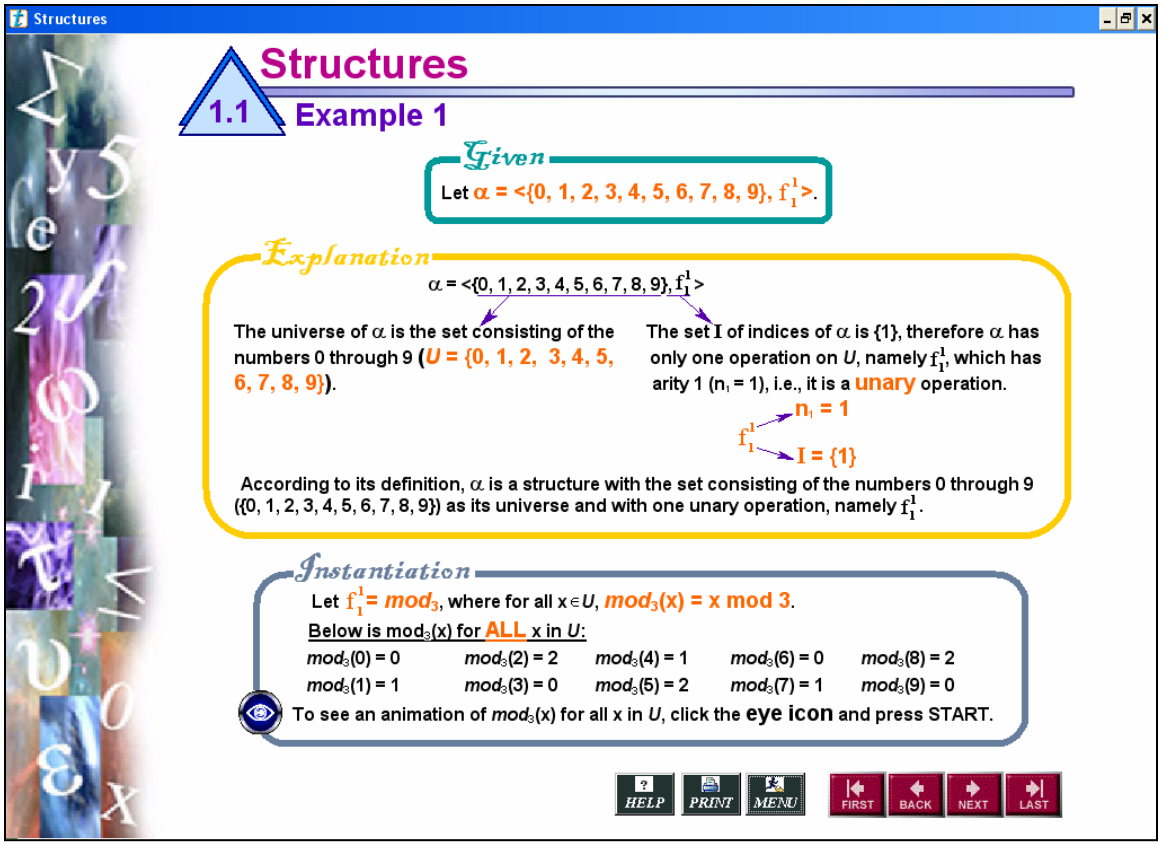
Each chapter in the e-book starts with an informal discussion of the concept at hand, followed by a formal definition of the concept, followed by a number of examples illustrating the definition, followed by a number of exercises, followed by a brief summary of the chapter and a brief introduction to the next chapter.
The following figures demonstrate the above organization through chapter 1: Structures.



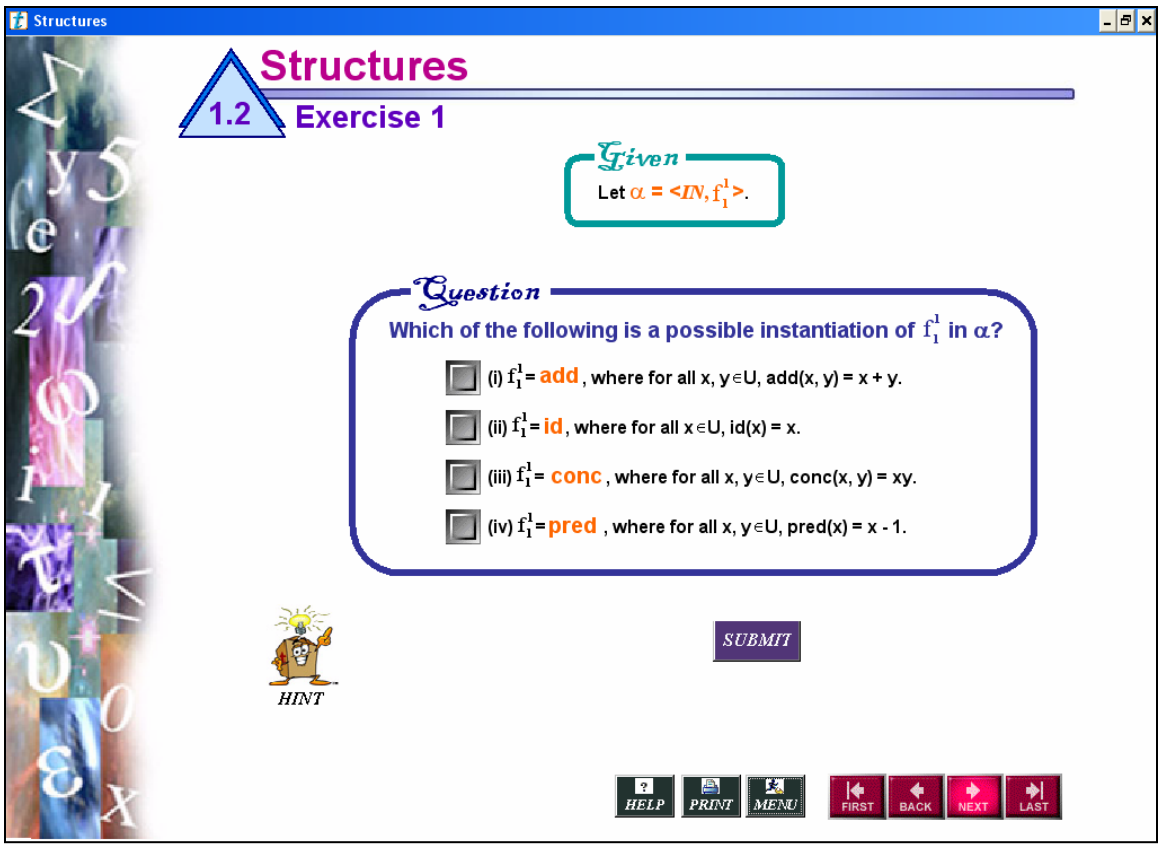
Informal discussion on Structures



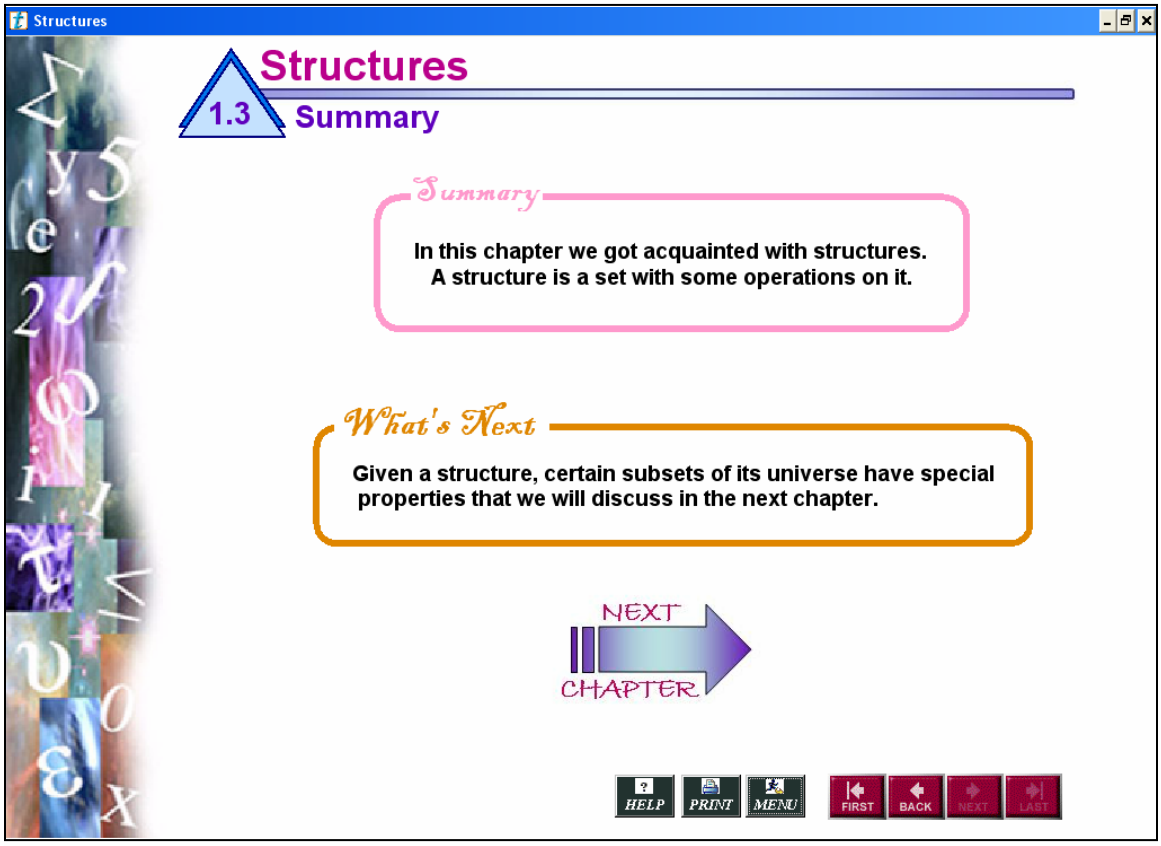
Formal definition of Structures



An example of a Structure.



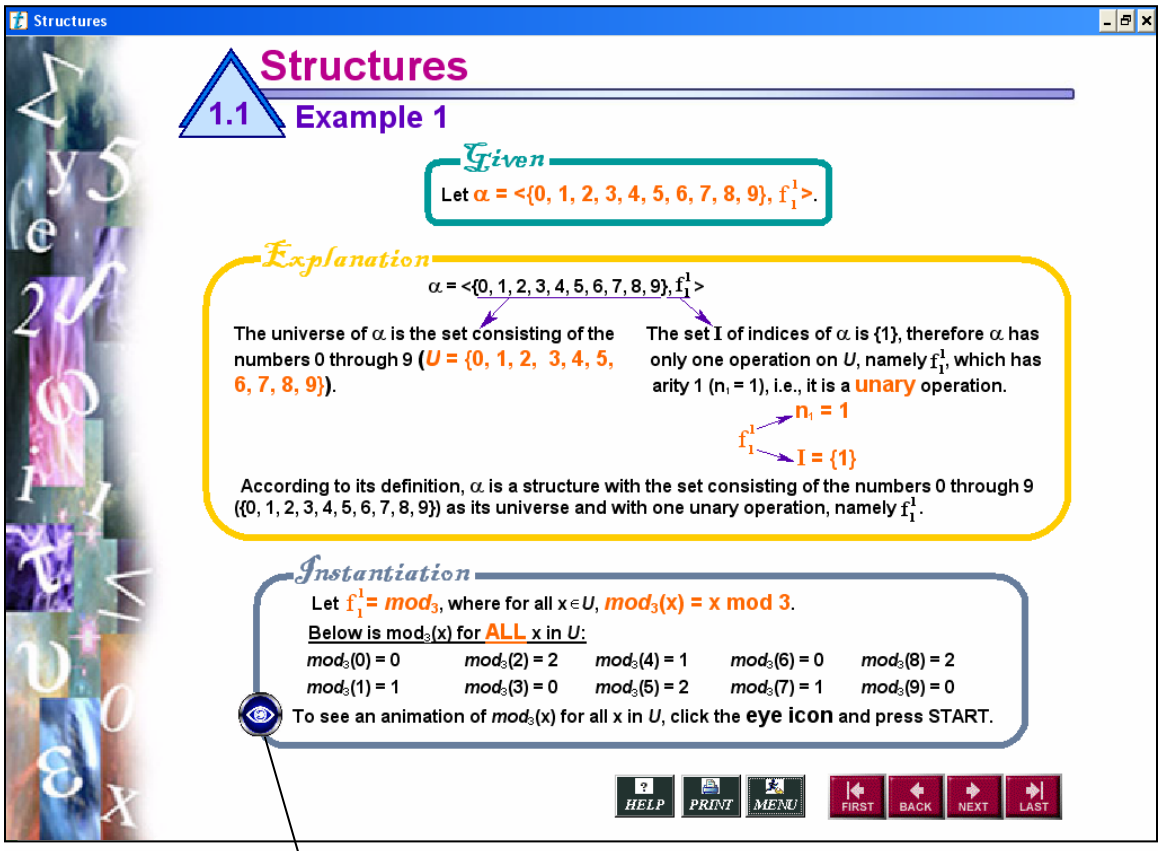
An exercise on Structures



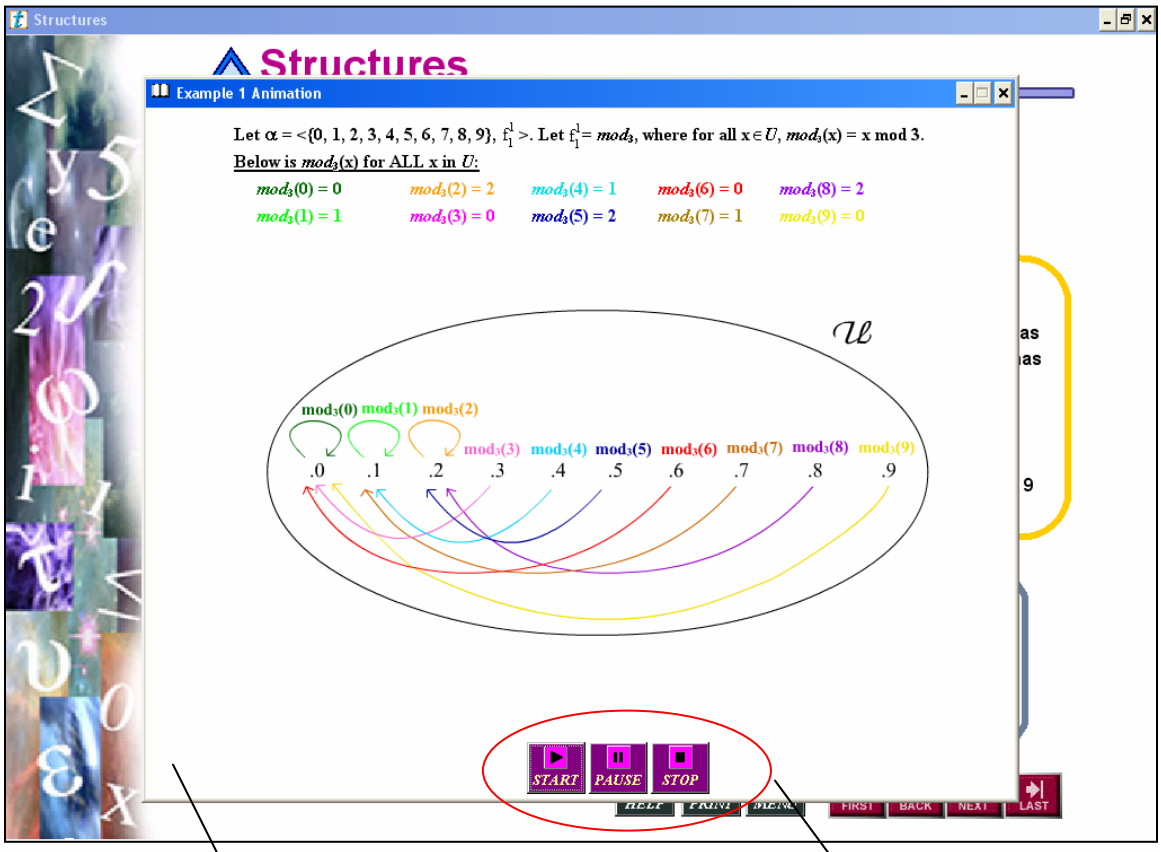
The summary page for Structures

B.4 Animations

Throughout the e-book students can view animations of different concepts. The pages that contain an animation have an animation icon (see figure below), so that students know in advance that they can view an animation. When the users click on the animation icon a pop up window appears in which the animation is presented. Users can control the animation in three ways: start the animation (*START*), pause the animation (*PAUSE*), and stop the animation (*STOP*).



Animation Icon
When users click the animation icon, the animation pop up window appears and the users can play the animation.

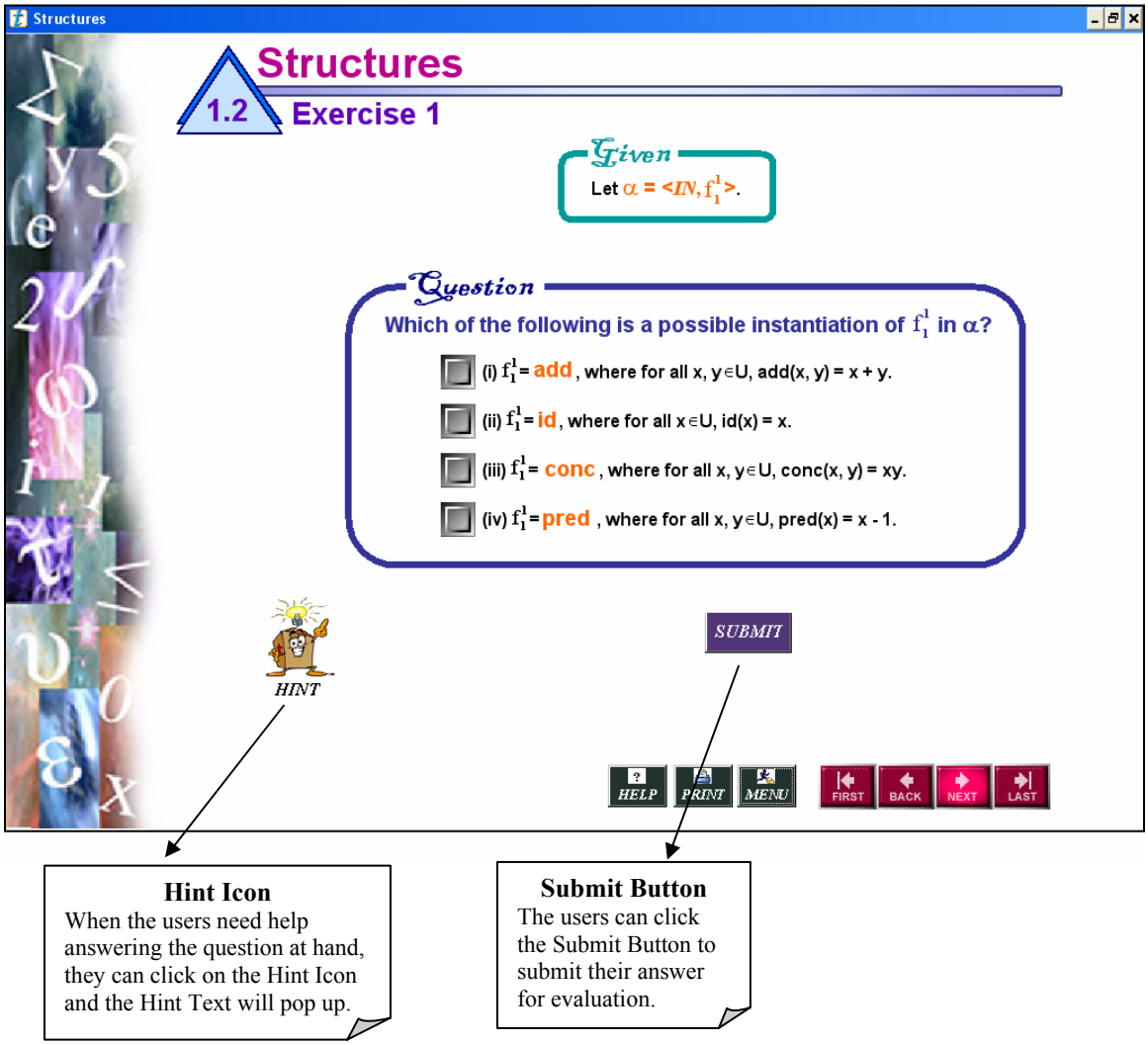


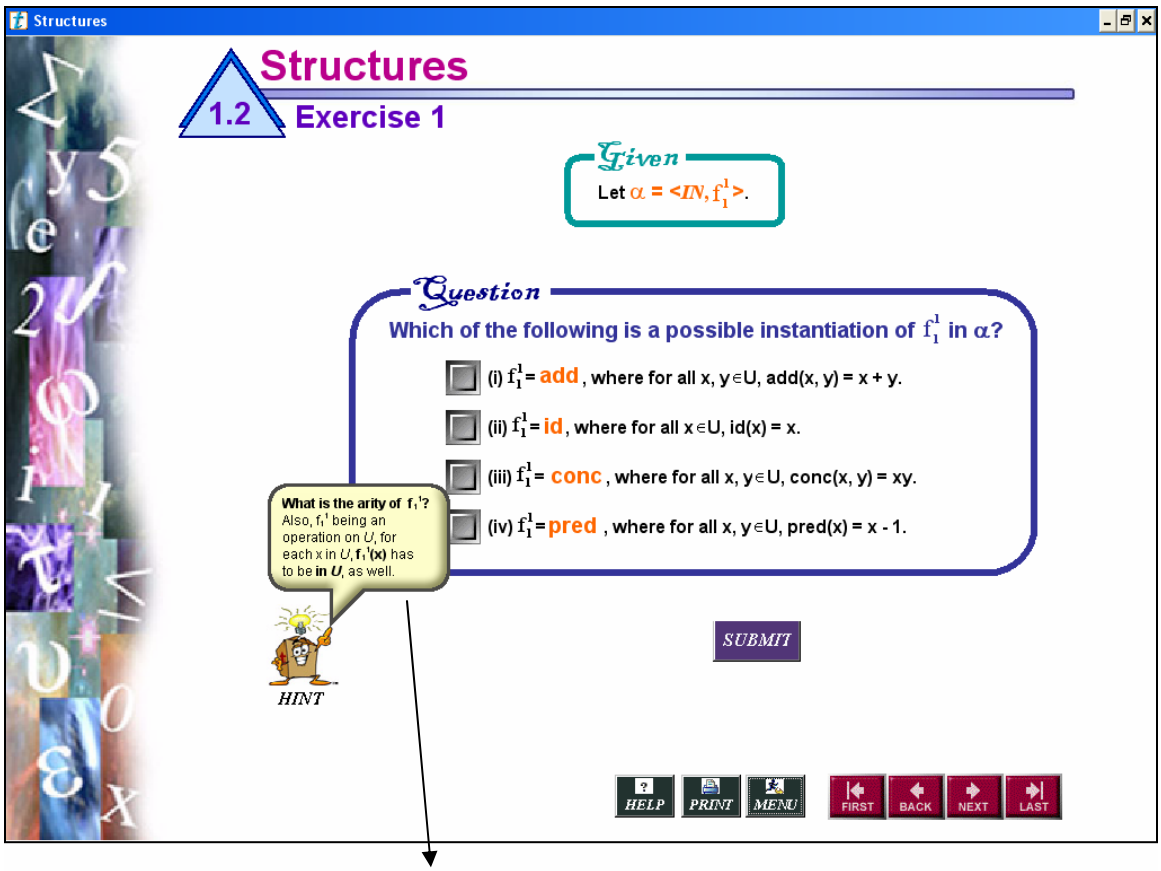
Animation Window
This is the pop window where the animation appears.

Animation Control
Users can start, pause, or stop an animation.

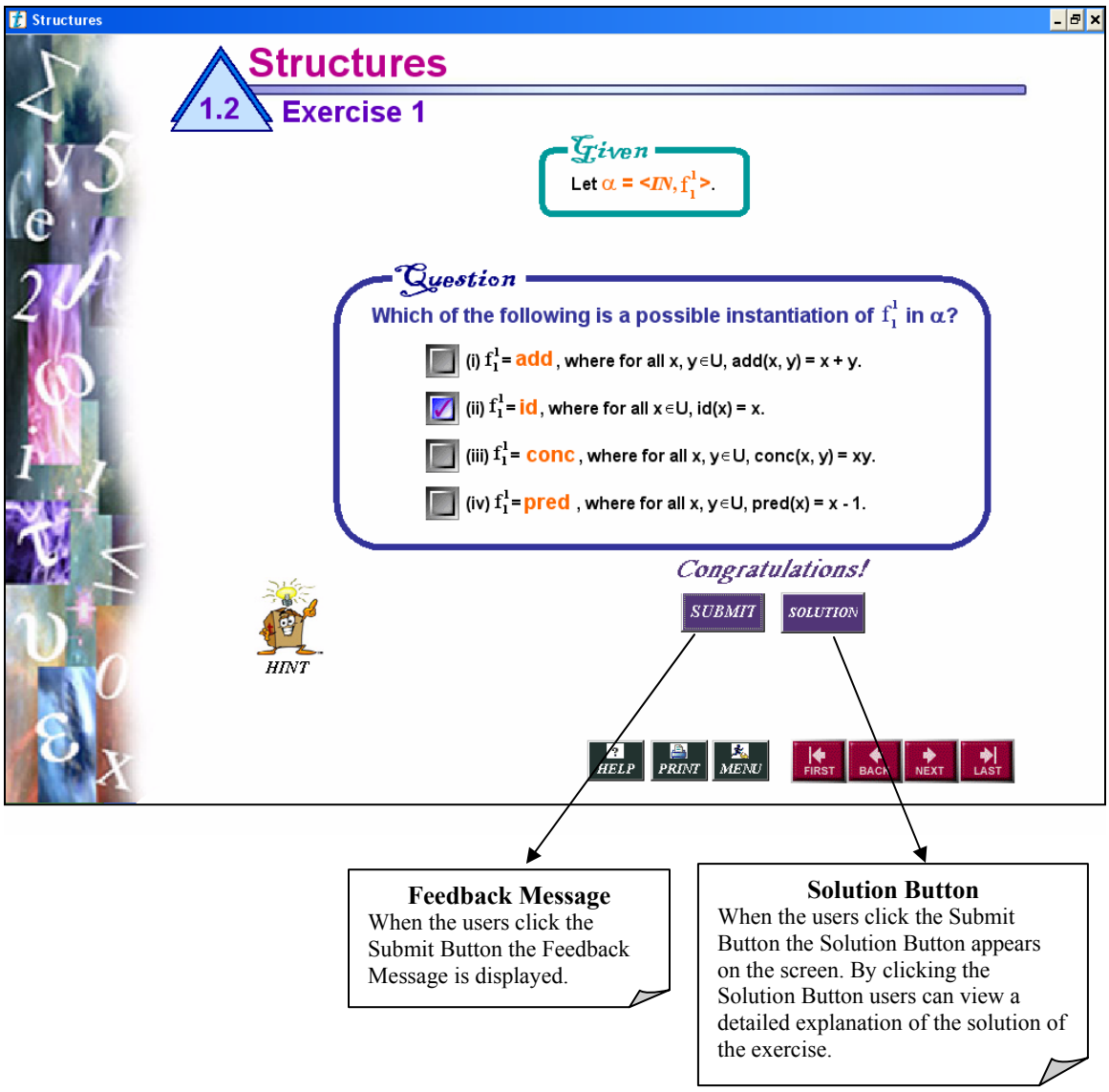
B.5 Exercises

For all exercises in the e-book, users can get a hint to help them answer the question, they can submit their answer and get feedback from the system, and they can view the solution of the exercise along with a detailed explanation.
In every exercise page, there is a Hint icon. When users click the Hint icon a pop up text area appears with the hint information. Users can click inside the text area for the hint information to go away. Also, in every exercise there is a Submit button. Users can click the Submit button to submit their answer and get feedback from the system. If the answer is correct the system displays the message "Congratulations!" and otherwise, the message "Incorrect Answer." Along with the feedback, the Solution button appears on the screen, next to the Submit button. When users click the Solution button, a pop up window appears with the solution of the exercise and a detailed explanation of it.





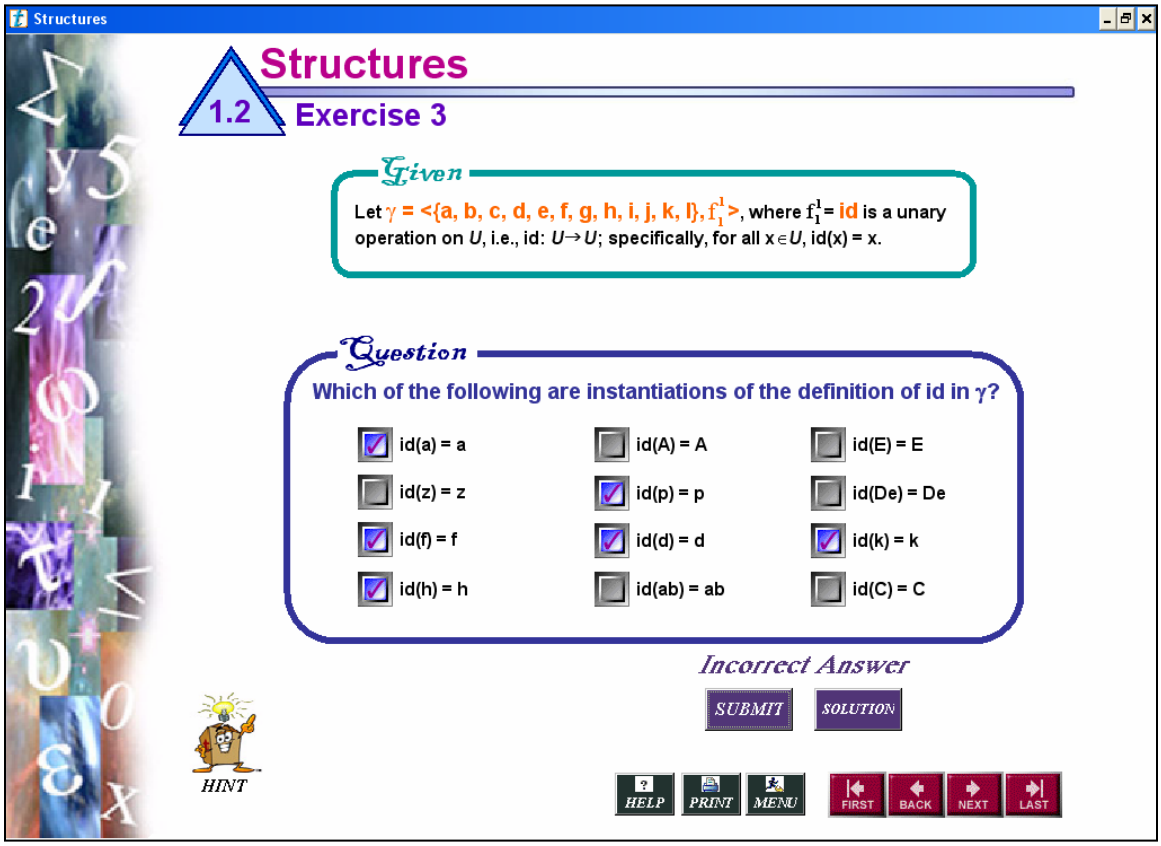
Hint Text
The Hint Text provides the users with guidance for answering the question at hand.

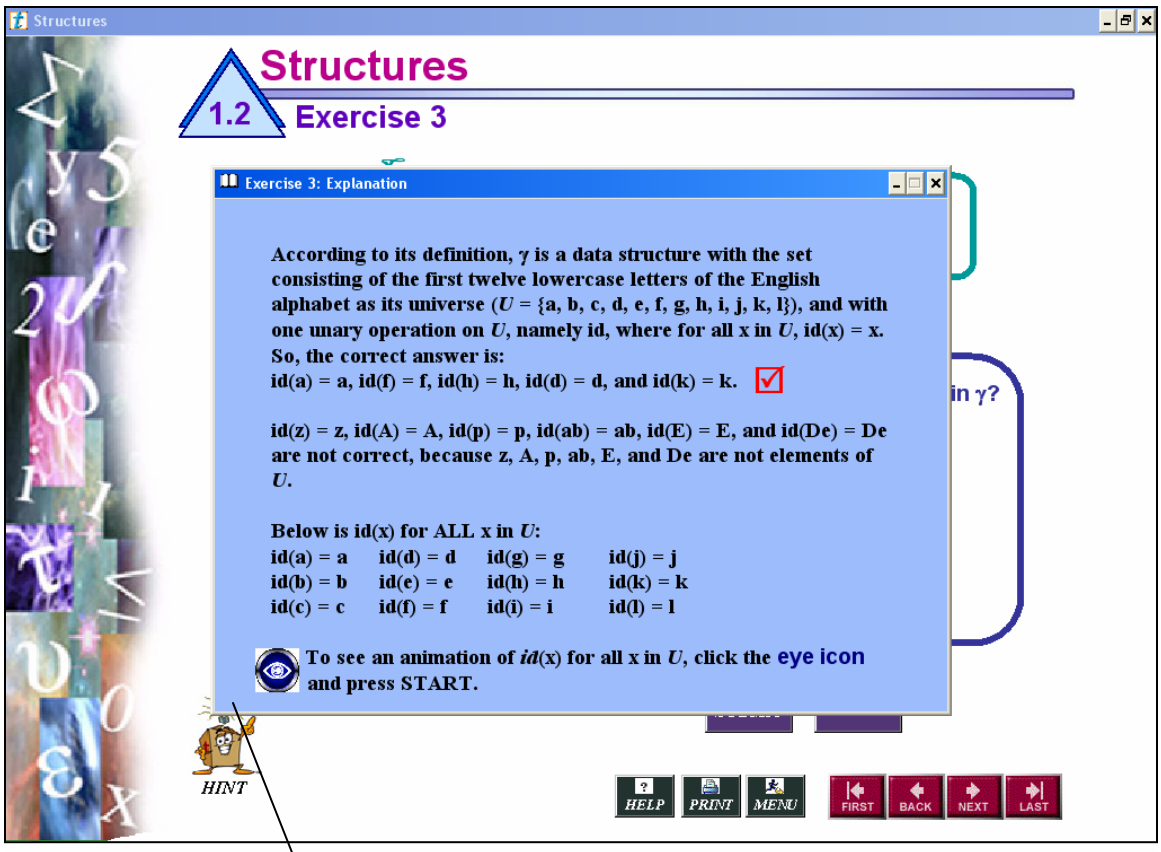


Feedback Message
When the users click the Submit Button the Feedback Message is displayed.

Solution Button
When the users click the Submit Button the Solution Button appears on the screen. By clicking the Solution Button users can view a detailed explanation of the solution of the exercise.

B.5.1 Solution to Exercises

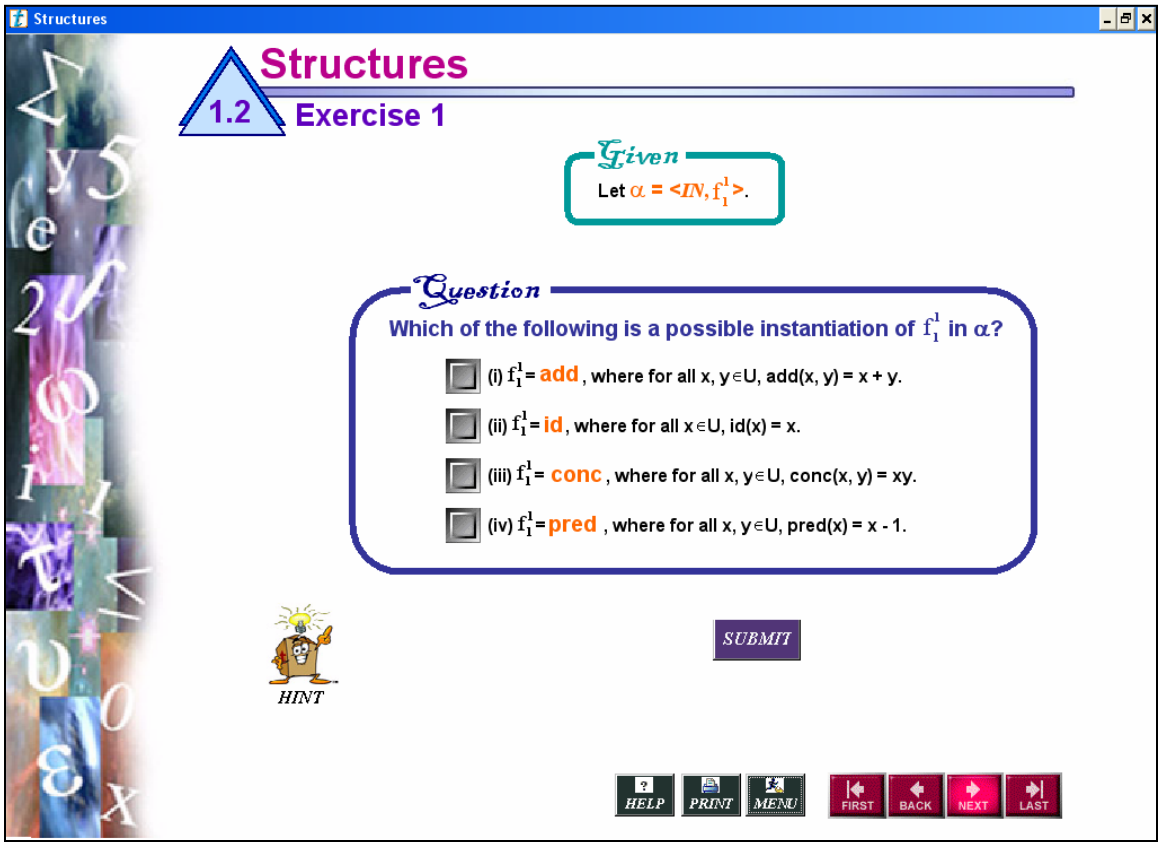




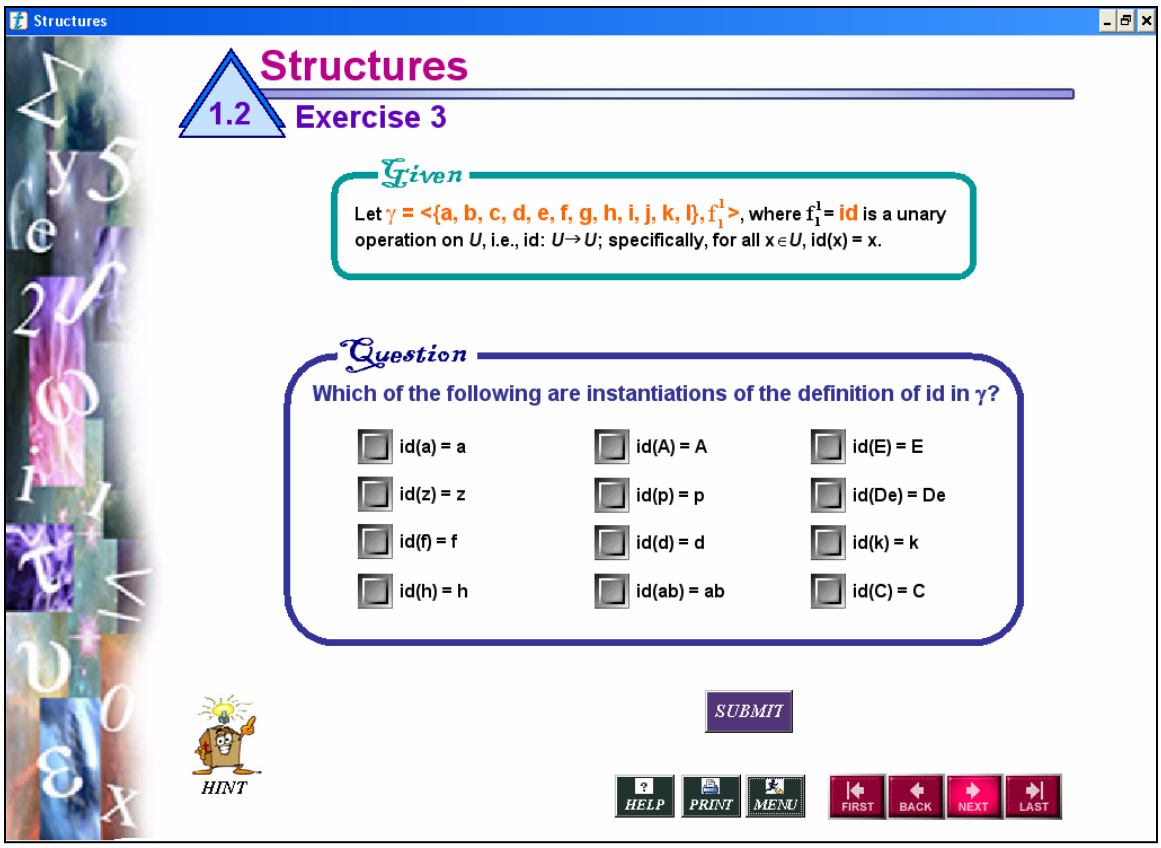
Solution Window
When users click the Solution Button on an exercise page, the pop up Solution Window appears with a detailed explanation of the solution of the exercise.

B.5.2 Type of Exercises

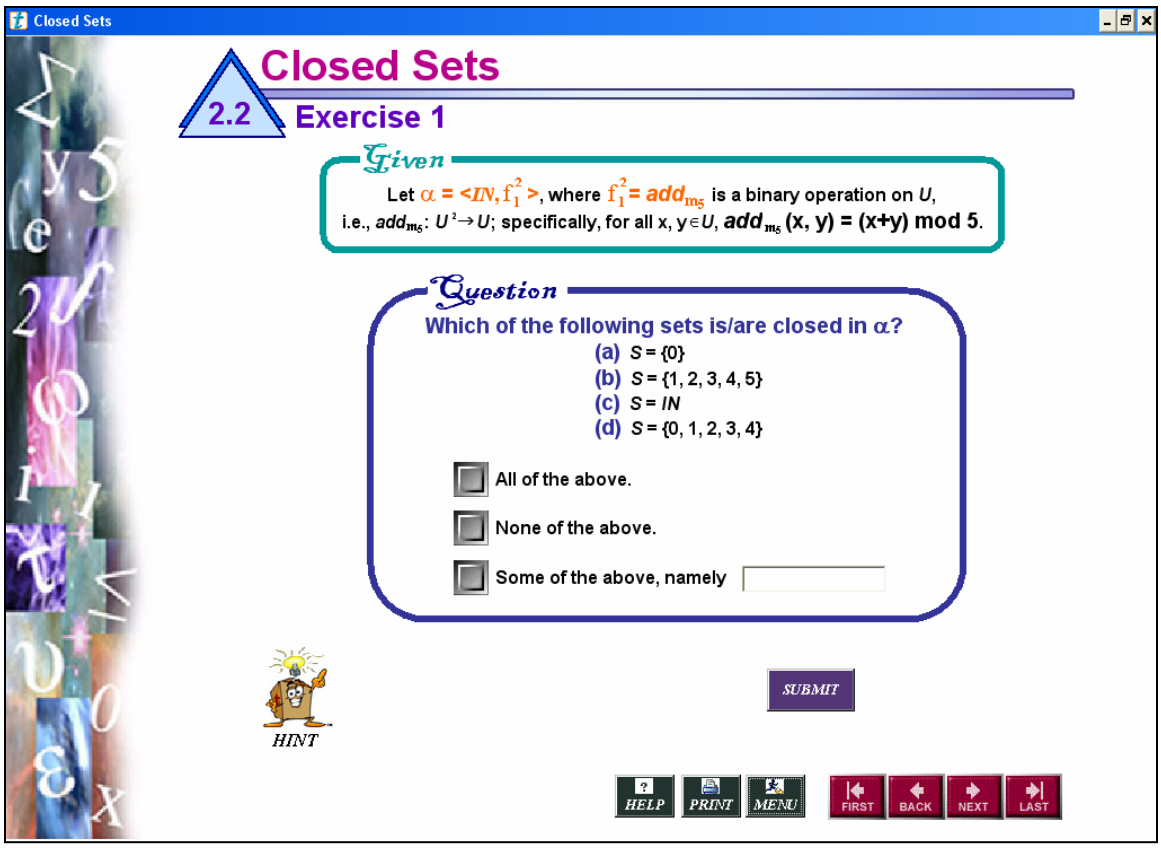
There are three major types of exercises in the e-book: multiple choice, fill in the blank, yes or no, and drag and drop exercises.



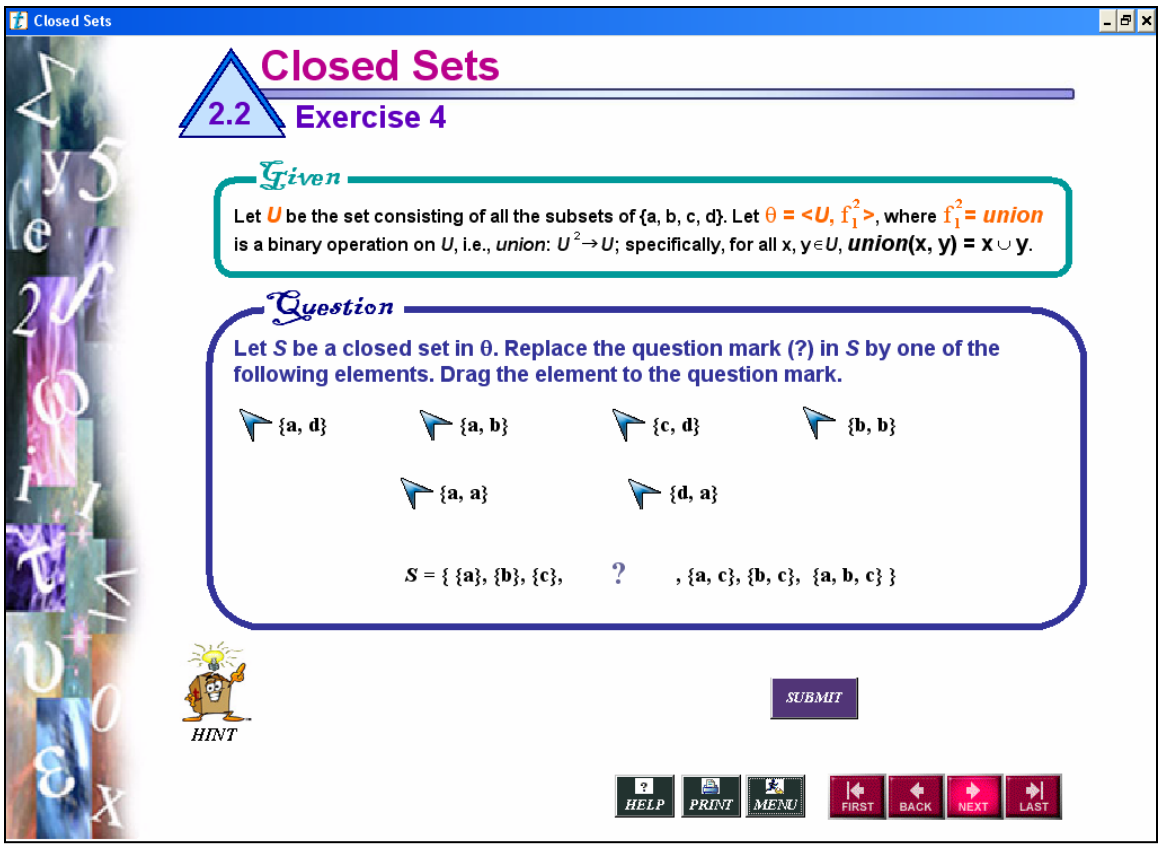
An example of a multiple choice exercise.



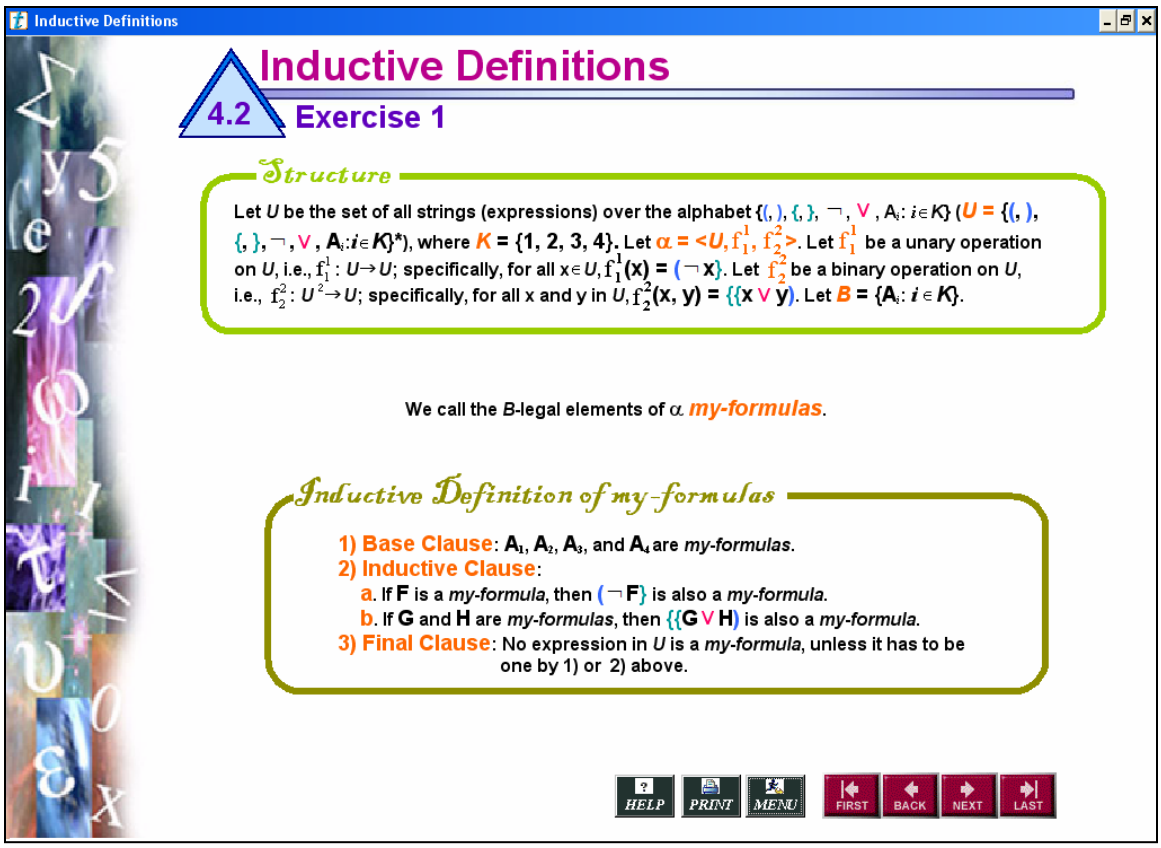
An example of a multiple choice/multiple answers exercise

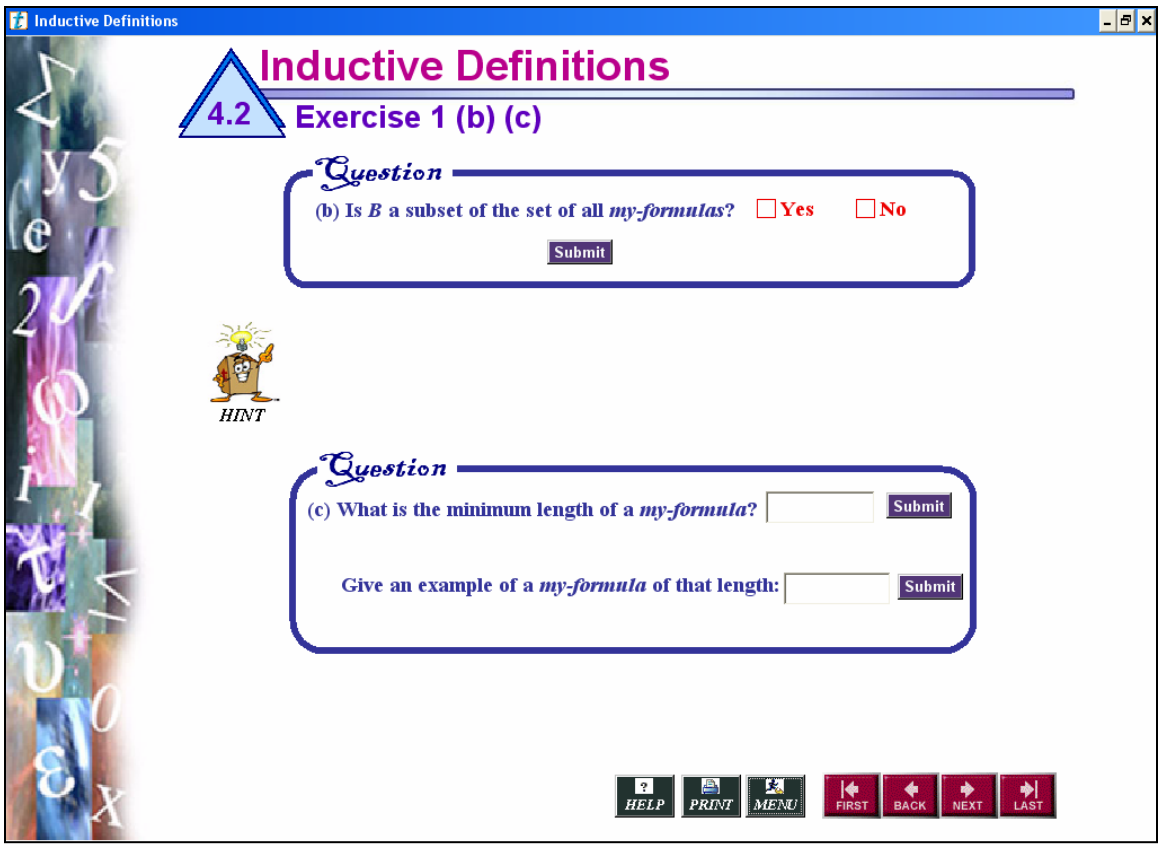


An example of a multiple choice exercise containing a fill in the blank field.

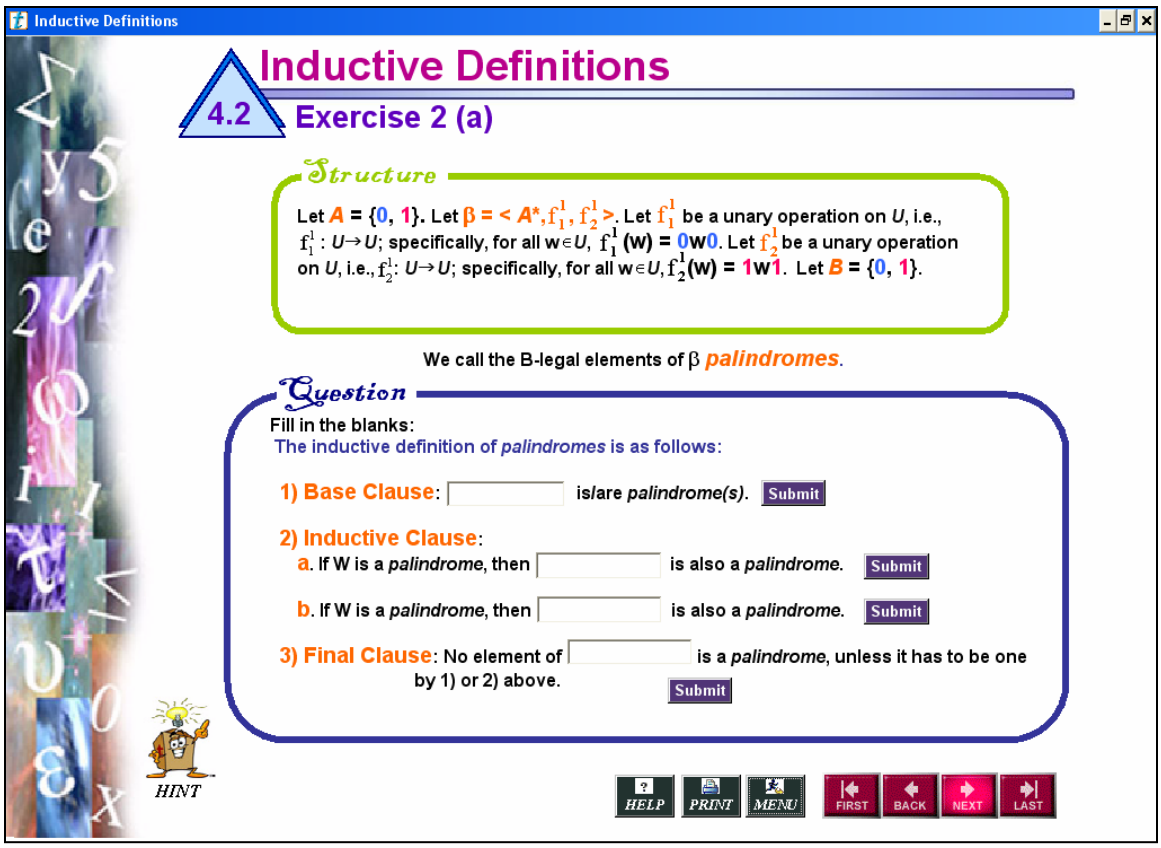


An example of a drag and drop exercise



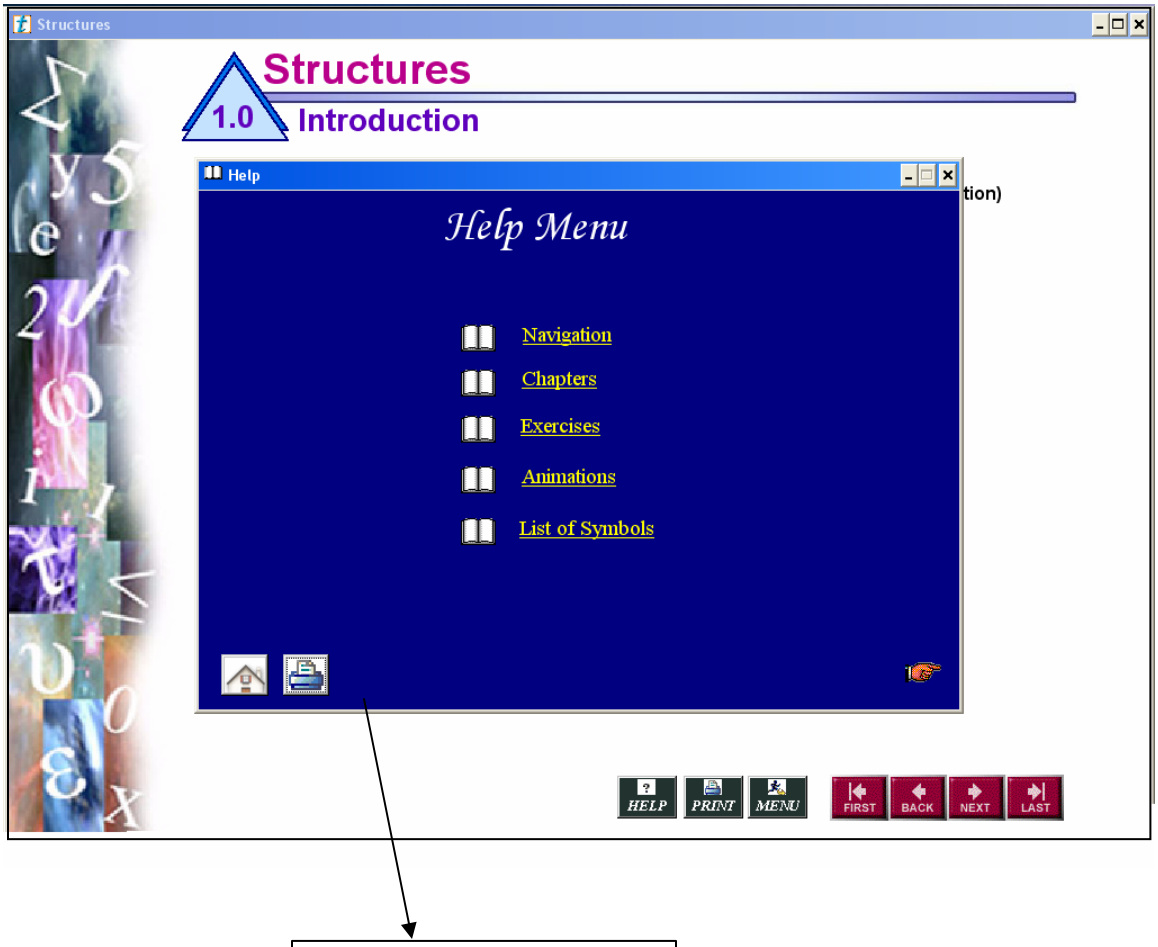


An example of a fill in the blank and a yes or no exercises.



An example of a fill in the blank exercise.

B.6 Help



Help Window
When users click the Help Button, the Help Window appears.

APPENDIX C

QUESTIONNAIRE

E-BOOK QUESTIONNAIRE

CLASS: COT 3420, Logic for Computer Science

NAME:

QUESTIONS ON CHAPTER 1: STRUCTURES

1) When you were presented with the first example in this chapter was it easy to understand what action to take to view the animation? Yes No

   If your answer is No, what was the problem?

2) When you were presented with the first Exercise in this chapter did you understand right away:
   a) How to select your answer? Yes No

      If your answer is No, what was the problem?

   b) How to view the solution of the exercise? Yes No

      If your answer is No, what was the problem?

3) In cases where the solution of an exercise is accompanied by an animation, do you prefer to see the animation first and then the text explanation or first the text explanation and then the animation? _____

   Explain in a few words why.

4) On a scale of 1 (easy) to 5, how would you rate:

   a) The level of difficulty of the reading material in this chapter?

      1 2 3 4 5

b) The level of difficulty of the exercises in this chapter?

1 2 3 4 5

5) Approximately, how many hours did you spend on this chapter? _____

6) Was the number of exercises in this chapter:
more than necessary sufficient insufficient

7) On a scale of 1 (least) to 5, how helpful were the animations in this chapter?

1 2 3 4 5

8) Were the animations in this chapter easy to understand? Yes No
If your answer is No, explain in a few words what your difficulties were.

9) Did the animations in this chapter enhance your understanding? Yes No
If your answer is No, explain in a few words why.

10) Did you find the exercise hints in this chapter:

not necessary necessary more than necessary

Do you have any suggestions to improve the exercise hints in this chapter?

11) Did you find the order that the material was presented in this chapter appropriate?

Yes No

If your answer is No, how would you change it?

12) Do you have any suggestions for improving this chapter?

QUESTIONS ON CHAPTER 2: CLOSED SETS

1) When you were presented with the first Exercise in this chapter did you understand right away how to select your answer? Yes No

If your answer is No, what was the problem?

2) On a scale of 1 (easy) to 5, how would you rate:

   a) The level of difficulty of the reading material in this chapter?

   1 2 3 4 5

   b) The level of difficulty of the exercises in this chapter?

   1 2 3 4 5

3) Approximately, how many hours did you spent on this chapter? _____

4) Was the number of exercises in this chapter:
   more than necessary sufficient insufficient

5) Did you find the order that the material was presented in this chapter appropriate?

   Yes No

   If your answer is No, how would you change it?

6) Did you find the exercise hints in this chapter:

   not necessary necessary more than necessary

   Do you have any suggestions to improve the exercise hints in this chapter?

7) Do you have any suggestions for improving this chapter?

QUESTIONS ON CHAPTER 3: INDUCTIVE SETS

1) When you were presented with the first Exercise in this chapter did you understand right away how to select your answer? Yes No

   If your answer is No, what was the problem?

2) On a scale of 1 (easy) to 5, how would you rate:

   a) The level of difficulty of the reading material in this chapter?

   1 2 3 4 5

b) The level of difficulty of the exercises in this chapter?

1 2 3 4 5

3) Approximately, how many hours did you spent on this chapter? _____

4) Was the number of exercises in this chapter:
more than necessary sufficient insufficient

5) Did you find the order that the material was presented in this chapter appropriate?

Yes No

If your answer is No, how would you change it?

6) Did you find the exercise hints in this chapter:

not necessary necessary more than necessary

Do you have any suggestions to improve the exercise hints in this chapter?

7) Do you have any suggestions for improving this chapter?

QUESTIONS ON CHAPTER 4: INDUCTIVE DEFINITIONS

1) When you were presented with the Exercises in this chapter did you understand right away:

a) How to select your answer? Yes No

If your answer is No, what was the problem?

b) How to view the solution of the exercise? Yes No

If your answer is No, what was the problem?

2) In cases where the solution of an exercise is accompanied by an animation, do you prefer to see the animation first and then the text explanation or first the text explanation and then the animation? _____

Explain in a few words why.

3) On a scale of 1 (easy) to 5, how would you rate:

a) The level of difficulty of the reading material in this chapter?

1 2 3 4 5

b) The level of difficulty of the exercises in this chapter?

1 2 3 4 5

4) Approximately, how many hours did you spend on this chapter? _____

5) Was the number of exercises in this chapter:

more than necessary sufficient insufficient

6) On a scale of 1 (least) to 5, how helpful were the animations in this chapter?

1 2 3 4 5

7) Were the animations in this chapter easy to understand? Yes No
If your answer is No, explain in a few words what your difficulties were.

8) Did the animations in this chapter enhance your understanding? Yes No
If your answer is No, explain in a few words why.

9) Did you find the order that the material was presented in this chapter appropriate?

Yes No

If your answer is No, how would you change it?

10) Did you find the exercise hints in this chapter:

not necessary necessary more than necessary

Do you have any suggestions to improve the exercise hints in this chapter?

11) Do you have any suggestions for improving this chapter?

QUESTIONS ON CHAPTER 5: PROOFS BY INDUCTION

1) On a scale of 1 (easy) to 5, how would you rate:

   a) The level of difficulty of the reading material in this chapter?

   1 2 3 4 5

   b) The level of difficulty of the exercises in this chapter?

   1 2 3 4 5

2) Approximately, how many hours did you spend on this chapter? _____

3) Was the number of exercises in this chapter:

   more than necessary sufficient insufficient

4) Did you find the order that the material was presented in this chapter appropriate?

   Yes No

   If your answer is No, how would you change it?

5) Do you have any suggestions for improving this chapter?

GENERAL QUESTIONS ON THE E-BOOK

1) On a scale of 1 (lowest) to 5, how would you rate:

   a) The navigation through this e-book?

   1 2 3 4 5

   b) The overall design of this e-book?

   1 2 3 4 5

   c) The ease of using this e-book?

   1 2 3 4 5

2) Did you like the sounds that accompanied this e-book's response to your answers (congratulations and incorrect answer responses)? Yes No

   If your answer is No, what would you prefer? Different Sound No sound

3) Did you encounter any difficulties/problems with this e-book? Yes No

   If your answer is Yes, describe in a few words what kind of difficulties/problems.

4) Is there anything else you would like to see in this e-book that can help you learn the concepts?

5) Do you have any suggestions for improving this e-book?

VITA

IRENE POLYCARPOU

November 19, 1979	Born, Cyprus
1997-2000	Associate Diploma in Computer Studies Higher Technical Institute of Cyprus Nicosia, Cyprus
2000-2002	B.S., Computer Science Florida International University Miami, Florida
2002-2004	M.S., Computer Science Florida International University Miami, Florida
2004-2008	Doctoral Candidate in Computer Science Florida International University Miami, Florida
	Teaching and Research Assistant School of Computing and Information Sciences Florida International University Miami, Florida

PUBLICATIONS

Polycarpou, I. and Pasztor, A. (2008). *An Interactive and Multimodal Software for Teaching Induction for Computer Science*. In the Proceedings of the 2008 International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS'08).

Polycarpou, I. (2008). *Induction as a Tool for Conceptual Coherence in Computer Science*. In the Proceedings of the 46th ACM Southeast conference.

Polycarpou, I, Pasztor, A., and Adjouadi, M. (2008). *A Conceptual Approach to Teaching Induction for Computer Science*. In the Proceedings of the 39th ACM Technical Symposium on Computer Science Education (SIGCSE'08), pp.9-13.

Polycarpou, I. (2007). *A New Approach to Teach Proofs by Induction for Computer Science*. In the Proceedings of the Grace Hopper 2007 conference. (Poster Session)

Polycarpou, I., Pasztor, A., and Alacaci, G. (2006). *Sources of Students' Difficulties with Proofs by Induction: A Study*. In the Proceedings of the Third International Conference in Teaching Mathematics (ICTM3) in Turkey, paper 446.

Polycarpou, I. (2006). *Computer Science students' difficulties with proofs by induction: An exploratory study*. In the Proceedings of the 44th ACM Southeast conference, pp.601-606.

Polycarpou, I. and Achilleos, P. (2000). *Cypriot Artists*. Multimedia software about Cypriot artists. Ministry of Education of Cyprus.

Polycarpou, I. and Achilleos, P. (2000). *Design and Analysis for Cypriot Artists software*. Ministry of Education of Cyprus.

Polycarpou, I. and Achilleos, P. (2000). *User Manual for Cypriot Artists software*. Ministry of Education of Cyprus.

The last three publications are available in all public libraries of Cyprus.

AWARDS

Florida International University Graduate School *Dissertation Year Fellowship* (2008).

Graduate Assistantship, sponsored by National Science Foundation (2005-2007).

Grace Hopper Conference Travel Scholarship, sponsored by National Science Foundation (2007).

VMware Academic Scholarship (2007).

Dean's Leadership and Community service Scholarship from the College of Engineering and Computing at Florida International University (2006 and 2007).

Outstanding Service award from the College of Engineering and Computing at Florida International University (2006).

Best Student Leadership award from the School of Computing and Information Sciences at Florida International University (2005).

Best Scientific Project award from the Cyprus Institute of Neurology and Genetics for the development of "Cypriot Artists" multimedia software for Ministry of Education of Cyprus (2000).