## Analogies Are Like Bowling Balls, or Why Analogies to English Need Some Explanation to Help Students Learn Scheme

Clint Eric Ryan

Electrical Engineering and Computer Sciences University of California at Berkeley

Technical Report No. UCB/EECS-2006-75
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-75.html
May 22, 2006

Copyright © 2006, by the author(s). All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

# Analogies Are Like Bowling Balls, <br> or <br> Why Analogies to English Need Some Explanation to Help Students Learn Scheme 

by

Clint Eric Ryan
B.S. (University of Arkansas at Fayetteville) 2000

A thesis submitted in partial satisfaction of the requirements for the degree of

Master of Science
in

Computer Science
in the

GRADUATE DIVISION
of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Senior Lecturer Michael Clancy, Chair
Lecturer Brian Harvey
Professor Marcia Linn

The thesis of Clint Eric Ryan is approved:

| Chair | Date |
| :---: | :---: |
|  |  |
|  | Date |

## Date

University of California, Berkeley
Spring 2006

# Analogies Are Like Bowling Balls, 

or
Why Analogies to English Need Some Explanation to Help Students Learn Scheme

Copyright (c) 2006
by

Clint Eric Ryan

To my wife

## Clarissa

to whom I dedicate pretty much everything.

## Contents

Contents ..... ii
List of Figures ..... v
List of Tables ..... vii
Acknowledgments ..... ix
1 Introduction ..... 1
1.1 Problem Statement ..... 1
1.2 Lists ..... 1
1.3 Words and Sentences in CS 3 ..... 5
1.4 Classroom Observations ..... 6
1.5 Motivation ..... 8
1.6 Goals and Design ..... 9
2 Theory ..... 10
2.1 Misconceptions ..... 10
2.2 Analogy, Similarity, and Structure Mapping ..... 21
3 Materials and Methods ..... 26
3.1 Population ..... 26
3.2 Differences Among Semesters ..... 29
3.3 Interviews ..... 42
3.4 Interview Procedure ..... 63
3.5 Scoring ..... 64
3.6 Possible Treatments ..... 65
4 Results ..... 66
4.1 One-Word Sentences ..... 66
4.2 Empty Words and Sentences ..... 81
4.3 Essential Elements of English and Scheme Sentences ..... 92
4.4 What Students Did Not Say About English, Scheme, and Collections ..... 94
4.5 Comparison of Sentences and Lists ..... 95
4.6 Other Observations ..... 97
5 Discussion ..... 99
5.1 Summary of Results ..... 99
5.2 Sources of Mistakes ..... 100
5.3 Success or Failure of Treatments ..... 107
5.4 Limitations and Weaknesses ..... 109
6 Conclusions and Future Work ..... 111
6.1 Conclusions ..... 111
6.2 Implications for Instruction ..... 112
6.3 Future Work ..... 115
References ..... 121
Bibliography ..... 126
A Interview Questions ..... 127
A. 1 Spring A ..... 127
A. 2 Summer A ..... 134
A. 3 Fall A ..... 135
A. 4 Spring B ..... 141
$B$ Changes in the Curriculum ..... 142
B. 1 Additions to Summer A ..... 142
B. 2 Additions to Fall A ..... 144
B. 3 Additions to Spring B ..... 145
C A Quick Introduction to Scheme ..... 148
C. 1 Scheme Basics ..... 148
C. 2 Working with Words and Sentences ..... 150
C. 3 Working with Lists ..... 152
C. 4 More Advanced Scheme ..... 152

## List of Figures

2.1 Incorrect Logo and Correct BASIC for the Same Problem ..... 11
3.1 An Analogy for Words ..... 34
3.2 Comparing English and Scheme ..... 35
3.3 Comparing English and Scheme, Revised ..... 37
3.4 WebScheme Activity on Empty Words and Sentences ..... 38
3.5 Defining Terms ..... 38
3.6 Question O5, negate-all ..... 47
3.7 Question O6, divide-by-largest Main Handout ..... 47
3.8 Question O6, divide-by-largest Test Cases Handout ..... 48
3.9 Question O6, divide-by-largest Code Handout ..... 48
3.10 Question E5, number-spell ..... 52
3.11 Question E6, sum-of-square-roots ..... 53
4.1 Questions O1-O4, butfirst and butlast of One-Word Sentences ..... 66
4.2 Question O5, negate-all ..... 71
4.3 Summary of Question O6, divide-by-largest ..... 75
4.4 Questions E1 and E2, Creating Empty Sentences ..... 81
4.5 Questions E3 and E4, empty words and sentences ..... 84
4.6 Question E6, Empty Words as Errors ..... 90
4.7 Question E7, Placing an Empty Word in a Sentence ..... 91
4.8 Questions P1 and P2, Essential Elements of Sentences ..... 92
5.1 Warning from Exploring Computer Science with Scheme ..... 103
B. 1 An Analogy for Words . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 143
B. 2 Comparing English and Scheme, Summer A . . . . . . . . . . . . . . . . . . 144
B. 3 Comparing English and Scheme, Fall A and Spring B . . . . . . . . . . . . . 145
B. 4 Defining Terms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 146
B. 5 WebScheme Activity About Empty Words and Sentences . . . . . . . . . . 147

## List of Tables

3.1 Number of Students Enrolled and Interviewed ..... 28
3.2 Exam Scores ..... 29
3.3 Changes in Class Format Between Semesters ..... 39
3.4 Differences Between Semesters ..... 39
3.5 Curriculum Changes by Semester ..... 40
3.6 Topics by Week for Spring A-Spring B ..... 41
3.7 Possible Elements of a Mapping Between English and Scheme ..... 43
3.8 Possible Elements of a Mapping Between Collections and Scheme Sentences ..... 44
3.9 Common Mistakes and Possible Reasons for Them ..... 45
3.10 Spring A Interviews ..... 57
3.11 Summer A Interviews ..... 58
3.12 Fall A Interviews ..... 59
3.13 Spring B Interviews ..... 61
4.1 One-word Sentence Creation, butfirst ..... 67
4.2 One-word Sentence Creation, butlast ..... 67
4.3 Semester Summary for the butfirst and butlast of Two- and Four-word Sentences ..... 70
4.4 Responses to the Negate-all Question ..... 70
4.5 Semester Summary for the negate-all Question ..... 75
4.6 Responses to the Divide-by-largest Question, Test Cases ..... 76
4.7 Responses to the Divide-by-largest Question, Code ..... 77
4.8 Semester Summary for the divide-by-largest Question ..... 80
4.9 Creating Empty Sentences ..... 82
4.10 Semester Summary for Taking the butfirst of a One-word Sentence ..... 84
4.11 Sentences of Empty Words ..... 85
4.12 Semester Summary for Sentences of Empty Words ..... 88
4.13 Mistakes for (number-spell 11000) ..... 88
4.14 Putting (bf 1) in a Sentence ..... 91
4.15 Essential Elements of English and Scheme Sentences ..... 93
4.16 Averages for Sentence and List Questions ..... 95
5.1 Students' Problems in the Spring A Semester ..... 101
5.2 Improvements from Spring A ..... 101
5.3 Success and Failure of Treatments ..... 109

## Acknowledgements

My thanks to Michael Clancy, Nate Titterton, Brian Harvey, and Dan Garcia for being my mentors. I feel very fortunate to have learned and to continue to learn from you all. I also owe a debt to Jieae Goo, Jennifer Tsang, Emily Watt, and Kathy Chong for helping me talk to students. Without you, I never could have hoped to gather information from so many CS 3 students. Finally, I would like to thank my students, from whom I learned so much.

## Chapter 1

Words should mean what they mean. That's what words are for. Once words start meaning more than one thing, it's not a language, it's a lottery.

## Introduction

Satan, Old Harry's Game

### 1.1 Problem Statement

Many common misunderstandings among students learning to program in Scheme involve lists, which are containers for data. In particular, students confuse the procedures that assemble or disassemble lists. Every course deals with this problem in its own way. For example, three of the most commonly used introductory Scheme books favor avoiding or delaying some or all of the details of lists. Berkeley's CS 3 "Introduction to Symbolic Programming" class replaces lists with words and sentences during the first part of the semester. This study is intended to discover what kinds of mistakes students in CS 3 make with words and sentences, why they make them, and what can be done to avoid them in the future. In particular, I have focused on how students understand the concept of "empty" for words and sentences, as well as the difference between words and one-word sentences.

### 1.2 Lists

A list is a type of data that is used to hold and group together other data. For example, a list might be used to represent a date by holding the day, the name of the month, and
the year. When written on paper or typed on a computer, it begins with a left parenthesis, contains zero or more elements, and concludes with a right parenthesis. For example, the list (8 october 1977) contains the elements 8, october, and 1977. While a list may contain several individual elements, it is one single object. Thus, a Scheme program would treat (8 october 1997) as one list rather than as two numbers and the symbol october.

## Difficulties With Lists

There are several aspects of lists that often confuse students. First, as mentioned above, a list is a single object rather than many objects grouped together. Students who are new to lists often write programs that treat the elements of lists as though they were not in a list. For example, students asked to work with a list of two numbers that represents a measurement in feet and inches should write programs that extract each number from the list as needed, but many write programs that deal directly with the two numbers as though they were not in a list at all.

Another source of confusion is that lists can contain any kind of data that Scheme supports. This includes numbers, symbols (like october), boolean values (true and false), and, in particular, other lists. Just as students learning about set theory in mathematics have difficulty understanding sets inside sets (Zazkis \& Gunn, 1997), students learning about lists in Scheme classes may have trouble understanding lists inside lists. For example, a list of lists might be used to represent a list of dates: ( (16 september 1980) (11 june 1934) (18 august 1990)). This list contains three elements, although some students would believe it contains nine. The first thing in the list is the list (16 september 1980), but some students would think it is 16 . For students who overcome these difficulties, there is still the potential confusion of dealing with a list inside a list inside a list inside a list inside a list.

Many problems with lists involve taking them apart and putting them together. Two lists can be combined in several different ways. (12) and (3 4) could be combined into
a single list with four numbers, (1 23 4 4), one list which contains two numbers and a list of two numbers, ( (12) 3 4), and a list of two lists, each with two numbers, ( ( 12 ) (3 4)). Furthermore, different procedures are required to extract the number 4 from each of these three lists.

A similar problem is that adding or removing things on the left side of a list is different from adding or removing things on the right side of a list. For example, putting the number 1 into the list (2 3) to produce (1 2 3) is different from combining the list (12) and the number 3 to produce the same result. This is due to the way Scheme represents lists internally, and it may seem entirely illogical until students learn about the inner workings of Scheme.

Students have difficulty properly using parentheses and quotes (Davis, Linn, Mann, \& Clancy, 1993). Parentheses are used to show where lists begin and end, but they are also used for a number of other purposes in Scheme programs. Quotes (written') identify lists and symbols as data rather than as code. ${ }^{1}$ While problems with parentheses and quotes are not unique to the learning of lists, they certainly make learning lists more difficult.

With three different ways to build lists, a student trying to produce the list of dates described earlier could end up with (16 september 198011 june 193418 august 1990) or ((16 september 1980) (11 june 1934) 18 august 1990). For a student who has trouble seeing that this second list contains five items instead of three or nine, and who may not have a solid understanding of how to access the individual components of each date, even if the list was correct, this situation can be extremely confusing.

## Approaches to Lists in Popular Textbooks

One of the standard introductions to computer science using Scheme is The Structure and Interpretation of Computer Programs (Abelson, Sussman, \& Sussman, 1996). This

[^0]book introduces students to a number of advanced topics, such as recursion, higher-order procedures, and order of growth before it introduces lists or any data types other than numbers and procedures. As a result, early examples involve finding roots of functions or computing the Fibonacci numbers in a logarithmic number of steps. The first full procedure students are exposed to is square, followed by sum-of-squares and a program that calculates the sum of the squares of $x+1$ and $2 x$. Students who love mathematics may be fine with these exercises, but others may not remain in the class long enough to experience anything else.

How to Design Programs: an Introduction to Programming and Computing (Felleisen, Findler, Flatt, \& Krishnamurthi, 2001), on the other hand, introduces lists late and slowly. Students must wait until chapter 9 to begin working with them, and at that point they learn only one method of constructing lists: they can add something to the front of a list. While this does keep students from getting confused, it also makes lists rather uninteresting and clunky, and it forces students to type more. For example, the list ( $\begin{array}{llll}1 & 2 & 3 & 4\end{array}$ 5) would have to be written as (cons 1 (cons 2 (cons 3 (cons 4 (cons 5 empty))))).

Simply Scheme (Harvey \& Wright, 1994) replaces lists with "sentences" and symbols with "words" in the first part of the book, although students are later introduced to lists. Sentences are like lists that can only contain words, and words are groups of symbols: letters, digits, and other characters. Although there are several ways to build a list, in this approach there is only one way to build a sentence. The two sentences (a) and (sentence) are put together into (a sentence) in the same way that the word a and sentence (sentence), the sentence (a) and word sentence, and the two words a and sentence are. ${ }^{2}$ This avoids most of the problems described earlier. In particular, sentences were intended to be symmetricadding or removing things on the left side of a sentence is as easy as adding or removing things on the right side.

[^1]Furthermore, words can be manipulated in almost the same way as sentences. Sentences hide much of the complexity that confuses students, but the combination of sentences and words allow students to solve reasonably interesting problems. For example, students studying recursion can easily write a program that converts individual words or whole sentences from English to Pig Latin.

### 1.3 Words and Sentences in CS 3

Since the Spring semester of 2001, the U.C. Berkeley CS 3 "Introduction to Symbolic Programming" class has used the Simply Scheme textbook. This book defines a sentence as a list that contains zero or more words and a word as a group of zero or more characters. That is, $\mathrm{B}, \mathrm{x}-303$, and scheme are all words, as are the one-letter word q and the empty word "". ${ }^{3}$ Words can be combined into larger words through the word procedure. For example, (word 'wo 'r 'd) produces the word word. Sentences are created through the sentence procedure, which can be abbreviated as se. For example, (sentence 'this 'is 'a 'sentence) produces (this is a sentence).

While different procedures are used to create words and sentences, the same procedures are used to take them apart. One can get the first letter (technically a one-letter word) of a word or the first word of a sentence with the first procedure. For example, (first 'this) is the word t , and (first '(this is a sentence)) is the word this. Similarly, the procedure last will return the last letter of a word or word of a sentence. One can discard the first letter of a word or word of a sentence with butfirst, also written as bf. For example, (butfirst 'this) is his, while (butfirst '(this is a sentence) is (is a sentence). The butlast procedure discards the last letter of a word or word of a sentence. For example, (butlast 'this) is thi, while (butlast '(this is a sentence)) is (this

[^2]is a). Taking the first or butfirst of an empty word or sentence causes Scheme to produce an error message, since there is no element to take or discard. Taking the last or butlast of an empty word or sentence causes an error message for the same reason.

### 1.4 Classroom Observations

As a teaching assistant (TA), I noticed certain common mistakes related to words and sentences. These mistakes were not limited to struggling students. For example:

- One of the early assignments in CS 3 is to write a program that converts a measurement in feet and inches, represented as a sentence of two numbers, to a measurement in inches. The solution is to take the first number out of the sentence, multiply it by twelve, and add it to the second number, as in this fragment of code: (+ (* (first measurement) 12) (first (butfirst measurement))). Most students make at least one of two mistakes. One is that they write the program to work with two numbers that are not in a sentence. The other is that they only take the butfirst of the measurement, which gives them a sentence containing the second number instead of the second number itself. In my experience, the first mistake is easily corrected by explaining to the student that the Scheme program will see the sentence rather than the individual numbers, unless they specifically instruct it to look inside the sentence. Students almost never repeat this mistake. The second mistake is not so easy to correct, and students repeat it throughout the semester. One of the best students in one of my classes made this mistake, and when I asked him why, he explained that (butfirst ' $(12)$ ) should produce the number 2 instead of the sentence (2), because a one-word sentence is useless.
- Students spent several days writing a program that spelled out numbers. For example, if the program were given 1000000025 , it would produce the sentence (one billion twenty five). The suggested solution was to break up the number into groups
of three digits and process each group separately. Many students' programs inserted empty words into the sentence for every group of three zeros: when given 1000000025, they produced (one billion "" "" twenty five). A few students noticed this and corrected the problem. Some never noticed. Many asked if this was acceptable, which was a reasonable thing to do. However, they were surprised when they were told it needed to be fixed. These students argued that their code was correct because the empty words were not really anything.
- Students were asked to write a procedure to find the largest number in a sentence. Many wrote code that returned this number in a sentence, and they seemed to think it was unreasonable for the TA to ask them to fix this.
- Students were asked to debug a procedure that was intended to find the longest word in a sentence. This procedure had two bugs. The first was an obvious mistake in how the code dealt with sentences (it had two calls to butfirst when it should only have had one), while the second was that it returned a sentence containing the longest word rather than the word itself. The students posted their corrected versions and comments in an online forum. Of the sixteen students who answered this question, all corrected the first problem and none corrected the second. Six students even posted examples that showed the word in a sentence. The next activity asked them to post the kinds of values that the code should return. Nine of the sixteen students, including four who had posted examples of the code producing sentences, said that it returned words. The other seven correctly noticed that the program produced sentences.
- On one exam, students were shown a line of Scheme code that produced a sentence. They were asked what each part of that line did and then what the end result would be. One of the parts produced an empty word, and while most students realized it would do so, many nevertheless omitted the empty word from the sentence.
- Students who added error-checking code to their programs often returned empty words and sentences instead of error messages. For example, on a timed midterm, several
students wrote procedures that required a sentence with at least two words, produced a number if they were given such a sentence, and returned () or "" if they were not. Students were not required to write their code to check for errors like this.

These mistakes, as well as others I have seen, can be described by four general observations. First, students did not seem concerned when procedures that should have returned words returned one-word sentences, or vice versa. Second, students treated empty words (words with no letters, written "") and empty sentences (sentences with no words, written ()) as things that were not important and could be ignored. Third, students acted as though one-word sentences did not exist, and that taking the butfirst of a two-word sentence should return the second word by itself instead of in a sentence. Finally, some students appeared to treat empty words and sentences as ways to say "error."

I initially suspected that students were making two fundamental mistakes. First, they had assumed that empty words and sentences, as well as one-word sentences, had no purpose in Scheme. Second, they assumed that the presence or absence of parentheses was more of a cosmetic issue than a programming issue. Just as a student writing a "Hello, World!" program might consider "hello world" to be close enough, CS 3 students considered hello and (hello) or (one billion seventeen) and (one billion "" "" seventeen) to be close enough. I had no good explanation for why some students considered empty things to mean "error."

### 1.5 Motivation

These mistakes may seem small and easy to fix. All teachers need to do is to remind their students that sentences need parentheses and that the butfirst of a sentence always returns a sentence. Why are they worth studying? While these mistakes may seem minor, they are certainly not easy to fix. Several lab activities ask students to differentiate between
words and one-word sentences, and yet many students forget this as soon as they finish one of the problems. Students often continued to make these mistakes throughout the semester.

There are two major problems with students holding these misconceptions throughout CS 3. First, these mistakes keep students from really understanding how their programs work. Often, the best way to understand a programming concept is to trace through a sample program line by line. Tracing, however, requires students to know exactly what each bit of the code does. Students who do not know how butfirst works can often write procedures that use it correctly, but they will become confused if they attempt to examine a program in detail.

Second, students can develop a false sense of programming itself. One of the hard parts about learning a new language is understanding that the way you express a concept in your language is not always the way other people express it in their languages. For example, a new speaker of Mandarin might want to say, "There is no spoon," but will have to say "I don't have a spoon," or "It's not a spoon." It is important for students to leave CS 3 with an understanding of when English is useful in programming and when it is not.

### 1.6 Goals and Design

Most data came from one-on-one interviews with students. During these, students thought aloud as they answered a set of written Scheme problems and then explained their reasoning to an interviewer. There were several follow-up questions prepared in case the students gave expected responses, but interviewers also questioned students spontaneously if they said anything unanticipated and interesting. Once students had learned lists, interviewers gave some students sentence questions and some students list questions. The list questions were exactly the same as the sentence questions, except that all references to sentences had been replaced with references to lists. Although students had been working with sentences far longer than with lists, they did significantly better on the list questions.

## Chapter 2

## Theory

### 2.1 Misconceptions

Students make mistakes while learning new material. When this happens, it is a teacher's job to understand why. Was it a silly mistake, like dropping the negative on a number? Was it the result of an educated (or wild) guess? Perhaps it was the result of an otherwise good idea taken a little too far, or solid reasoning based on a mistaken premise. Mistakes in this last category are called misconceptions, alternative conceptions, naive beliefs, and many other things. The most common term is "misconceptions," and although Smith, diSessa, and Roschelle (1993) give a convincing argument that "misconceptions" are important to learning and should thus be called something else, that term will be used throughout this work.

While misconceptions have only been studied since the 1970s, the notion of looking at students' ideas and trying to understand them, rather than simply counting them right or wrong, can be traced back to Piaget's work in the 1920s (for example, 1926/1972). Piaget showed that while children do not think about the world in the same way as adults, they

| Logo | BASIC |
| :--- | :--- |
| TO main | 10 gosub 100 :rem intro |
| intro | 20 gosub 200 :rem init |
| init | 30 gosub 300 :rem play |
| play | 40 gosub $400:$ rem check |
| check | 50 IF answer $\$=" Y "$ goto 10 |
| IF :answer $=" y$ [intro] | 60 END |
| END |  |

Figure 2.1: Incorrect Logo and correct BASIC for the same problem.
do work hard to make sense of their world. They may give some very strange answers to an interviewer's questions, but they arrived at those answers after some actual thought.

Misconceptions have been cataloged in many fields of study, including meteorology (Stevens, Collins, \& Goldin, 1979), chemistry (Schmidt, 1997), second language learning (Jarvis \& Odlin, 2000), physics (Gentner \& Gentner, 1983; McCloskey, 1983), mathematics (Stacey \& MacGregor, 1993; Fischbein \& Baltsan, 1998-1999), and programming (Pea, 1986; Taylor, 1990). There are a number of possible sources of misconceptions; the remainder of this chapter describes some of them.

## Prior Knowledge

Some misconceptions appear to come from students improperly applying knowledge they already have. Research on students learning to program has focused mostly on the effects of prior programming languages, although research in other fields has often examined the effects of more general real-world knowledge.

Lee and Lehrer (1988) studied Logo programmers and found that students with prior programming experience in BASIC often mistakenly applied BASIC techniques to Logo. For example, they tried to apply the syntax of a goto loop when asked to write a recursive procedure, as seen in Figure 2.1. The IF line in the Logo program is a student's attempt to start the loop over again at intro, but instead, it calls intro again and then stops.

Scherz, Goldberg, and Fund (1990), studying novice Prolog programmers, also found that students who knew other languages tried to write Prolog as though it was another language. For example, one student they described as "bright" wrote ancestor (X,Y) :parent (X, ancestor (Z, Y) ) instead of ancestor (X,Y) :- parent (X, Z), ancestor (Z, Y). This concept, composition of functions, works quite well in languages like Scheme, C, and BASIC, but it is not used in Prolog.

Scholtz and Wiedenbeck (1993) studied students who had already learned to program in Pascal and were in the process of learning Ada or Icon. Students sometimes tried to apply a plan (a series of steps to achieve one particular goal) that would have worked in Pascal but not in the new language. In addition, when students found a plan that would work in both languages, they often tried to apply it in a Pascal-like way that did not work in the new language. However, Wiedenbeck and Scholtz (1996) performed a longer study in which students who had several years of experience with Pascal learned Icon. The students in their study overcame problems with plans over the course of a semester, but they had trouble adapting to Icon's mechanisms for control flow.

Kolikant (2005) found that students sometimes counted a program as correct even if it printed extra (or sometimes incorrect) information and hypothesized that this was due to students' experiences with buggy professional software. Kolikant gave students descriptions of two programs that printed incorrect information. One was described as a complicated program that displayed a lot of information, while the other was described as a program to analyze the student's family tree and produce a list of all the student's cousins. The description of the first program also said that it printed one extra bit of information that it should not have, while the description of the second said that it also printed one uncle. Students were more likely to judge the second, more personally relevant, program as incorrect, even though it had almost exactly the same problem as the first program. Kolikant also found that students were willing to count an incorrect program as "relatively correct." Students admitted that a program had problems but decided that it still worked. College
students were more likely to judge incorrect programs as correct or relatively correct than were high school students. Presumably, the college students had more exposure to defective commercial software and knew that developing bug-free software was extremely difficult.

## Analogical Reasoning

An analogy can be defined as an "inference that if two things agree in certain respects then they probably agree in others" (Gentner, 1999, p. 17). Abuse of analogies when reasoning is a special case of the improper application of prior or real-world knowledge.

Stevens et al. (1979) observed that some mistakes made by their meteorology students were cases of improper applications of analogies. For example, some students compared clouds to sponges. Both are fluffy, both hold water, and both can release it. However, some students assumed that increasing pressure on a cloud would cause rain, since increasing pressure on a sponge would cause it to release the water it had absorbed.

Gentner and Gentner (1983) found that physics students made predictable mistakes depending on what analogy they used to make sense of circuit diagrams. They found that students spontaneously applied one of two different analogies. One compared the flow of electricity through wires to the flow of water through pipes. Batteries were like pumps, and resistors were very narrow pipes. Students using this analogy accurately predicted the behavior of two batteries in parallel (one pump on top of another) and in series (two pumps next to each other), but they had trouble predicting the behavior of two resistors (narrow pipes) in parallel or in series. The other analogy compared the flow of electricity through wires to the flow of people moving through hallways. Resistors were like narrow gateways, but batteries did not have an obvious equivalent. Thus, students using this analogy did a good job of predicting the results of placing two resistors in series or parallel, but they had trouble with parallel or series batteries.

When Gentner and Gentner tried teaching students using these two analogies, they got an unexpected result-students using the flow-of-people model did quite well on both resistors and batteries. The authors were unable to provide a satisfactory explanation for this. However, one major difference between this experiment and the previous one was that, when they taught the students the crowd model, they compared the batteries to loudspeakers shouting encouragement to the people walking the track. Two loudspeakers right next to each other would probably not be any more effective than one, but arranging them in series would mean that the walkers would hear encouragement for a longer time, and thus feel more motivated to keep on walking. Expanding the analogy to provide a better model for batteries significantly improved the students' abilities to answer questions.

## Natural Language

Many technical words that are used in the sciences also have meanings in everyday speech. While a student in a mathematical physics class might be unlikely to think a "Poincaré-invariant generator" is used to produce electricity, what would a student in a chemistry class think of a "normal salt," a category which includes things very different from ordinary table salt? An examination of the treatment of Newton's laws in five popular introductory physics textbooks reveals that the laws are usually defined using terms with both technical and (often multiple) everyday meanings (H. T. Williams, 1999). In one particularly egregious case, a textbook author claimed that nearly all of the terms used up to that point, including force, had the same meanings in physics as they had in everyday life.

It is possible that students with good everyday vocabularies may have more trouble with such terms than other students (Ryan, 1985a). One study that supports this idea was carried out on between 4,300 and 7,500 German senior high school students (Schmidt, 1997). These students were given a series of multiple-choice chemistry questions whose distractors were taken from the most common mistakes in a pilot study. Each student involved in the
experiment was given a random selection of six of one hundred twenty possible questions. In addition, several groups of students were videotaped while discussing some of these problems. In one case, 167 students answered questions that dealt with the definition of a redox reaction; $38 \%$ chose only the reactions that involved oxygen. One student explained that a redox reaction is a reduction and an oxidation, and "oxi" implies oxygen. In this case, good language skills led the student to the wrong conclusion. Similarly, students asked about acids and bases used the word neutralization, which was not mention in the question, to justify saying that the product would have a neutral pH even when the stoichiometry said it would not.

Spohrer and Soloway (1986) and Bonar and Soloway (1989) saw students learning to program in Pascal use while as though they were writing in English. In English, "while" means "as long as X is true, keep doing Y " but in Pascal, it means "If X is true, do Y , check X again, do Y again, and so on." Students who assumed while had an English-like meaning in Pascal assumed that the program would stop as soon as X became false, and not wait until it was done doing Y.

Pea (1986) also studied computer science students and found that they treated if in the same way. In English, one might say, "If it is not raining, I'll go to the store to get butter and cheese." In this case, "if" means almost the same thing as "when," except "if" does not imply it is raining right now. In programming languages, however, if does not mean "when." If a computer were to say, "If it is not raining, I'll go to the store to get butter and cheese," it would look out the window, see that it was raining, and not go to the store.

Taylor (1990) observed that students reasoned about Prolog by using the rules for English, but they failed to translate their English conclusions back into Prolog. As a result, they ended up with incorrect Prolog code that could be read somewhat like English. For example, when trying to represent the sentence "Any of Misha's students who work hard will succeed," one student wrote succeed(student, misha) :- work_hard(student). This
could be read in English as "Student of Misha will succeed if student works hard," but in Prolog, it means very little.

Höök, Taylor, and Du Boulay (1990) showed that some students learning Prolog stop thinking in Prolog and start thinking in another domain of knowledge if the variable names in the program remind them of that other domain. For example, a student presented with a Prolog program to find even divisors of a number tried to reason about the code with his knowledge of math instead of reading through the program.

Scherz et al. (1990) found that students assumed that Prolog works like English. For example, they thought that Prolog could understand English words like "everyone." In this case, the student wanted to express "Jane likes everyone," but wrote likes(jane, everyone) instead of the correct code, likes(jane, X).

Scherz et al. also noticed that students could be confused when writing about things they are familiar with in Prolog. Their students wrote a predicate called father, which relates a father and a son. For example, father (abraham, isaac) tells Prolog that Abraham is Isaac's father. The correct way to ask whose father Abraham is would be something like ?-father (abraham, X), which asks the Prolog system to find a value for X. However, many students asked, ?-father(abraham). Unfortunately, as it is written, father must take two arguments, not one. Students did not make this mistake when presented with logically equivalent Prolog statements that had no connection to English. For example, when presented with the fact $\mathrm{a}(\mathrm{b}, \mathrm{c})$, they did not try ? $-\mathrm{a}(\mathrm{b})$ to find c .

Davis et al. (1993; 1995a, 1995b) speculate that students' beliefs about the rules of a programming language may come from their understanding of the rules of English grammar. Students who see the rules of a computer language as more guidelines than actual rules may do so because they see the rules of grammar as similarly flexible.

## Superbugs

Many small mistakes may be caused by a single deeper misconception. Pea (1986) studied computer science students and suggested that individual "superbugs" (high-level mistakes) could be responsible for many smaller mistakes. For example, the confusion over if, the common assumption that a program would do something "because it wants to," and the notion that a computer can fill in the details might all be caused by the confusion between human discourse and computer discourse.

Similarly, Van Someren (1990) studied students learning Prolog and found that a fairly small set of malrules could account for many common programming mistakes. In this study, students had a generally good understanding of the material but had one or two small misconceptions that caused their programming bugs. For example, both I and , are used to define a Prolog list. They have different properties, and students who assume they are interchangeable can make several different mistakes when trying to build or take apart lists or to understand code that deals with lists.

Because many different mistakes can be caused by a few deeper issues, it is likely that many misconceptions can be treated by fixing one or two underlying problems.

## The Collection Model

Linchevski and Vinner (1988) identified five common misconceptions about set theory among elementary school teachers. They found that many teachers believed

1. the elements of a set must all share some common property
2. sets must contain more than one element
3. duplicate elements are distinct elements
4. an element of a set cannot also be an element of another set
5. two sets are equal if they contain the same number of elements

Fischbein and Baltsan (1998-1999) propose a single underlying model, the collection model, that unites Linchevski and Vinner's apparently diverse misconceptions into one basic misunderstanding. The "collection model" is simply an everyday notion of a realworld collection. For example, a collection is made up of multiple items that share a common property, and it may contain duplicate items.

One aspect of the collection model is of particular use when thinking about students' understanding of words and sentences. Viewing a sentence as a collection makes the idea of one-word or empty sentences seem absurd. Imagine that a friend asks what you think of his/her record collection while pointing to a shelf that contains a single record. ${ }^{1}$

One aspect of real-world collections that is not described by Linchevski and Vinner or Fischbein and Baltsan is the status of a collection as a thing. Collections seem to switch between being things unto themselves and merely being composed of things. A collection might have a value that is greater than the sum of the values of the objects in it, as in the case of a complete set of state quarters. It may even have its own name and a history that extends beyond the objects it contains. A sports team, for example, is a collection of individual players, but it has a life independent of some or all of its current members. On the other hand, it is quite possible for the objects in a collection to be far more important than the collection itself. People often treat collections of rare art and artifacts in museums this way. Most visitors to the Metropolitan Museum of Art are there primarily to appreciate the paintings; while they may be impressed that so many works of art are together in one place, that is probably not foremost on their minds.

Collections can switch from being things to being collections of things quite easily, depending on the circumstances. A collection of all baseball cards printed in the year 1977 is worth far more than the individual cards. If that collection is lost, however, it would

[^3]not be unreasonable to miss the individual cards. Similarly, when asked to describe one's record collection, one might describe a record collection as large, eclectic, or thorough; one might also describe it by listing individual albums.

## Instruction

Some people believe that when students are given the right concepts, they will abandon their misconceptions. Others believe that misconceptions should be actively challenged; when students see that their ideas do not work, these people say, they will gladly abandon them in favor of the right ones. McCloskey (1983), for example, suggests that teachers discuss misconceptions with their students and carefully explain what is wrong with each misconception. Similarly, Davis et al. (1993) identified common mistakes with quotes and parentheses in Common Lisp, a close relative of Scheme. They found that when students were asked to think about why these mistakes were wrong, those mistakes became much less common.

However, Collins and Gentner (1987), M. D. Williams, Hollan, and Stevens (1983), and others have shown that students are quite capable of using multiple lines of reasoning that can lead to very different answers. McCloskey (1983) found that students distorted information provided in the classroom to make it work with their misconceptions. Piaget (1926/1972) found that young children did the same, taking bits of what they had heard in class and inserting them into their understanding of the world.

Taylor (1990) and Smith et al. (1993) suggest that misconceptions are so durable because they work, at least in some cases. In Taylor's case, reasoning about programs in a natural language sense can be useful - it can help students think in a more abstract and comfortable way, as long as they remember to translate their natural language results into the programming world. In fact, Hoadley, Linn, Mann, and Clancy (1996) found that students were more likely to reuse code they had written if they could think about it in

English. Furthermore, while students are still learning to reason in a programming sense, natural language may be their only reliable way to think about a program.

Smith et al. point out that many misconceptions are actually good ideas that are used where they should not be. One example they give is the "Denominator Principle" for comparing fractions. According to this strategy, if both fractions have the same numerator, the one with the smaller denominator is larger. Both novices and experts might use a similar trick when the numerators are close but not the same, but experts have a better sense of what "close" means. Similarly, an expert programmer learning a new language might consider how a given problem could be solved in a more familiar language, just as Lee and Lehrer (1988)'s students did. However, the expert would be more careful when applying the results to the new language.

Thus, many misconceptions are only bad when they are misapplied. When used properly, they can be very convenient. Telling students that their ideas are wrong will not help, since the students can think of many instances when their ideas worked perfectly and may even have been endorsed by the teacher (Smith et al., 1993).

One way to deal with these misconceptions, which are perhaps better called misapplications, is to help the students see how and when they should be applied, and to help them see what to do when they can not do things the way they want. Smith et al. call this "knowledge refinement," and they believe the way to achieve it is to encourage classroom discussion without confrontation.

Linn and Eylon (in press) describe "knowledge integration," a perspective that encourages students to take advantage of their wide range of ideas. They give four steps to promote knowledge integration. First, instruction must elicit the ideas that students already have. Second, instruction should introduce new ideas. Third, instruction must help students develop criteria for evaluating ideas. Fourth, instruction should help students use their criteria to evaluate their ideas. Students should compare their original ideas with the
new ones provided in class, see which make sense, try to resolve contradicting ideas, and identify things they do not understand.

### 2.2 Analogy, Similarity, and Structure Mapping

An analogy is a comparison between two systems that have similar relations among their constituent parts, although their parts may not share similar attributes. For example, I might say, "an atom is like the solar system," meaning that the sun or nucleus attracts the smaller planets or electrons, which orbit the sun or nucleus. I would not mean that the nucleus is the size or color of the sun. In contrast, similarity is a comparison between two systems whose parts have comparable relations and attributes. If a NASA press release said that a newly discovered solar system was like ours, we could reasonably assume that in addition to the facts that its sun is larger than its planets and that the planets orbit the sun, we might also assume that the other sun is yellow and of about average size. A mere-appearance match is a comparison between two systems whose elements have similar attributes but not relations. If I compared a beach ball and the sun, all I might mean was that the ball was round and possibly bright yellow.

## How Analogies Work

## Structure Mapping

Gentner (1998) gives five steps in the use of analogies. First, a familiar situation, the base, is retrieved from long-term memory. Second, the base is mapped onto the new situation, the target. Third, the analogy and the inferences it offers are evaluated. Fourth, the structure common to both analogies is abstracted. Fifth, one or both of the representations are adapted to improve the analogical match. Gentner points out that the last two stages may or may not occur in a given instance. The existence, although not the details, of the first four steps are accepted by most researchers in the field (Holyoak \& Hummel, 2001).

The second step of this process, mapping, is the most important one, at least for the remainder of this work. According to Gentner (Gentner, 1983; Medin, Goldstone, \& Gentner, 1993; Gentner \& Markham, 1997), mapping consists of two steps, structural alignment and inference projection. Structural alignment is the process of finding the best set of correspondences between the features of two structured representations, while inference projection is the creation of a series of inferences about the target from what is known about the base. For example, connecting an atomic nucleus to a sun and electrons to orbiting planets would be a part of structural alignment, while suspecting that the electrons might be attracted to the nucleus just like planets are attracted to the sun would be a part of inference projection.

Gentner and her associates give three characteristics of structural alignment. The first characteristic, and the key to the successful alignment of two systems, is structural consistency. There must be a one-to-one correspondence between objects in the base and objects in the target. Further, corresponding relations must have corresponding arguments (parallel connectivity).

The second characteristic is relational focus. That is, analogies focus on shared relations (function) and not on shared attributes (form).

The third characteristic is systematicity. When connecting relations in the base with relations in the target, people tend to favor interconnected sets of relations rather than isolated relations. Furthermore, when many of the relations in a set have been mapped from target to base, people are quite likely to map the rest. For example, people comparing a solar system an an atom are likely to know the following about a solar system: ATTRACTS(sun, planets), ORBIT(planets, sun), CAUSE(ATTRACTS(sun, planets), ORBIT(planets, sun)), and HOTTER_THAN(sun, planets). Because CAUSE connects ATTRACTS and ORBIT, people who know nothing about an atom are more likely to map these than the isolated fact HOTTER_THAN. Further, people who know that the nucleus of an atom attracts electrons are much more likely to also map the ORBIT and CAUSE relations.

Gentner calls this process structure mapping, and has provided evidence that it also applies to cases of similarity. However, similarity comparisons do not have a relational focus. Instead, most or all features of the target are mapped onto the base.

Gentner and Toupin (1986) have shown that transparency, the obvious similarity between surface features of the base and the target, can have an influence on the ease of forming an analogy. If systematicity is high (the target and base share many causal features), mapping will be relatively easy. If systematicity is low, transparency has a strong influence on the ease of mapping.

## Symbolic Connectionism and Multiconstraint Theory

Holyoak and associates have proposed an alternative to structure mapping called symbolic connectionism (Hummel \& Holyoak, 1997; Holyoak \& Hummel, 2001). This process, which relies on a neural network, is more emergent than structure mapping. Symbolic connectionism is based in Holyoak's multiconstraint theory (Holyoak \& Thagard, 1997), which proposes three general kinds of constraints on mapping. These are similarity, shared relations between the source and target; structure, consistency and one-to-one correspondence between relations of the target and source; and purpose, what the reasoner wishes to achieve with the analogy. Similarity and structure have similar meanings in multiconstraint theory and structure mapping, but purpose plays little role in structure mapping. These three constraints are not absolute, as they are in structure mapping. Instead, they are used to guide the mapping process. This means, for example, that a symbolic connectionist model such as LISA (Hummel \& Holyoak, 1997) would encourage, but not force, a one-to-one mapping. This appears to be consistent with human behavior, as Spellman and Holyoak (1996) found that people sometimes map multiple elements of the source to one element of the target (Spellman \& Holyoak, 1996).

One prediction of the LISA model that is supported by Waltz, Lau, Grewal, and Holyoak (2000) is that working memory is a critical resource for mapping, and that an increase in
working-memory load makes people more likely to base their mappings on surface features rather than on relations. Tohill and Holyoak (2000) have found similar results when subjects' anxiety levels are increased.

## Using Analogies

There are several ways an analogy can provide new information. First, when some members in a set of connected relations are mapped from the base to the target, the remaining relations are often mapped as well. Thus, an analogy automatically provides some additional information with which to understand the target.

Analogies can be used to generate additional information once the structure mapping is complete. To make a prediction about the target, one makes a similar prediction about the base, confirms that the prediction can be mapped to the target, and performs the mapping.

Not all analogies can be used in this way, however. Some people switch between multiple unconnected analogies in what Collins and Gentner (1987) call a pastiche model. For these people, many bases are mapped to a single target, but each base maps to only a very tiny portion of the target. The individual analogies are thus useless in predicting the behavior of the target. In addition, sometimes someone may use an analogy to explain a concept to another person without actually using the analogy internally. A physics professor, for example, might compare electricity in wires to water in pipes to convey certain information to students, but would never think about water when designing a circuit at home.

Generative analogies are those analogies that are, in fact, used to create new knowledge and make predictions. Gentner and Gentner (1983) show that water through pipes and people on a track can be generative analogies in the domain of electrical circuits. If a person switches from one generative analogy to another, that person can arrive at very different conclusions (see page 13).

Structural alignment, or mapping in general, can also be used to compare and contrast two systems. Gentner and Markham (1994) found that people found it easier to list differences between two alignable objects than between two non-alignable objects. It is easier, for example, to list differences between a mongoose and a weasel than between a mongoose and a monsoon.

## Checking and Improving Analogies

An analogy is a system for generating hypotheses about the target; the hypotheses must be verified by some other means. Systematicity may provide a limited self-test for a newly formed analogy (Gentner, 1983). Because groups of connected relations tend to be mapped, one can detect and eliminate mapping errors by making sure that these collections are internally consistent. However, Holyoak and Hummel (2001) argue that systematicity is a guideline, not an inviolable rule, meaning that it may not act as a reliable self-test.

The quality of an analogy can be improved through the final two steps described on page 21. However, before people can abstract the common structure of the base and target and refine the two to fit more closely together, they must think about the nature of the target, base, and the analogy that connects them. As was the case with correcting misconceptions, people need to examine their ideas and refine their knowledge.

## Chapter 3

## Materials and Methods

### 3.1 Population

All students in the Spring A, Summer A, Fall A, and Spring B sections of Berkeley's CS 3 were required to attend a one-hour interview. Students were not paid for their participation, but they did fulfill a course requirement. Most students even found the interviews to be useful- they saw them as an hour of one-on-one tutoring. In several cases, students tried to sign up for multiple interviews. Between $10 \%$ and $20 \%$ of the students in each semester later dropped the course (see Table 3.1 on page 28), but their responses were not excluded because the interviews were anonymous.

Not all students in every semester were interviewed. Three students in the Spring A semester were given an initial version of the first survey; their answers are not included in this data. Five students in the final round of interviews in Spring A were interviewed by a new interviewer who did not sufficiently understand what to do. Their answers were not included in this data. In some cases, students were excused because an interviewer was unable to attend the interview. Several students felt very nervous when they were told that the interview would involve Scheme questions. When students looked nervous,
the interviewers asked if they wanted to skip the Scheme questions and answer a set of questions unrelated to Scheme. One student did this, two others were willing to answer the questions so long as the interviewer looked away, and the rest said they were still willing to answer the Scheme questions. Some students, especially in Spring B, only tried to sign up for interviews in the last few days before the end of the semester. Some never attempted to sign up for any interview. While attending an interview was officially a course requirement, students who did not attend were not penalized in any way.

Students were allowed to pick one of the three or four rounds of interviews to participate in. It is possible that this allowed for some self-selection effects, but I believe these would either have minimized the observed problems and differences between lists and sentences or have been counteracted by other forces. Students who took part in the last round of interviews were certainly the survivors of CS 3, and they might be expected to have a better understanding of the material. While some of this may have happened, many students involved in the last round were distinctly unmotivated students who had either waited until the last minute to participate, signed up for one (or even two) earlier interviews and forgot to show up, or were completely unaware that there were interviews at all until I asked them point blank in lab, "Have you been interviewed yet?" Many of these students were weeks behind.

Table 3.1 shows the number of students enrolled at the start of the class and at points very near every interview, as well as the number of students who took part in every interview (and percentage of the students enrolled at that point) and the total number of students who were interviewed. These numbers are approximate. The number of student accounts created in the UC-WISE course portal was used to determine the number of students enrolled at the beginning of the semester. Because interviews took place one week before a midterm, the number of students who took each midterm was used to estimate the number of students enrolled when that round of interviews occurred. The number of students who took each final exam was used to estimate the number of students enrolled for the last round

|  | Spring A | Summer A | Fall A | Spring B |
| :--- | :--- | :--- | :--- | :--- |
| Initially Enrolled | 167 | 51 | 224 | 117 |
| Enrolled by I1 | 129 | 36 | 186 | 89 |
| Interviewed in I1 | $37(29 \%)$ | $12(33 \%)$ | $37(20 \%)$ | $19(21 \%)$ |
| Enrolled by I2 | 124 | 35 | 176 | 78 |
| Interviewed in I2 | $34(27 \%)$ | $10(29 \%)$ | $34(19 \%)$ | $20(26 \%)$ |
| Enrolled by I3 | - | - | 169 | 77 |
| Interviewed in I3 | - | - | $34(20 \%)$ | $13(17 \%)$ |
| Enrolled by I4 | 111 | 32 | 168 | 72 |
| interviewed in I4 | $34(31 \%)$ | $11(34 \%)$ | $52(31 \%)$ | $17(24 \%)$ |
| Total Interviewed | 105 | 32 | 157 | 69 |
| \% Dropped After I1 | $14 \%$ | $11 \%$ | $16 \%$ | $20 \%$ |

Table 3.1: Number of students enrolled in CS 3 at the time of each interview and the number (and percent of enrolled students) interviewed in each interview. (I1 is Interview 1, etc.)
of interviews each semester. Spring and Summer A had only three rounds of interviews; the information for the final round of interviews in those two semesters are categorized as Interview 4 rather than Interview 3, since they were the final rounds of interviews in those semesters. The third midterm in the Fall A semester was not graded or recorded in any way, so the number of students who completed homework or quizzes assigned at the time the midterm was given out was used. In addition, there was no midterm after the recursion section of the Spring B semester. The number of students who completed homework or quizzes at the start of the week when this midterm would otherwise have been given was used to estimate the number of students enrolled at this point. Midterms rather than quizzes were used whenever possible, for several reasons. First, many students were behind on quizzes and homework, and some may have dropped after a given week of interviews but before they completed the appropriate homework or quizzes. Second, some students never completed a given homework or quiz even though they were enrolled in the class. Far fewer students missed a midterm or final than missed a given group of homework and quizzes. Third, the numbers for the exams were already calculated by the instructors and reported on the course website.

| Exam |  | Spring A | Summer A | Fall A | Spring B |
| :--- | ---: | :--- | :--- | :--- | :--- |
| Midterm 1 Mean | $14 / 20(70 \%)$ | $34.9 / 40(87.3 \%)$ | $20.8 / 25(83.2 \%)$ | $22.8 / 30(76 \%)$ |  |
|  | Stdev | 4.7 | 5.3 | 4.3 | 4.9 |
| Midterm 2 Mean | $39.8 / 50(79.6 \%)$ | $34.7 / 40(86.8 \%)$ | $39.7 / 50(79.4 \%)$ | $22.4 / 30(74.7 \%)$ |  |
| Midterm 3 Mean | $29.3 / 50(58.6 \%)$ | 5.4 | 10 | 5.1 |  |
| Stdev | $8.7 .1 / 40(77.8 \%)$ | $-/-$ | $-/-$ |  |  |
| Stdev | 12.1 | 6.2 | - | - |  |
| Final Mean | $70.4 / 120(58.7 \%)$ | $41.1 / 50(82.2 \%)$ | $51 / 75(68 \%)$ | $32 / 60(53.3 \%)$ |  |
| Stdev | 24.4 | 8.9 | 17 | 11.8 |  |

Table 3.2: Mean exam score/total points (and standard deviation) for each semester.

### 3.2 Differences Among Semesters

Many things changed from one semester of CS 3 to the next. There were three instructors in four semesters. Individual topics were emphasized more one semester than another, even under the same instructor. Each semester had a unique mix of students, as well. These differences are described below and summarized in Table 3.2 on page 29 and Tables 3.3, $3.4,3.5$, and 3.6 on pages 39-41.

Exam scores in Spring A were generally lower than those in other semesters. The only exceptions were for Midterm 2, where the average was almost exactly the same as that in Fall A and slightly higher than that in Spring B. Exam scores for Spring B were lower than those in Summer or Fall A. Summer A had the highest exam scores of all. In all cases except for the Midterm 2 in Spring and Fall A, the differences were significant at the $p<.05$ level, and most were significant at the $p<.01$ level. If exam scores are correlated with students' answers in interviews, students in Spring B should not do as well as those in Fall or Summer A. Scores are given in Table 3.2.

## Common Elements

Each CS 3 class, except for Summer A, took place over a regular semester. Classes filled the first 15 weeks, while the last week was reserved for final exams. In summer, CS 3 was taught in only eight weeks, with only the last day reserved for a final exam. All semesters since at least Fall 2000 included three or four exams. The two or three exams that occurred
during the first fifteen weeks of the semester were usually called midterm exams, although they were not all held near the middle of the term.

The first four or five weeks of the spring and fall semesters were spent introducing students to Scheme. A midterm was often given at the end of this period. After basic Scheme, students studied recursion for about four weeks, and then another midterm was often given. After recursion, students spent several weeks on higher-order procedures. After higher-order procedures, students were introduced to lists and, briefly, any other topics the instructor considered important. There was often a midterm at the end of higher-order procedures; it may or may not have included lists. Final projects occupied students for the rest of the semester. The final exams were comprehensive. Table 3.6 gives a more detailed breakdown of the topics in each of the relevant semesters.

In previous semesters, students attended two hours of lecture, two hours of lab, and one hour of discussion each week. Starting in Summer 2002, and in all of the semesters described in this thesis (except Summer A), students attended one hour of lecture and either five hours of lab and one hour of discussion or six hours of lab each week. Because summer classes take only eight weeks, students spent fourteen hours in lab every week. With these changes, lab became the place to learn new material; relatively few students attended lecture because they felt it offered them nothing new. Lab activities were available on the UC-WISE web portal. Students worked through material at their own pace, while the TAs and lab assistants were there to offer help when students had trouble. Some TAs or lab assistants used the system to monitor students and identify those who were having trouble, but most either dealt with students only when students asked for help or circled the lab and actively checked up on students. When TAs noticed that many students were having similar problems, or when they felt there was something important to say, they would stop the class and conduct a discussion. For further details on the UC-WISE system, see Clancy, Titterton, Ryan, Slotta, and Linn (2003).

## Spring A

The Spring A semester was taught by Instructor A, a lecturer with several years of experience with CS 3. He emphasized the small details of Scheme more than the other teachers (e.g., "What do you get if you multiply zero numbers together?"); many of his students said they spent hours typing unusual things into the Scheme interpreter to prepare for exams. He gave three midterms during the semester. The first covered basic Scheme, the second covered recursion, and the third covered higher-order procedures and lists. This encouraged students to study lists, so they should have been reasonably familiar with them by the time the third round of interviews took place.

This was the first regular (non-summer) semester in which the entire CS 3 class used the UC-WISE course management system, and it was the instructor's first time to teach using these specific activities and in a lab-heavy manner.

Spring A had a wide range of students. Ten percent were Electrical Engineering and Computer Science (EECS) students, a total of nineteen percent were engineering students, and the rest came from a variety of other majors. Some mentioned during the interviews or in lab that they were taking the class just to see what computer science was like. These students appeared willing to participate in the interviews, possibly because they had been told that if they did not participate, they would not get their points for class participation (two percent of their total grade). In any case, students signed up for interview slots almost as soon as they had the chance. Sign-up sheets were posted in the labs one week before the start of interviews, and almost all slots were full by the end of the week. Some students in every round of interviews would forget to show up, but most of these signed up for and actually participated in the next round.

## Summer A

Instructor B taught this class. Instructor B had taught CS 3 several times before and had been a TA for the class for three years. He did not emphasize as many of the little details as Instructor A. He gave three midterms during the semester, and they covered the same material as those in the spring. However, Instructor B's third midterm did not include lists. The final project, on the other hand, did emphasize lists more than in the previous semester. The round of interviews that compared lists and sentences was conducted near the end of the final project, so students should have had some experience using lists.

Instructor B had worked with the UC-WISE system several times. In the summer, CS 3 runs twice as fast and fits into eight weeks. Instructor B did not give regular lectures or hold regular discussions. Instead, students were in lab three hours per day Monday through Thursday, with two extra hours on Friday that were generally for catching up. Lecture or discussion happened whenever the TA or instructor thought something was worth talking about, and was often in response to frequent mistakes made in lab.

The students were very diverse. Some were high school students and wanted to learn something new or get an advantage on their college applications. Some were from other universities and wanted to take classes at Berkeley. Others had just been admitted and wanted to start one class ahead. These students were so willing to participate in interviews that I did not need sign-up sheets. I simply asked for volunteers in each lab section, and within a few minutes I had more students than interviews.

In addition to differences in teacher and pacing, there were several potentially important changes to the class material. The first was a change in the way domain and range were taught. The domain of a procedure is the set of acceptable input values, while the range is the set of things that the procedure can produce given valid inputs. In Spring A, students were expected to read a mystery program and describe its domain and range, without understanding what the mystery program was intended to do. Thus, students took the
domain to mean whatever values their programs, even if buggy, accepted, rather than only the values that should have been accepted. Similarly, they took range to mean anything a procedure could possibly return without actually giving an error message, rather than the kinds of things a procedure should return when it is given correct arguments. As a result, students sometimes tried to justify defective code by saying that it matched the domain and range. This statement is true but worthless when domain and range are based on what the code happens to do rather than what it should do. Starting in Summer A, substantially less importance was placed on domain and range during tests, and students were no longer expected to work with arbitrary mystery procedures.

The second major change was the addition of a page to the second lab, followed by a discussion. Students had been given a very brief introduction to words and sentences on the first day, and by the time they reached the new activity, they had been given a formal introduction to words and sentences. The text to this activity can be found in Figure 3.1. In the last half of class, the instructor or TA in charge of that lab section ran a short discussion on this topic, gave a quick demonstration with a $\mathrm{Pez}^{\circledR}$ dispenser, and then gave the students candy.

The second activity was a homework assignment given at the end of the section on basic Scheme, but before the midterm review session. In an online discussion, students were asked to list ways in which English and Scheme were similar or different, and to comment on other students' ideas. The text of this assignment is given in Figure 3.2.

At the end of the week, the instructor and TA held a midterm review session. Close to half of this session focused on the details of Scheme words and sentences and why they are not like English. Similar activities had been done in discussion or lab sections in previous semesters, although not at formal midterm review sessions.

The recursion section ended with the number-spelling project, in which students were asked to take a number like 12345 and convert it into a sentence like (twelve thousand three hundred forty five). A common mistake in this project was the inclusion of

## An Analogy for Words

A sentence is a collection of words. A word is a collection of letters. Amazingly enough, your TA or instructor will now explain how words and sentences are like Pez candy dispensers. Here's a basic summary of the argument:

- The sentence or word itself is the dispenser
- Individual words in the sentence or letters in the word are like the candies.
- Individual candies are in a specific order within the dispenser, just like individual words or letters are in a specific order within the sentence or word.
- With a flip of your finger, you can separate the first candy (first) from the dispenser and all of the rest of the candies (butfirst). You can use those two procedures to separate the first word or letter from the rest of the sentence or word.
- As long as it is your Pez dispenser, it's OK to take the last candy out. It's also OK to take the first or butfirst of a one-word sentence or a one-letter word.
- People collect empty Pez dispensers. I promise I'm not making this up. If you don't believe me, check out the Burlingame Pez Museum. It's equally OK in Scheme to have an empty sentence (it looks like ()) or an empty word (it looks like " ").
- A Pez dispenser is only empty when it doesn't have any candy at all in it. You can't just say it's empty if the last thing in there is a candy you don't like. Likewise, you can't say a sentence is empty just because you don't like what it contains. If " " is still a word, ("") is not an empty sentence.

Don't go too crazy with the analogy, though. You can pop the top on a Pez dispenser even after it's empty, although you won't get any candy. If you try to take apart an empty word or sentence, Scheme throws a fit.

Figure 3.1: A new reading activity added to lab in Summer A and later semesters.
empty words in the sentence in place of a string of zeros: 1000000007 became (one billion "" "" seven) rather than one billion seven). Thus, many students should have been exposed to the idea of empty words in sentences.

There were several other changes that were unrelated to the interview questions. Specifically, the lab activities on let and local variables that were given early in the semester, as well as input/output, graphics, and fractals, which had been given near the end of the semester, were removed. These activities had no relationship to sentences or lists. Further-

## Comparing English and Scheme

Both Scheme and English deal with numbers, words, and sentences. Sometimes Scheme and English agree, but sometimes they do not. These differences can cause all kinds of trouble on exams.

List as many ways that Scheme and English words, sentences, and numbers are alike as you can think of. Then list as many ways that Scheme and English words, sentences, and numbers are different as you can think of. Then comment on what other people have said. Here are some things to consider:

1. If somebody says "No" in English, it doesn't really matter if it is the word "no" or the sentence "No." You get the idea. How about Scheme? Is (no) the same as no?
2. How are parentheses used in English? How about Scheme?
3. English doesn't even have empty sentences or words. What do you think () or " " are in Scheme?
4. Numbers and words are pretty different in English. For example, it's OK if you name a baby after a famous singer. It's not OK if you name a baby after a famous number ("This is 2.718281828 . Isn't she cute?"). What about in Scheme?

Figure 3.2: Homework activity given out in Summer A.
more, because input/output, graphics, and fractals were given while students were working final projects, very few of the Spring A students had seen them by the time they were interviewed.

## Fall A

This semester was again taught by Instructor A. He focused on small details, although less than he had before. He had three midterms that covered the same material as the three in the spring, except that lists were not included on the third midterm. The final project did use lists, but not extensively. However, the final round of interviews occurred near the end of the projects, and students should have had a reasonable amount of practice with lists.

This semester, the class involved one hour of lecture and about six hours of lab every week. Teaching Assistants were asked to spend about an hour of lab time every week giving a discussion, but in reality, discussions varied from about half an hour to an hour.

There were substantially more engineering students. Twenty-nine percent of the students were EECS students, and forty-two percent were engineers of some kind. Semesters with a larger proportion of engineers appear to have a lower dropout rate and may have higher class averages. This was the semester with the highest concentration of engineering students. However, these students did not seem motivated to participate in interviews. Sign-up sheets were still made available one week in advance, but they almost never filled up. In fact, a fair number of students did not look for an interview slot until the last day of the last round of interviews. Many students sent e-mails either on the last scheduled day or on the weekend after the last scheduled day saying that they had just noticed the interview sign-up sheet and couldn't find any open times. Because of this, the final round of interviews was extended by another week.

This semester inherited several features from the summer. First, it downplayed domain and range and did not include lab activities on variables early in the semester. Second, it used the $\mathrm{Pez}^{\circledR}$ example, although TAs did not pass out candy. Third, it had a homework activity that replaced the Summer A "Comparing English and Scheme" discussion. Fourth, it used the number-spelling project. The text of this replacement activity, an online discussion given at the start of the third week (near the end of basic Scheme), is shown in Figure 3.3.

Like Spring A, Fall A covered input/output, graphics, and fractals. The review session for the first midterm did not spend much time on words and sentences and how they differed from English.

## Comparing English and Scheme

Both English and Scheme have things called words and sentences. These are similar, but not identical. List at least two ways in which English words or sentences are like Scheme words or sentences and at least two ways in which they are not like Scheme words or sentences. Also, make at least one intelligent comment on a classmate's list.

Just so you know, things like "Yeah!" do not count as intelligent responses, no matter how much thought you put into them.

Figure 3.3: A homework activity give in Fall A and later semesters that replaced the one shown in Figure 3.2.

## Spring B

The Spring B semester was taught by Instructor C, who had many years of teaching experience and was very familiar with the UC-WISE version of CS 3 . He did not emphasize the same kinds of little details that Instructor A did. He also made some changes to the curriculum, removing unsuccessful activities and writing new ones. He gave two midterms during the semester. The first covered basic Scheme. The second covered both recursion and higher-order procedures, but not lists. The final projects, however, made extensive use of lists.

The organization of this class was similar to that of Fall A. The instructor gave one hour of lecture every week, and students spent six hours every week in lab. As in the Summer A semester, discussion was held whenever the TAs felt it was needed.

This was the semester with the fewest engineering students. Five percent were EECS, and a total of seven percent were engineers of any kind. Students signed up for most of the available interview times, although this often took more than one week.

There were a number of changes to the course materials in this semester. First, many of the activities in the early part of the semester were modified to use WebScheme (see p. 146). Thus, instead of students writing programs or answering questions and judging for themselves whether they were right or wrong, they entered their programs or answers into an interactive web page and were shown green check marks when they were right and red Xs when they were wrong. One of these activities was relevant and is shown in Figure 3.4.

Fill in the blanks below to get the right answer. To see if you are right, press the arrow. If you see a green check, you got it.

You will need to wait until you see the word "SchemeHandler" in the upper left corner of the page before you start.

| Scheme Expression | Value |  | Correct? |
| :---: | :---: | :---: | :---: |
| (sentence 'I '(me mine)) |  |  | (?) |
| (sentence '( ) '(is empty)) |  | $\left[\frac{2}{3}\right.$ | (3) |
| (word 'ab 'cd) |  | Tin | (? |
| (sentence 'ab 'cd) |  | Dis | (8) |
| (sentence 'a (word 'k 'c)) |  | T画 | (8) |

Figure 3.4: WebScheme activity on empty words and sentences. It was given in Spring B and later semesters.

## Defining Terms

Give good definitions for Scheme words and sentences. Make sure you mention how they are or are not like English words and sentences. Give this some thought. Once you submit it, you won't be able to go back and change what you wrote.

Figure 3.5: A new homework given in Spring B and later semesters.

Students who entered sentences without parentheses were presented with error messages. Another relevant WebScheme exercise introduced in this semester dealt with empty words and sentences. In the past, students had been asked to write down explanations for (butfirst '(x)) and (butfirst 'x), but they were not required to show their answers to a TA, and few TAs asked to see the explanations. The WebScheme activity asked students to type in what Scheme would produce for each of these expressions. Again, students who typed in incorrect answers were given red Xs or error messages.

The "Comparing English and Scheme" homework was assigned on the second day of lab, which was when students were introduced officially to words and sentences. Another homework activity was also assigned on that day. This new homework was not an online discussion, although it was an online activity. In this case, once students posted their answers, they were able to see answers submitted by other students in their lab section. The text of the new homework is given in Figure 3.5.

| Semester | Instructor | Lecture | Lab | Discussion |
| :--- | :--- | :--- | :--- | :--- |
| Spring A | Instructor A | 1 hour | 5 hours | 1 hour |
| Summer A | Instructor B | as needed | 14 hours | as needed |
| Fall A | Instructor A | 1 hour | $5-6$ hours | $0-1$ hours |
| Spring B | Instructor C | 1 hour | 6 hours | as needed |

Table 3.3: Instructor and number of hours per week of lecture, lab, and discussion for each semester.
\(\left.$$
\begin{array}{|l|l|l|l|l|}\hline & \text { Spring A } & \text { Summer A } & \text { Fall A } & \text { Spring B } \\
\hline \text { Instructor } & \text { Instructor A } & \text { Instructor B } & \text { Instructor A } & \text { Instructor C } \\
\hline \text { Students } & 19 \% \text { engineers } & \begin{array}{l}\text { mixed } \\
\text { eager } \\
\text { eager }\end{array} & 42 \% \text { engineers } & \begin{array}{l}7 \% \text { engineers } \\
29 \% \text { EECS } \\
\text { not eager }\end{array}\end{array}
$$ \begin{array}{l}5 \% EECS <br>

neutral\end{array}\right]\)| \# in First Interview | 129 | 36 | 186 |
| :--- | :--- | :--- | :--- |
| \# in Last Interview | 111 | 32 | 157 |
| Lists Tested by | midterm | final project | midterm |
| Other Material | small details |  | final project |
| Midterms | 3 | 3 | 3 small details |
| \# of Interviews | 3 | 3 | 4 |

Table 3.4: Instructors and students, motivation to learn lists, number of midterms, and number of interviews for each semester.

There were several other changes that probably had little to do with students' performance in the interviews. First, the lab activities for trees, graphics, fractals, and input/output were removed. Again, in previous semesters, these activities were given during the final project, and students generally did not work on them until after the final round of interviews. Second, the section on higher-order procedures ended with the election project, in which students wrote a procedure that calculated the winner of a presidential election when given one sentence describing the number of electoral votes per state and another describing the popular votes in each state. This project did not deal with empty words or sentences, one-word sentences, or anything else in the interviews. Third, after the election project, students worked on a pattern-matching program. This was a large recursive program that implemented a limited set of regular expressions. This had very little to do with any of the issues covered in the interviews, and because it was unpopular, most students ignored it until classes were over and they were studying for the final.

This information is summarized in Tables 3.3, 3.4, and 3.5.

| Semester | Relevant Changes from Previous Semester | Other Changes |
| :---: | :---: | :---: |
| Summer A | domain and range: minimized, students no longer expected to find domain and range of arbitrary procedures to keep them from misusing the concepts to justify broken code <br> $\mathbf{P e z}{ }^{\circledR}$ analogy: reading and discussion in lab when words and sentences were covered, complete with candy, to give students a better analogy for empty words and sentences <br> Comparing English and Scheme: online discussion for homework near the end of the basic Scheme section; students asked to compare and contrast English and Scheme sentences <br> first midterm review: heavy coverage of English and Scheme sentences number-spelling project: students convert a number into a sentence, many students had empty words in their sentences and thus should have known that empty words could exist in sentences | early lab on variables removed, along with labs on input/output, graphics, and fractals |
| Fall A | $\mathrm{Pez}^{\circledR}$ analogy: less emphasis and no candy Comparing English and Scheme: different online discussion for homework first midterm review: did not spend much time on English and Scheme sentences | did cover input/output, graphics, and fractals |
| Spring B | WebScheme activities: interactive lab activities given out on the day words and sentences were covered; confronted students who wrote sentences without parentheses or treated empty words and sentences as nothing <br> Comparing English and Scheme: assigned on the day words and sentences were covered <br> Defining Terms: online homework, asked students to define Scheme words and sentences, given on the day words and sentences were covered | lab activities for input/output, trees, graphics, and fractals removed more higher-order procedures practice with the elections miniproject <br> students worked with a pattern-matching program, which gave more practice with reading and working with recursive procedures |

[^4]Week

| 1 | introduction to Scheme, words \& sentences | introduction to Scheme; words \& sentences; conditional expressions; more words \& sentences | introduction to Scheme; words \& sentences | introduction to Scheme; words \& sentences |
| :---: | :---: | :---: | :---: | :---: |
| 2 | conditional expressions; more words \& sentences | Interview 1; "Difference Between Dates" case study | conditional expressions; more words \& sentences | conditional expressions; more words \& sentences |
| 3 | "Difference Between Dates" case study | Midterm 1; introduction to recursion; advanced recursion | "Difference Between Dates" case study; Interview 1 | conditionals; more words \& sentences; "Difference Between Dates" case study |
| 4 | Interview 1; variables; functions as data | Interview 2, "Roman Numerals" case study; more kinds of recursion; number-spelling project | Midterm 1; "Difference Between Dates" case study; introduction to recursion | "Difference Between Dates" case study |
| 5 | Midterm 1; introduction to recursion; more recursion | Midterm 2; introduction to higherorder procedures; higher order procedures \& lambda; bridge project | Interview 1; introduction to recursion | introduction to recursion; more recursion |
| 6 | advanced recursion; "Roman Numerals" case study | more higher-order procedures \& lambda; introduction to lists; start of final projects | advanced recursion; "Roman Numerals" case study | Midterm 1; more recursion; "Roman Numerals" case study |
| 7 | Interview 2; more kinds of recursion | Midterm 3; trees; final projects | Interview 2; number-spelling project; more kinds of recursion | advanced recursion; number-spelling project |
| 8 | Midterm 2; introduction to higherorder procedures | Interview 3; lists; deep recursion; trees; final projects; Final Exam | Midterm 2; number-spelling project; introduction to higher-order procedures | Interview 2; number-spelling project |
| 9 | higher-order procedures \& lambda |  | higher-order procedures \& lambda | introduction to higher-order procedures; lambda |
| 10 | introduction to lists |  | lambda; introduction to lists | election project |
| 11 | Midterm 3; input/output; final projects |  | Interview 3; input/output; final projects | Interview 3; election project; patternmatching |
| 12 | fractals \& graphics; advanced list processing; final projects |  | Midterm 3 (faux); advanced list processing; final projects | election project; Midterm 2; basic \& advanced lists |
| 13 | Interview 3; fractals; final projects |  | fractals; final projects | lists; final projects |
| 14 | trees; final projects |  | trees; final projects | final projects |
| 15 | deep recursion; trees; final projects |  | Interview 4; review of deep recursion; trees; final projects | Interview 4; final projects |

Table 3.6: Topics by week for each semester. Note that summer semesters have only eight weeks

### 3.3 Interviews

Spring and Summer A students were given a choice among three interviews throughout the semester, while Fall A and Spring B students were given a choice among four. The first round of interviews was always held the week before the first exam. At this point, students had learned only basic Scheme. They could manipulate words and sentences and write simple branching programs, but they could not do loops or recursion. The second round was always held the week before the second midterm, or the week after the end of recursion for Spring B (when there was no recursion midterm). Students had just learned recursion, so in addition to some of the basic questions from the first interview, they were asked about recursive procedures that contained similar bugs. The third interview for Fall A and Spring B was held the week before the third midterm (the second midterm for Spring B). This interview included the questions from the previous interview, along with a question about higher-order procedures. The final round of interviews was held in the last week of classes and, because many students forgot to show up for their assigned times, often stretched into the week after the end of classes. The questions on this interview were exactly the same as those on the interview before it.

Weekly schedules for all semesters can be found in Table 3.6. Copies of all interview forms can be found in Appendix A.

## Questions and Predicted Mistakes

## Hypotheses

The main hypothesis is that most of the mistakes with one-word sentences and empty words and sentences are caused by students misapplying their real-world knowledge to Scheme. In particular, students are either assuming that Scheme works the same way English does or that sentences work the same way real-world collections do.

| English Sentences | Scheme Sentences |
| :---: | :---: |
| Helpful |  |
| sentences | sentences |
| words | words |
| meaning to humans | meaning to Scheme procedures |
| sentences are composed of words | sentences are composed of words |
| words are composed of letters | words are composed of one-letter words |
| sentences have meaning | sentences have meaning |
| words have meaning | words have meaning |
| the order of words in a sentence is controlled by syntax | the order of words in a sentence is controlled by the requirements of the program |
| the meaning of a sentence comes from its words and their order | the meaning of a sentence comes from its words and their order |
| capital letters at the start of a sentence | ( |
| punctuation marks at the end of a sentence | ) |
| capital letters and punctuation marks delimit sentences | parentheses delimit sentences |

Harmful

| words are made of letters | words are made of letters |
| :--- | :--- |
| parentheses | parentheses |
| parentheses contain words | parentheses contain words |
| parentheses ignored when the words are read | parentheses ignored when the words are read |
| parentheses are optional and can be replaced <br> with commas or hyphens | parentheses are optional |
| one-word sentences are rarely correct in formal <br> English | one-word sentences are not correct |
| one-word sentences and words are pretty much <br> the same thing | one-word sentences and words are pretty much <br> the same thing |

Table 3.7: Possible Elements of a Mapping Between English and Scheme Sentences.

Tables 3.7 and 3.8 list possible elements of a mapping between English and Scheme (3.7) and collections and Scheme (3.8). The mapping from English to Scheme illustrates several problems. First, it provides a reason to assume that empty sentences have no meaning: if the meaning of a sentence comes from the words it contains and their order, an empty sentence must have no meaning. Second, it offers no insight at all about empty words. The mapping from collections to Scheme also illustrates potential problems. Because sentences and words operate in very similar ways, collections would have to be mapped to both of them. Because Gentner's model predicts that people seek one-to-one mappings, collections are unlikely to be mapped to both sentences and words. Other problems are listed in Table 3.9 on page 45 .

| Collections | Helpful |
| :--- | :--- |
| collections sentences <br> collections words <br> collections can contain subcollections sentences contain words <br> collections can contain subcollections words contain one-letter words |  |

Harmful

| collections can contain individual elements | words contain letters |
| :--- | :--- |
| a collection must contain more than one ele- <br> ment | a sentence must contain more than one word |
| elements of subcollections are really elements <br> of the main collection | letters are elements of sentences |
| sometimes a collection is a thing and some- <br> times it is a bunch of elements | sometimes a sentence is a thing and sometimes <br> it is a bunch of words |
| container | parentheses |
| container(s) for a collection can be ignored <br> when thinking about the collection | parentheses can be ignored when thinking <br> about the sentence |
| subcollections with no elements can usually be <br> removed | words with no elements can usually be re- <br> moved |

Table 3.8: Possible Elements of a Mapping Between Collections and Scheme Sentences.

## Questions About One-Word Sentences

1. What is (bf ' (1 2))? (question O1)
2. What is (bl '(1 2))? (question O2)
3. What is (bf '( $\left.\begin{array}{l}1 \\ 2\end{array} \quad 3 \quad 4\right)$ )? (question O3)
4. What is (bl ' (1 $\left.2 \mathrm{l}_{3} 34\right)$ )? (question O 4 )

There were four questions that dealt with creating one-word sentences. The butfirst or butlast of a two-word sentence should be a one-word sentence, while the butfirst or butlast of a four-word sentence should be a three-word sentence. The first and third problems, which dealt with butfirst, appeared on every interview. The other two, which dealt with butlast, appeared only on the first round of interviews every semester.

Students using analogies to English or the collection model should have made one of two characteristic mistakes. First, they might have believed that the butfirst or butlast of a two-word sentence was a word. With the exception of commands, one-word sentences

| Mistake | English Explanation | Collection Explanation |
| :---: | :---: | :---: |
| one-word sentences become words | Spoken English does not distinguish between the word and a sentence of one word: "no" vs. "No." | A collection with only one element rarely makes sense: a record collection with only one record isn't really a collection. |
| parentheses not included with sentences, especially one-word sentences | Parentheses are typically ignored when reading. They are far less important than the words they contain. Also, they can be replaced by commas or dashes. | If a collection has a physical container, it is less important than the collection itself. Parentheses may be thought of as a container for the words in a sentence. Furthermore, it is often appropriate to treat a collection as a bunch of objects rather than as a collection. Students may focus more on the words than on the sentence that contains them. |
| empty words and sentences are special cases/not really words or sentences | English does not have such things, so empty words and sentences do not fit with the rest of the analogy. | A collection of zero elements is not a collection at all. |
| empty words or sentences are errors | English does not have such things, so empty words and sentences do not make sense. Students may assume that because of this, something is wrong. Also, students may forget about them unless directly reminded, leaving them with no convenient way to explain what happens if all of the words are removed from a sentence. | It does not make any sense to talk about a group of zero objects. There's nothing there, which is not an option in Scheme. |
| empty words are nothing | TAs often describe empty words and sentences as "nothing" when focusing on what might be in the word or sentence: Scheme code that keeps only the even numbers in the sentence ( $\left.\begin{array}{lll}1 & 3 & 5\end{array}\right)$ returns "nothing" rather than "the empty sentence." | Empty collections aren't really collections: a pile of zero records isn't a pile or, for that matter, anything at all. |
| empty words inside sentences can be ignored | Empty words are nothing. Also, even if empty words make sense on their own, they are overshadowed by the real words in a sentence. | Empty words are nothing. Also, empty categories within a larger collection can usually be removed: if a record collection has no folk albums, why have a tag on the shelf for folk? |

Table 3.9: Common mistakes among CS 3 students and possible English/collection model explanations.
are not legitimate in formal English. They are used in spoken English, but in that case, the content is more important than the form: "No." (the sentence) means the same as "no" (the word). Students using the collection model were predicted to believe that oneelement collections were absurd, so the answer should be a word rather than an illogical one-word sentence. Second, they might omit parentheses around all sentences, even though they believe they are writing sentences. In English, parentheses are used to group specific information, not to delimit a sentence. In addition, parentheses themselves are never read. Students using the collection model should focus entirely on the words. The container for a collection, if one exists, is not something people often think about.

There were also two more complicated questions that dealt with one-word sentences. Both of these required students to read and understand recursive procedures, so they were given on all interviews except for the first of every semester. To prevent students' issues with recursion and reading recursive code from getting in the way, interviewers corrected students whenever they appeared to be confused by recursion.

In the negate-all problem (question O5, Figure 3.6), students were asked to determine whether or not a recursive procedure worked. However, the real test was whether they would notice that it produced a word when it should have actually produced a one-word sentence. Because the interviewers helped students whenever they had trouble with the recursion, students were likely to decide that negate-all worked unless they noticed that it returned a number instead of a one-number sentence.

In the divide-by-largest problem (question O6), students were asked to debug several related recursive procedures. However, the real test was whether they noticed in either the code or the test cases provided that one procedure (sent-max) produced a one-word sentence when it should have produced a word. Again, because of help from the interviewers, students should have either found the bug or given up without discovering it. This problem included two versions; which version a student got depended on when the interview started. All were initially given a handout shown in Figure 3.7

```
The procedure negate-all takes a sentence of numbers and swaps their signs. In other words, it returns a sentence with all of those numbers multiplied by -1 . It doesn't have to work with an empty sentence. Here's a version somebody wrote. Does it work? How do you know? Test it for at least the following sentences: (1), (-1 \(2-3\) ), and (10 203040 -50).
```

```
(define (negate-all sent)
```

(define (negate-all sent)
(if (empty? (bf sent))
(if (empty? (bf sent))
(* -1 (first sent))
(* -1 (first sent))
(se (* -1 (first sent))
(se (* -1 (first sent))
(negate-all (bf sent)))))

```
        (negate-all (bf sent)))))
```

Figure 3.6: Question O5, negate-all.
You and a friend are working on a homework problem together. Let's say that this homework is a group project, so you aren't cheating. Here is the problem:

Write a procedure called divide-by-largest, which takes a sentence of numbers and divides every number in it by the largest number. For example, (divide-by-largest ' (1 234 $\left.\begin{array}{llllllll}3 & 2 & 1)\end{array}\right)$ should give you ( 0.250 .50 .7510 .750 .50 .25$)$.
Your friend writes some of the code, but you write the rest. Here is your code:
(define (divide-by-largest sent)
(divide-sentence-by sent (sent-max sent)))
; ; divide-sentence-by should take a sentence and a num and divide
; ; all of the numbers in that sentence by that num
(define (divide-sentence-by sent num)
(if (empty? sent)
' ()
(se (/ (first sent) num)
(divide-sentence-by (bf sent) num))))
You try (divide-by-largest '( $\left.\begin{array}{lllllll}1 & 2 & 3 & 4 & 3 & 2 & 1\end{array}\right)$ ) and get an error!

Figure 3.7: Question O6, divide-by-largest main handout.

At this point, students were given one of two possible handouts. Those whose interviews started on an odd hour were given one shown in Figure 3.8. Ideally, students given this handout would read the third line (the base case, "sent") and see that sent-max returns a sentence. The base case should actually read "(first sent)".

Those whose interviews started on an even hour were given a handout shown in Figure 3.9. Ideally, these students should have looked at what sent-max returns, noticed the parentheses around every number, and realized that it only returned sentences. This bug

```
Here is your friend's code. Can you find the error?
;;sent-max should take a sentence and return the largest number in it
(define (sent-max sent)
    (cond ((= (count sent) 1)
        sent)
        ((> (first sent) (first (bf sent)))
            (sent-max (se (first sent) (bf (bf sent)))))
            (else (sent-max (bf sent)))))
```

Figure 3.8: Question O6, divide-by-largest test cases handout.

```
Your friend sends you some tests that he or she says will prove that sent-max works. Can
you find the error?
> (sent-max '(55))
(55)
>(sent-max '(1 2 3 4 3 2 1))
(4)
> (sent-max '(-3 -2 -1))
(-1)
```

Figure 3.9: Question O6, divide-by-largest code handout.
was predicted to be at least as easy to identify using the test cases as it was when using the code, since the test cases clearly showed sentences instead of numbers, while the code required students to read and infer more.

Students using analogies to English or the collection model were expected to give certain kinds of answers when explaining the negate-all question. Those using analogies were expected to notice that it returns a number and not be concerned, because either they would not see a difference or they would not think that parentheses are important enough to notice. Those using the collection model were predicted to believe that the procedure produces a number and believe this to be correct, since a one-number sentence would not make sense.

Similarly, students using analogies to understand divide-by-largest should either ignore or never notice the parentheses in the test cases, again because they either did not consider parentheses important or did not see a difference. Students using the collection model would be directly confronted with a one-word sentence and, since they would not
have a place for this in their view of the world, were expected to ignore them and only notice the number.

## Questions About Empty Words and Sentences

Students were given eight questions that related in some way to empty words or sentences. One of those will not be discussed here, because nearly every student had a great deal of trouble with it for reasons that are beyond the scope of this research. The remaining seven are described below.

Two questions dealt directly with creating empty sentences. The second appeared only on the first interview of every semester, in part to allow room for more complicated questions and in part because it raised issues not directly related to one-word sentences.

1. What is (bf '(1)) (question E1)
2. What is (bl '(1)) (question E2)

Because butfirst discards the first word of the sentence, the butfirst of the one-word sentence (1) should be the empty sentence, (). Similarly, butlast discards the last word of the sentence. In a one-word sentence, the first and last words are the same, so the butlast of the one-word sentence (1) would also be (). Students using either analogies to English or the collection model were expected to say that both expressions caused error messages because there was nothing left once the 1 was discarded.

These questions are unique because they were in no way drawn from common mistakes in lab. Students who had experience with recursion (every student except those in the first round of interviews every semester) dealt with this exact case a dozen times or more. Practically every recursive procedure they read or wrote that dealt with sentences took the butfirst of the sentence at every step and stopped when the sentence was empty or when the butfirst of the sentence was empty. In my experience, though, students who
think that taking the butfirst of a one-word sentence should cause an error message rarely have problems writing or understanding recursive code that involves taking the butfirst of a one-word sentence. If students looking at recursive procedures were to follow patterns instead of thinking about how the code actually works, they might be able to deal with the program without actually knowing what happens with one-word sentences. For example, I have seen students who appeared to treat a procedure as "moving to the next word" without really understanding why it did so. Students who miss E1 should either have made a silly mistake or have thought about the problem in a new way, most likely one based on their intuitive understanding of sentences. The same arguments apply to butlast and question E2.

The next two questions asked students what they thought "empty" meant. Students who understood an empty word or an empty sentence on its own might still have had difficulty when confronted with a sentence that contains an empty word.

1. What is (empty? '(""))? (question E3)
2. What is (first '(""))? (question E4)

The procedure empty? takes a word or sentence as its argument. It returns true if that word or sentence contains absolutely nothing, and false otherwise. Because "" is an empty word, it is not absolutely nothing, and the sentence ("") is not empty. The procedure first takes a word or sentence as its argument. If the word or sentence contains at least one element (one letter for a word and one word for a sentence), first will return the first element. If the word or sentence has nothing inside, first will produce an error message. Because "" is a real Scheme word and it is in the sentence (""), (first '("")) should return "".

Students who were using analogies to English should have found these questions confusing, since while they may have accepted empty words as things that stand on their own and say that there is no word, this interpretation falls apart when it is applied to a sentence. Sentences, after all, are designed to contain words. Can they contain something that means
"no word" instead? Students might also have used natural language to assume that since an empty word was "nothing," it should disappear from the sentence. Students who thought that an empty word was a way of writing "no word" should have said that ("") was empty, since there were no words, but they should have said that the first of it was "", since there was still some Scheme object in the sentence. Students who thought the empty word was "nothing" would probably have said that ("") was empty, since it was pretty much the same as (), and that taking first of it would produce an error message, again because trying to take the first of () causes an error message.

Students using a collection model might have assumed that the empty word simply vanished, since it is usually safe to remove empty things from a collection. For example, consider a pile of markers on a table. If one or two run dry, they can be thrown away without a second thought. These students would also probably have said that (") was empty and that taking the first of it would cause an error message, for the same reasons as given above. Some students might also have had trouble because ("") is a one-word sentence. These students thought that a one-word sentence was the same as the word it contained, so they saw ("") as "", which is empty.

One question (E5, Figure 3.10) was asked only on the second round of interviews in the Spring A and Summer A semesters. This question asked whether students thought extra empty words in a sentence were bad. Students were asked to consider a procedure called number-spell, shown four possible incorrect return values, one of which contained an empty word, and asked which incorrect value was most serious and which was least serious.

This question was based on students' behavior in the Summer 2002 semester. Those students did write number-spell, and they did make these four mistakes. Students who believed that empty words disappeared or were overshadowed in a sentence should have said that option 3 was the least bad, or possibly that it was correct, in particular because empty words were not really things in the same way that words with letters were. Students who believed that empty words were regular words were expected to say that 3 was not

```
This summer we had students write a program called number-spell, which takes a number and returns a sentence with that number written out in words. For example, (number-spell 11000 should give (eleven thousand). Don't worry, you won't have to write this!
Students made a lot of mistakes on this. Here are four examples:
1. One student's program spelled 11000 as (eleven thousand zero zero zero)
2. One student's program spelled 11000 as (eleven thousand zero)
3. One student's program spelled 11000 as (eleven thousand "")
4. One student's program spelled 11000 as (ten one thousand)
Which of these sound like the most serious error to you? Which sound like the least serious? Why?
```

Figure 3.10: Question E5, number-spell.
much worse than 2 , since both were essentially the same mistake. They might also have said that option 3 was better than option 2 because it looked better from a human point of view, or because it was less misleading, since the empty word was clearly not a number, while zero was. Students in Spring and Summer A were the only ones to see this question. It was not given to students in the other semesters because they actually did the numberspelling project, and I did not want students' actual experience with this program and the comments from their TAs to influence their opinions. The number-spelling program was also assigned in Summer A, but interviews were planned to take place several days before it was assigned. Unfortunately, three of the students were interviewed after they started working on the problem. They were not asked this question.

One question (E6, Figure 3.11) was asked only on the first round of the Spring A interviews. In previous semesters, I had seen students write procedures that checked for invalid arguments and, if there were any, returned empty sentences. This question asked students to look at one such procedure and explain what the author might have intended.

In this example, sum-of-square-roots makes sure that neither number is negative. If either one is, sum-of-square-roots returns the empty sentence. Otherwise, it properly computes the sum of the square roots of the two numbers. Students who used English or a collection model were expected to say the empty sentence represented an error or was a way

```
Here is something that you might have been asked to do in homework or on a test:
Write a procedure called sum-of-square-roots, which takes two numbers, \(x\) and \(y\), finds the square root of each, and returns the sum of those square roots. \(x\) and \(y\) will not be negative numbers.
One student writes this as an answer on the test:
(define (sum-of-square-roots \(x\) y)
(if (or (<x 0) (< y 0)) ;;line 1
'() ;;line 2
(+ (sqrt x) (sqrt y)))) ; ;line 3
Why did this person include lines 1 and 2 ?
```

Figure 3.11: Question E6, sum-of-square-roots.
of returning nothing. Other students were expected to have been confused. It was removed from later interviews because every student found it completely baffling.

The final question (E7) asked students whether they thought the butfirst of a oneletter word would produce an error and, if they did not, whether they thought it would stay in a sentence or disappear. This question was, "What is (every bf ' (1 22333 4444))?" The procedure every calls bf (butfirst) on every word in the sentence (1 22 333 4444). This may seem strange, but numbers are considered to be words, and their digits are considered to be letters. The butfirst of 1 is " ", because there is only one letter in 1. The empty word is a real word in Scheme, so it should stay in the sentence. The result should be ("" 233 444). Students who accepted the creation of an empty word by getting rid of the only remaining letter in a word but who still relied on English to understand it should have said the answer was (2 33 444), either because they thought empty words were nothing or because they thought the empty word would be overshadowed by the real words in the sentence. (In this case, students might have assumed that the empty word meant "no word" and that it would not be needed in the sentence because there were words present.)

Students who accepted the creation of empty words but who still relied on the idea of a collection to understand it should have said the answer was (2 33 444) , in this case because the empty word would be removed by Scheme or or because it was nothing.

## Questions About the Elements of English and Scheme Sentences

Starting with the first round of interviews in Fall A, students were asked what they thought the essential elements of English and Scheme sentences were. Students were generally allowed to interpret the questions, although interviewers would correct them if they completely misinterpreted a question. The questions were

1. What do you think are the essential parts of an English sentence? (question P1)
2. What do you think are the essential parts of a Scheme sentence? (question P2)

All students, including those who used English or a container model, should have mentioned words as essential parts of both English and Scheme sentences. Students who used analogies between English sentences and Scheme sentences were likely to explicitly compare and, more importantly, contrast English and Scheme sentences, because finding meaningful differences between two things is substantially easier for people who already see the two as similar (Gentner \& Markham, 1994). Students with a good understanding of sentences ${ }^{1}$ should have said that parentheses were also essential to Scheme sentences. Students who used English to understand Scheme sentences but were careful about how they did so should have been likely to mention starting an English sentence with a capital letter (equivalent to an open parenthesis in a Scheme sentence) and ending it with a punctuation mark (equivalent to a close parenthesis in a Scheme sentence). Unfortunately, the issue of comparing and contrasting was not considered until after the Spring B interviews, so interviewers generally did not follow up on answers that might have been interesting.

[^5]
## Interviews by Semester

## Spring A

There were three rounds of interviews in Spring A. These are described below and summarized in Table 3.10. The first round of interviews was conducted between February 10 and February 14. This was the fourth week of school and the week before the first midterm. Thirty-seven students (twenty-nine percent of the class at that time) participated. This interview contained three parts:

1. Five warm-up questions were intended to get the students talking and thinking. Most were taken from an old CS 3 exam and were rather difficult. The questions did not elicit any more comments (students generally thought out loud from the very first question), and they took a great deal of time. For these reasons, they were dropped from later interviews.
2. Six questions about sentences and words asked students about the first, last, butfirst, and butlast of different sentences (questions O1-O4, described on page 44, and E1 and E2, described on page 49).
3. Three questions asked students about empty words and sentences. Two dealt with an empty word in a sentence (questions E3 and E4, described on page 50). The third attempted to determine whether students considered empty sentences to be errors (question E6, described on page 52).

The second round of interviews was conducted between March 3 and March 7. This was the seventh week of school and the week before the second midterm. Thirty-four students (twenty-seven percent of the class at that time) participated. This interview also contained five parts:

1. Three questions about words and sentences were taken from the first interview. Only the first and butfirst questions (O1, O3, and E1) were kept, to make room for additional questions.
2. Two questions about empty words in sentences (E3 and E4) were taken directly from the first interview.
3. negate-all (question O5, described on page 46)
4. number-spell (question E5, described on page 51)
5. divide-by-largest (question E6, described on page 52)

The third round of interviews was conducted between April 21 and April 25. This was the $13^{\text {th }}$ week of school and two weeks before the end of the semester. Thirty-four students (thirty-one percent of the class at that time) participated. Students in this interview were divided into two groups. Those whose interviews started on even hours were given list questions, while those whose interviews started on odd hours were given sentence questions. After these questions, all students were asked another set of questions about their interactions with their TAs. These questions are unrelated to the topic of this thesis and will not be discussed further. Seventeen students were given list questions and seventeen were given sentence questions. This interview was composed of five parts:

1. Negate-all (question O5) was moved to the front of the interview to separate the two hardest questions (it and divide-by-largest). The list version was exactly the same, except that all references to sentences were replaced with references to lists.
2. The three sentence and word (first and butfirst) questions from the second interview (O1, O3, and E1) were included. The list version was exactly the same, except first was replaced with car and butfirst was replaced with cdr.

| Interview | Students | Time | Questions |
| :---: | :---: | :---: | :---: |
| 1 | 37 | February 10-14 | 5 warm-up |
|  |  | $4^{\text {th }}$ week | 6 word/sentence |
|  |  | just before $1^{\text {st }}$ midterm | 3 empty |
| 2 | 34 | March 3-7 | 3 word/sentence |
|  |  | $7^{\text {th }}$ week | 2 empty |
|  |  | just before $2^{\text {nd }}$ midterm | 1 word or one-word sentence |
|  |  |  | 1 empty |
|  |  |  | 1 word or one-word sentence |
| 3 | 34 | April 21-25 <br> $13^{\text {th }}$ week <br> two weeks before end of school | either list or sentence questions |
|  |  |  | 1 word or one-word sentence/list |
|  |  |  | 3 word/sentence or word/list |
|  |  |  | 2 empty |
|  |  |  | 1 word or one-word sentence/list |

Table 3.10: Number of students involved, timing, and general kinds of questions for each interview in Spring A.
3. The two questions about empty words in sentences from the first interview (E3 and E4) were also included. The list version asked the same questions about a null list in another list.
4. The divide-by-largest question (O6) from the second interview was also included, although all students were given the test cases first. The list version was exactly the same, except that all references to sentences were replaced with references to lists.

## Summer A

There were three rounds of interviews in the Summer A semester. They are described below and summarized in Table 3.11. The first was conducted between July 3 and July 7. This was the second week of the summer semester (equivalent to the fourth week of the other semesters) and the week before the first midterm. Twelve students (thirty-three percent) participated. This interview was essentially the same as the first interview in Spring A, except that it did not include the five warm-up questions. It contained two parts:

1. The six word-and-sentence questions from the first Spring A interview were included.

| Interview | Students | Time | Questions |
| :---: | :---: | :---: | :---: |
| 1 | 12 | July 3-7 | 6 word/sentence |
|  |  | $2^{\text {nd }}\left(4^{\text {th }}\right)$ week just before $1^{\text {st }}$ midterm | 2 empty |
| 2 | 10 | July 16-18 | 3 word/sentence |
|  |  | $4^{\text {th }}\left(8^{\text {th }}\right)$ week | 2 empty |
|  |  | just before second midterm | 1 word or one-word sentence |
|  |  |  | 1 empty |
|  |  |  | 1 word or one-word sentence |
| 3 | 11 | August 11-14 | either list or sentence questions |
|  |  | $8^{\text {th }}\left(15^{\text {th }} \& 16^{\text {th }}\right)$ week | 1 word or one-word sentence/list |
|  |  | last week of class | 3 word/sentence or word/list |
|  |  |  | 2 empty |
|  |  |  | 1 word or one-word sentence/list |

Table 3.11: Number of students involved, timing, and general kinds of questions for each interview in Summer A.
2. The two questions about empty words in sentences were also included.

The second round of interviews was conducted from July 16 to July 18. This was the fourth week of class (equivalent to week eight in the regular semester), and the week before the second midterm. Ten students (twenty-nine percent) took part in this round, but four of the interview sheets were lost. The interview questions were identical to those from the second round of interviews in the previous semester.

The third round of interviews was conducted from August 11 to August 14. This was the eighth week of class (like the $15^{\text {th }}$ and $16^{\text {th }}$ weeks of a regular semester) and took place just before the final exam. A total of 11 students $(34 \%)$ took part in this round of interviews. Questions were the same as those from the third round of the previous semester.

## Fall A

There were four rounds of interviews in Fall A. They are described below and summarized in Table 3.12. When students finished the interview questions, they were asked another set of questions identical to those asked in the final round in Spring A; those questions will not be discussed here. The first round took place from September 8 to September

| Interview | Students | Time | Questions |
| :---: | :---: | :---: | :---: |
| 1 | 37 | September 8-12 | 6 word/sentence |
|  |  | $3^{\text {rd }}$ week | 2 empty |
|  |  | just before $1^{\text {st }}$ midterm | 2 English/Scheme sentences |
| 2 | 34 | October 6-10 | 3 word/sentence |
|  |  | $7^{\text {th }}$ week | 2 empty |
|  |  | just before second midterm | 2 word or one-word sentence |
| 3 | 34 | November 3-7 <br> $11^{\text {th }}$ week just before third midterm (faux) | 3 word/sentence |
|  |  |  | 2 empty |
|  |  |  | 1 word or one-word sentence |
|  |  |  | 2 higher-order/empty |
|  |  |  | 1 word or one-word sentence/list 2 English/Scheme sentences |
|  |  | December 1-11 $15^{\text {th }}$ and $16^{\text {th }}$ weeks last weeks of class |  |
| 4 | 52 |  | either sentence or list questions |
|  |  |  | 3 word/sentence |
|  |  |  | 2 empty |
|  |  |  | 1 word or one-word sentence |
|  |  |  | 2 higher-order/empty |
|  |  |  | 1 word or one-word sentence/list |
|  |  |  | 2 English/Scheme sentences/lists |

Table 3.12: Number of students involved, timing, and general kinds of questions for each interview in Fall A.
12. This was the third week of the semester and the week before the first exam. Thirty-seven students (twenty percent) participated. The questions were divided into three parts:

1. The six word-and-sentence questions from the first Spring A interview (O1-O4, E1 and E2) were included.
2. The two questions about empty words in sentences (E3 and E4) were also included.
3. Students were asked to list the essential elements of English and Scheme sentences (questions P1 and P2, described on page 54).

The second round of interviews took place from October 6 to October 10. This was the seventh week of school and the week right before the second exam. A total of 37 students $(19 \%)$ took part. There were five groups of questions, none of them new:

1. The three word-and-sentence (first and butfirst) questions (O1, O3, E1)
2. The two questions about empty words in sentences (E3 and E4)
3. negate-all (O5)
4. divide-by-largest (O6) -again, half of the students were given all code and half were given some code and some test cases
5. The two questions about the elements of English and Scheme sentences (P1 and P2)

The third round of interviews took place from November 3 to November 7. This was the $11^{\text {th }}$ week of school and the week before the third exam. This exam was actually a "faux midterm," meaning that it was given out but not graded. There was a review session for it, and students took it reasonably seriously. A total of 34 students ( $20 \%$ ) took part. However, this interview took place three weeks before the third interview in Spring A, because there was one additional test in Fall A. This interview was composed of six parts:

1. The same three word-and-sentence questions (O1, O3, and E1)
2. The same two empty word questions (E3 and E4)
3. negate-all (O5)
4. Two questions about higher-order procedures (question E7, described on page 53, and one that will not be discussed in this paper)
5. divide-by-largest (O6)
6. The essential elements of English and Scheme sentences (P1 and P2)

The fourth round of interviews took place from December 1 to December 11. It covered the $15^{\text {th }}$ and $16^{\text {th }}$ weeks of school. These were the final week of class and the week before the final exam. A total of 52 students took part, although one began to panic when the Scheme questions came out and was excused. A total of 51 students (31\%) answered questions about either sentences or lists. Students were given list questions if their interviews started
Interview
1

2

3

4

## Students

Time
19
February 6-20
$5^{\text {th }}$ week
just before $1^{\text {st }}$ midterm
20 March 8-12
$8^{\text {th }}$ week last week of recursion

13

17

May 3-7
$15^{\text {th }}$ week
last week of class

Questions
6 word/sentence
2 empty
2 English/Scheme sentences
3 word/sentence
2 empty
2 word or one-word sentence
2 English/Scheme sentences
3 word/sentence
2 empty
1 word or one-word sentence
2 higher-order/empty
1 word or one-word sentence/list
2 English/Scheme sentences
either sentence or list questions
3 word/sentence
2 empty
1 word or one-word sentence
2 higher-order/empty
1 word or one-word sentence/list
2 English/Scheme sentences/lists
Table 3.13: Number of students involved, timing, and general kinds of questions for each interview in Spring B.
in even hours and sentence questions if their interviews started in odd hours. Twenty-seven student answered list questions, and twenty-four answered sentence questions.
The sentence questions used in this interview were identical to those used in the third interview, except that all students started with the test cases for divide-by-largest. The list questions were almost the same questions, with only a few minor changes. First, all references to sentences were replaced with lists. All sentence procedures, such as first, butfirst, empty?, every, and sentence, were replaced with the equivalent list procedures, such as car, cdr, null?, map, and cons. In addition, while the sentence students were asked about empty words in sentences, the list students were asked about null lists in lists.

## Spring B

There were four rounds of interviews in Spring B. They are described below and summarized in Table 3.13. The first round took place from February 16 to February 20. This was the fifth week of the semester and the week right the first exam. Nineteen students (twenty-one percent) participated. The questions were the same as those in Fall A.

The second round of interviews took place from March 8 to March 12. This was the eighth week of school and the last week in which students studied recursion. Had there been three midterms, the second would have been the next week. A total of 20 students ( $26 \%$ ) took part. The questions were the same as those in Fall A.

The third round of interviews was conducted between April 5 and April 9. This was the $12^{\text {th }}$ week of school and the week before the second exam. In addition, it appears it was a very bad week for most students. Many students signed up to participate, but only $13(17 \%)$ made it to the interviews. The questions on this interview were intended to be exactly the same as those from the third interview in Fall A, but they were not followed by questions about help-seeking. Unfortunately, due to a miscommunication, half of the students were given the second interview from Spring A. Additionally, students seemed to have an unusual amount of trouble answering the questions. Several said that they were tired and not really trying. All of these factors lead me to ignore this round of interviews.

The fourth round of interviews took place from May 3 to May 7. This was the $15^{\text {th }}$ and final week of school. A total of 19 students (26\%) took part. Students were given list questions if their interviews started in even hours and sentence questions if their interviews started in odd hours. Nine student answered list questions, and ten answered sentence questions. The questions were the same as those in Fall A. Students had been exposed to lists on the $13^{\text {th }}$ week, and they had been forced to use them in their projects.

### 3.4 Interview Procedure

The basic format of every interview was the same. Each individual student sat in a small room with an interviewer. There was a brief introduction, during which the interviewer informed the student that the results would be completely anonymous and would not be graded. The interviewer also asked the student to think out loud. If the student appeared to be nervous, the interviewer would ask if the student was willing to answer the questions. If the student was willing, the interviewer gave the student a blank sheet for scratch work and answers and another sheet with the questions. As the student worked, the interviewer took notes on the student's activities, comments, and questions.

If the interviewer realized that the student did not understand a question, the interviewer would give an explanation. If the student made a serious mistake on one of the debugging questions, the interviewer corrected the student. The goal of both of these activities was to try to make sure that if students got one of the complex problems wrong, they did so because they had genuine misconceptions rather than simple misunderstandings.

Interviewers were instructed to listen for certain comments during the interview and to ask follow-up questions if needed. When a student finished the Scheme questions, the interviewer would go through each question and ask the student how s/he got the answer. Once the student had explained the reasoning behind all related questions, the interviewer would say whether those answers were right or wrong, giving explanations if needed.

After a student had explained all of his/her answers and the interviewer had explained any mistakes, the student was encouraged to talk about the class in general. Students in Fall A and those taking the final interview of Spring A were also asked a more structured set of questions about how they interacted with their TAs and how they got help. Some students used the remaining time to complain about or praise the course, offer suggestions, ask about unrelated CS 3 topics or other academic matters, or chat.

Four different people conducted interviews over the three semesters. One was Interviewer A, a male computer science graduate student. The other three were female undergraduates. Two, B and C, were computer science majors, and the third, D, was a cognitive science major. A had seven semesters of experience as a TA for CS 3. B, C, and D had taken CS 3 and worked as lab assistants for the class. C and D also had experience grading CS 3 homework and quizzes. A conducted 266 interviews, B conducted 7, C conducted 22, and D conducted 70 . $\mathrm{A}, \mathrm{B}$, and D practiced interviews with three students before the first round of interviews in Spring A and agreed upon standards for the interviews. Interviewers were given an introduction to give to each student, told when to ask and answer questions, given a series of questions to ask when students gave certain kind of answers, and told which kinds of student behavior they should make note of. Interviewer C volunteered to help at the end of Spring A and was not properly trained before conducting interviews. The five interviews conducted by this person were discarded and she was retrained before the start of interviews next semester.

### 3.5 Scoring

Each question was worth one point if it was answered correctly and zero points otherwise. If a response was correct as written but the student gave an explanation that indicated they got the right result for the wrong reason, such as saying that (" ") was not empty because it contained a two-letter word, that answer was counted as wrong.

Students were counted as believing something-for example, that (bf '(12)) produces the number 2-if they said that they believed it. Students were counted as not believing something, for example that (bf ' (12)) produces the sentence (2), if they said that they believed something else or thought something else might be true.

The coding was done by one person. A second person was then given twenty interviews to code. The two agreed on $93 \%$ of the 229 items on the interviews.

### 3.6 Possible Treatments

Because the treatments were devised before the author became familiar with the collection model, they focus almost exclusively on English and Scheme. To be useful, an activity should either encourage students to examine and refine their Scheme-English analogies or offer them a more interesting analogy. Two homework problems and one lab activity were designed to help students overcome their problems.

One homework activity asked students to compare and contrast English and Scheme. The other asked students to define Scheme sentences and words. Both were intended to make students think about the relationship between Scheme and English and realize that they might to do need more than just rely on their knowledge of English words and sentences.

The lab activity was intended to give students a model for empty words and sentences, since they were unlikely to have one of their own. During the Spring A interviews, one interviewer tried to explain (butfirst ' (12)) and empty sentences by comparing a sentence to a bag of bagels. The bag contains bagels, just as the sentence contains words. Taking the top bagel out of the bag is like removing the first word from the sentence. The top bagel is the first of the sentence, while the rest of the bag is the butfirst of the sentence. Taking out the top bagel from a bag of two bagels leaves one bagel, which is still in a bag. Similarly, taking the butfirst of a two-word sentence leaves the last word in the sentence. Taking the last bagel out of the bag results in an empty bag, while taking the last word out of the sentence returns an empty sentence. The interviewer said the students liked the analogy. There was, however, one minor problem with this analogy. An empty bag has no value and could be discarded without a second thought. A good analogy should imply that the empty sentence does in fact have a value and is a thing. Instead, the lab activity dealt with $\mathrm{Pez}^{\circledR}$ dispensers. Empty dispensers are still things, and some people collect them. There is even a museum for them (the Burlingame $\mathrm{Pez}^{\circledR}$ Museum, http://www.pezmuseum.com).

Copies of these activities can be found in Appendix B.

## Chapter 4

## Results

### 4.1 One-Word Sentences

## Questions O1-O4: butfirst and butlast of One-Word Sentences

Most students correctly answered the butfirst and butlast questions (O1-O4, described on pages $44-46$ ), but a substantial minority gave wrong answers in the two-word cases and some gave wrong answers in the four-word cases. The percentages of students who made mistakes on the butfirst and butlast questions, as well as the percentages of the total mistakes predicted, are shown in Tables 4.1 and 4.2. The third and fourth columns ("bf2" and "bf4") show the numbers and percentages of students giving an incorrect answer for the butfirst and butlast of two- and four-word sentences. The fifth column ("both") shows how many students gave correct answers for both the two- and four-word cases. The fifth column ("bf $2=\mathrm{bf} 4$ ") shows how many students gave answers for two- and four-word

```
O1: What is (bf '(1 2))?
O2: What is (bl '(1 2))?
O3: What is (bf '(1
O4: What is (bf '(1
```



Table 4.1: Numbers and percentages of students who gave the correct answer for (bf '(12)) and for (bf '(123 4)), as well as those who got both correct and who either got both wrong or both right. Also the numbers and percentages (out of all students who made mistakes) of those who made mistakes of the kind predicted earlier. Interviews are written as the semester "SpA" for Spring A, "SuA" for Summer A, "FaA" for Fall A, and "SpB" for Spring B; followed by "I" and the interview number. The final interview in each semester was divided into sentence questions ("Sents") and list questions ("Lists").

| Interview | \# students | b12 | b14 | both | $\mathrm{bl2}=\mathrm{bl4}$ | \# predicted, |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | \# correct, | \# correct, | \# correct, | \# same, | \% predicted |
|  |  | \% correct | \% correct | \% correct | \% same |  |
| SpA I1 | 37 | 24, 65\% | 32, 86\% | 24, $65 \%$ | 29, $78 \%$ | 6, $46 \%$ |
| SuA I1 | 12 | 8, $67 \%$ | 8, $67 \%$ | 8, $67 \%$ | 12, 100\% | 2, $50 \%$ |
| FaA I1 | 37 | 27, 73\% | 32, $86 \%$ | 26,70\% | 30, 81\% | 5, 45\% |
| SpB I1 | 19 | 11, $58 \%$ | 16, 84\% | 11, $58 \%$ | 14, $74 \%$ | 8, 100\% |

Table 4.2: Numbers and percentages of students who got the right answer for (bl '(12)) and for (bl '(1) 2 (1) $4)$ ), as well as of those who got both correct and whose answers were either both right or both wrong. Also, numbers and percentages (out of all students who made mistakes) of students who made the kind of mistakes predicted earlier. Interviews are written as the semester "SpA" for Spring A, "SuA" for Summer A, "FaA" for Fall A, and "SpB" for Spring B; followed by "I" and the interview number.
sentences that were either both correct or both incorrect. The sixth column shows the percentage of students whose mistakes can be explained by either analogies or the collection model.

Students were counted as making predicted mistakes if they gave any of the following responses:

1. the correct answer for the four-word case but no parentheses for the two-word case, and they said that the two-word case did not need parentheses
2. the correct answer for the four-word case but no parentheses for the two-word case, and they were not sure whether the two-word case needed parentheses
3. no parentheses for either answer and they believed that parentheses were not needed
4. no parentheses for either answer and they were not sure whether parentheses were needed

Students were not counted if they made either of those mistakes but said that they knew the correct answers and had written the wrong things by mistake.

Students using analogies to English might believe that both answers are sentences but that sentences do not require parentheses, since parentheses do not delimit sentences in English, or they might believe that (2 344 ) is a sentence but that (2) is a number, since English does not deal with many one-word sentences. Students using a collection model would probably believe that (2) is a number, since a sentence is a collection and thus requires more than one element. Most of the students whose mistakes with butfirst were not counted as predicted said that (bf '(12)) returned the sentence 2 (without parentheses) and claimed to have forgotten to put parentheses. Several others thought that butfirst meant "second," so (bf '(12)) should return the word 2, as should (bf ' (1 2 3 4)). Students whose mistakes with butlast were not predicted often thought that the last element was or involved parentheses, so that taking the butlast would get rid of the parentheses.

As shown in Table 4.1, mistakes with butfirst tended to decrease as each semester progressed, but they did not quite disappear. Most of the mistakes made by students can be explained by students' use of English or the collection model, especially after the first interview. In the first interview, students made a number of other mistakes, including confusing butfirst with first or butlast. Table 4.2 shows that mistakes with butlast were not as easy to predict, most likely because they were only asked on the first interview
each semester. While students did not make many more mistakes with butlast than with butfirst, they had a wider range of reasons for making their mistakes.

Average scores on these four questions did not significantly improve from semester to semester (see Table 4.3). However, there were differences among the semesters. Students in Spring A often thought that either the butfirst of a two-word sentence was a number or parentheses were not required when writing sentences. Students in later semesters considered these two possibilities but were not certain.

Students in the first round of interviews in Spring A found questions about the butfirst or butlast of a two-word sentence to be confusing. Thirty-nine percent of the students who missed (bf '(12)) (eleven percent of all students that semester) made it very clear that they believed one-word sentences either did not exist or did not require parentheses. One student, explaining the difference between one and three elements, said, "If there's something left, return that something. . . unless there are more than one somethings. . . then [they should be returned] in a sentence." This student went on to say that there was no use returning one element in a sentence. When talking aloud while working on the (bf '(1 2)) problem, two students said that the answer should be a word. A student who answered all of these questions correctly said that, until very recently, $\mathrm{s} / \mathrm{he}^{1}$ had not put parentheses around one-word sentences: "When you use parentheses in English, it doesn't have a special meaning. It is just to group things together." Students were often required to deal with the butfirst of a two-word sentence during the semester, and they did significantly better on this question as the semester progressed ( $p<.05$ ).

Forty-one percent of the students who missed (bf '(12)) in the Fall A semester (six percent of all students that semester) either were not sure whether the answer should be a number or thought that it was a sentence but were not sure whether that sentence needed parentheses. In addition, four students who got the right answer expressed similar concerns.

[^6]| Semester | butfirst |  |  | butlast |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | \# Students | two-word | four-word | Students | two-word | four-word |
| SpA Sents | 88 | \#\% correct, <br> 68, $77 \%$ | \#,\% correct, $80,91 \%$ | 37 | \#\% correct, $24,65 \%$ | \#\% correct, <br> 32, $86 \%$ |
| SpA Lists | 17 | 17, 100\% | 17, 100\% | - | - | - |
| SuA Sents | 23 | 19, $83 \%$ | 20, $87 \%$ | 12 | 8, $67 \%$ | 8, $67 \%$ |
| SuA Lists | 6 | 4, $67 \%$ | 5, 83\% | - | - | - |
| FaA Sents | 130 | 113, $87 \%$ | 122, $94 \%$ | 37 | 27, $73 \%$ | 32, $86 \%$ |
| FaA Lists | 26 | 23, $88 \%$ | 24, $92 \%$ | - | - | - |
| SpB Sents | 49 | 39, $80 \%$ | 46, 94\% | 19 | 11,58\% | 16, $84 \%$ |
| SpB Lists | 7 | 6, 86\% | 7,100\% | - | - | - |

Table 4.3: Semester summary for O1-O4, taking the butfirst and butlast of two- and four-word sentences. Semesters are written as SpA for Spring A, SuA for Summer A, FaA for Fall A, and SpB for Spring B, followed by either "Sents" for all of the sentence questions from all interviews or "Lists" for the list questions from the final interview.

|  |  | Answers by Category for Negate-all (\#, \%) |  |  |  |  |  |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- |
|  |  |  |  | Predicted |  |  |  |
| Semester | \#Students | Correct | Sent | Not Notice | No Problem | Depends | Other |
| SpA I2 | 34 | $5,15 \%$ | $9,26 \%$ | $13,38 \%$ | $2,6 \%$ | $4,12 \%$ | $1,3 \%$ |
| SpA I3 Sents | 17 | $5,29 \%$ | $3,18 \%$ | $3,18 \%$ | $5,29 \%$ | $1,6 \%$ | $0,0 \%$ |
| SpA I3 Lists | 17 | $8,47 \%$ | $3,18 \%$ | $5,29 \%$ | $0,0 \%$ | $1,6 \%$ | $0,0 \%$ |
| SuA I2 | 6 | $1,17 \%$ | $2,33 \%$ | $3,50 \%$ | $0,0 \%$ | $0,0 \%$ | $0,0 \%$ |
| SuA I3 Sents | 5 | $3,60 \%$ | $0,0 \%$ | $0,0 \%$ | $1,20 \%$ | $1,20 \%$ | $0,0 \%$ |
| SuA I3 Lists | 6 | $3,50 \%$ | $1,17 \%$ | $1,17 \%$ | $0,0 \%$ | $0,0 \%$ | $1,17 \%$ |
| FaA I2 | 34 | $9,26 \%$ | $3,9 \%$ | $12,35 \%$ | $6,18 \%$ | $2,6 \%$ | $2,6 \%$ |
| FaA I3 | 34 | $12,35 \%$ | $3,9 \%$ | $7,21 \%$ | $7,21 \%$ | $4,12 \%$ | $1,3 \%$ |
| FaA I4 Sents | 25 | $11,44 \%$ | $4,16 \%$ | $4,16 \%$ | $2,8 \%$ | $2,8 \%$ | $2,8 \%$ |
| FaA I4 Lists | 26 | $11,42 \%$ | $6,23 \%$ | $8,31 \%$ | $0,0 \%$ | $1,4 \%$ | $0,0 \%$ |
| SpB I2 | 20 | $2,10 \%$ | $3,15 \%$ | $8,40 \%$ | $5,25 \%$ | $2,10 \%$ | $0,0 \%$ |
| SpB I4 Sents | 10 | $2,20 \%$ | $0,0 \%$ | $3,30 \%$ | $5,50 \%$ | $0,0 \%$ | $0,0 \%$ |
| SpB I4 Lists | 7 | $3,43 \%$ | $2,29 \%$ | $1,14 \%$ | $0,0 \%$ | $1,14 \%$ | $0,0 \%$ |

Table 4.4: Numbers and percentages of students who gave correct and incorrect answers to the negate-all question, as well as specific categories of incorrect answers. Interviews are written as the semester "SpA" for Spring A, "SuA" for Summer A, "FaA" for Fall A, and "SpB" for Spring B; followed by "I" and the interview number. The final interview in each semester was divided into sentence questions ("Sents") and list questions ("Lists").

Unlike students in the Spring A semester, none of these students was certain that a wrong answer was correct.

Sixty percent of the students who missed (bf '(12)) in the Spring B semester (eight percent of all students in that semester) either believed that the answer should be a number or thought the answer was a sentence but were not sure whether that sentence needed parentheses.

The procedure negate-all takes a sentence of numbers and swaps their signs. In other words, it returns a sentence with all of those numbers multiplied by -1 . It doesn't have to work with an empty sentence. Here's a version somebody wrote. Does it work? How do you know? Test it for at least the following sentences: (1), (-1 2 -3) , and (10 203040 -50).

```
(define (negate-all sent)
    (if (empty? (bf sent))
        (* -1 (first sent))
        (se (* -1 (first sent))
            (negate-all (bf sent)))))
```

Figure 4.2: Question O5, negate-all.

## O5: negate-all

Students did not do so well with negate-all (O5, page 46). Their answers generally fell into four categories of mistakes. Some (counted as "Sent" in Table 4.4) wrote that negate-all of a one-word sentence would return a sentence. Others ("Not Notice") wrote that it returned a number but never appeared to be aware of that fact. When asked, they said they had written a sentence. Others ("No Problem") were aware that it returned a number but did not seem at all concerned. Finally, a few ("Depends") noticed that it returned a number but said that whether it worked depended on how it was being used-it worked well if its results were only for human consumption but it did not work if its results were to be used by another Scheme program. Several students failed to understand the code at all or never committed to an answer. These student fall into the "Other" category.

Most of the mistakes were categorized as "Not Notice," one of the categories predicted for those who relied on English analogies. In the majority of these cases, students wrote that negate-all of a one-word sentence returned a sentence and went on with the problem, never noticing that anything was out of the ordinary. When asked, most of these students did not show any awareness that they had written a number rather than a sentence. Some, however, initially wrote down a one-word sentence but corrected it to be a number. When questioned by interviewers, these students also appeared unaware that anything was wrong. A small minority of the students in this category used no parentheses around sentences in
any of their scratch work. The vast majority of the students in all three of these conditions either completely ignored the difference between a word and a one-word sentence or almost immediately forgot about it.

The second most common category of mistake was "Sent." Students who wrote that (negate-all '(1)) returned the sentence (-1) either believed in one-word sentences or were swayed by the statement that negate-all was supposed to produce sentences. Most of these students never noticed that the code actually produced a number, but a few initially wrote down a number and added parentheses later.

The third category of mistakes was "No Problem," meaning that students were aware that negate-all produced a number but believed this to be correct, or at least acceptable. Students generally followed one of two patterns. In the first, they said that negate-all worked, except for one-word sentences. Then they told the interviewer that the program worked. In the second, less common pattern, students said that the program worked and that returning a number in one case and a sentence in another was fine because there was not really a difference. Both patterns in this category were predicted by analogies to English, and the second pattern was predicted by the collection model.

The least common but still identifiable category of mistakes was "Depends." Rather than saying that negate-all worked or did not work, these students said that it depended on how negate-all was being used. Like many of the students in the No Problem category, these students did not see anything particularly wrong with returning a sentence in some cases and a number in others, at least when a human would be inspecting the results. However, these students realized that Scheme differentiated between words and sentences, saying that it would not work if it was to be used with more Scheme code. Most of these students appeared to believe this second case was rather unlikely. This belief is probably a consequence of the way CS 3 is taught. Most activities involve procedures that are never seen again, so students might not develop the idea that real programs involve multiple
procedures working together. However, the category itself is predicted by both English and the collection model.

Predicted answers accounted for $60-85 \%$ of the mistakes in almost every semester. There were a few students who gave wrong answers for other reasons. Some students completely misunderstood the code and thought that it caused an actual error message. Interviewers corrected these students and encouraged them to keep working on the problem; most of these students settled into one of the five standard categories, but a few never did. Some students looked at each line of the code, decided that it did not contain a bug, and never considered the procedure as a whole.

The data under "No Problem" in Table 4.4 suggests that there is a difference in how students think about sentences and lists. In the final interviews of Spring A (SpA I3) and Spring B (SpB I4), more students correctly answered the list version than correctly answered the sentence version of this problem. This difference was not significant. However, none of the students who were given the list version of negate-all answered No Problem. Chisquare tests show that this is significant for the Spring A and B semesters ( $p<.03$ in each case). One possible explanation is that according to the rules taught in CS 3, the list version of negate-all produces an error message no matter what input it is given, while the sentence version never produces an actual error message. Thus, students who are aware of negate-all's behavior should, in the case of lists, have a very good reason to say negate-all does not work properly.

The "No Problem" and "Depends" answers are consistent with the findings of Kolikant (2005). Students may have been willing to judge this program as relatively correct because they saw negating a sentence as somewhat abstract, or they may have done so because of the way they thought about sentences. However, because students were never willing to answer "No Problem" for list questions, the latter possibility seems more likely.

While students in later semesters were not consistently more likely to give the correct answer than students in Spring A, their explanations did get better. Table 4.5 shows the responses for all students by semester.

A number of students in Spring A who gave "No Problem" answers used a defective understanding of domain and range to justify their statements. These students defined domain and range as properties of the code itself without any relation to how the code should perform. The domain was the set of all arguments that this particular version of the code happened to take without giving an error, and the range was the set of values that this particular version of the code happened to return, whether or not they were what it should return. Students explained that the procedure was correct because it matched its domain and range, and they explained that the domain and range were based on how the code worked. This circular logic has little to do with analogies to English or the collection model. When the way domain and range were taught changed in Summer A and later semesters, students made the same mistakes but had more trouble justifying them.

Students in Spring A were more likely than students in later semesters to say that negate-all returned a sentence. This difference is significant for Fall $\mathrm{A}(p<.05)$ and nearly significant for Spring $\mathrm{B}(p<.07)$. While none of the interventions specifically targeted this mistake, students may have learned to be more careful when writing parentheses. Alternatively, they may have learned to be more thorough when reading code.

Students in Fall A did significantly better than those in Spring A. They were more likely to give the correct answer $(p<.05)$ and more likely to notice that negate-all sometimes returned a sentence ("Correct" + "No Problem" + "Depends", also $p<.05$ ). Students in Summer A appeared to do about as well, but comparisons are not statistically significant because only eleven students answered this question.

|  |  | Answers by Category for Negate-all (\#, \%) |  |  |  |  |  |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- |
|  |  |  |  | Predicted |  |  |  |
| Semester | \#Students | Correct | Sent | Not Notice | No Problem | Depends | Other |
| SpA Sents | 51 | $10,20 \%$ | $12,24 \%$ | $16,31 \%$ | $7,14 \%$ | $5,10 \%$ | $1,2 \%$ |
| SpA Lists | 17 | $8,47 \%$ | $3,18 \%$ | $5,29 \%$ | $0,0 \%$ | $1,6 \%$ | $0,0 \%$ |
| SuA Sents | 11 | $4,36 \%$ | $2,18 \%$ | $3,27 \%$ | $1,9 \%$ | $1,9 \%$ | $0,0 \%$ |
| SuA Lists | 6 | $3,50 \%$ | $1,17 \%$ | $1,17 \%$ | $0,0 \%$ | $0,0 \%$ | $1,17 \%$ |
| FaA Sents | 93 | $32,34 \%$ | $10,11 \%$ | $23,25 \%$ | $15,16 \%$ | $8,9 \%$ | $5,5 \%$ |
| FaA Lists | 26 | $11,42 \%$ | $6,23 \%$ | $8,31 \%$ | $0,0 \%$ | $1,4 \%$ | $0,0 \%$ |
| SpB Sents | 30 | $4,13 \%$ | $3,10 \%$ | $11,37 \%$ | $10,33 \%$ | $2,7 \%$ | $0,0 \%$ |
| SpB I4 Lists | 7 | $3,43 \%$ | $2,29 \%$ | $1,14 \%$ | $0,0 \%$ | $1,14 \%$ | $0,0 \%$ |

Table 4.5: Semester summary for O5, deciding whether negate-all works correctly. Semesters are written as SpA for Spring A, SuA for Summer A, FaA for Fall A, and SpB for Spring B, followed by either "Sents" for all of the sentence questions from all interviews or "Lists" for the list questions from the final interview.

Why does (divide-by-largest '( $\left.\begin{array}{lllllll}1 & 2 & 3 & 4 & 3 & 2 & 1\end{array}\right)$ ) result in an error? Students were given one of two scenarios:

1. Find the error based on a procedure definition. If unsuccessful, find it using test cases.
2. Find the error based on test cases. If unsuccessful, find it using code.

Figure 4.3: Summary of Question O6, divide-by-largest.

## O6: divide-by-largest

Students trying to find the bug in the divide-by-largest question (O6, page 46) made a number of different mistakes. Virtually every line of the code was, at some point over the three semesters, blamed for the error. However, interviewers were instructed to correct students when they appeared confused by the code. In Spring A, students were more likely to find the bug with test cases than with code. Overall, students were about equally likely to find the error using either only the test cases or only the code. In the final round of interviews every semester, all students were given the test cases first. This was done in part because there were not enough students to divide them both by lists/sentences and by tests/code and in part because the decision was made after the second interview in Spring A, when the results suggested that the tests were at least as easy to interpret as the code. Most students in these interviews found the problem with just the test cases, so few even saw the code.

Table 4.6: Numbers and Percentages students who saw and correctly answered the divide-by-largest question using test cases. This table represents only part of the students who answered O6. For the rest, see Table 4.6. Categories are the total numbers of students interviewed ("\# Students"); numbers and percentages of people who solved the problem in any way; numbers and percentages of people who saw the test cases in the course of trying to solve the problem ("Total" under "Saw Tests"), who saw the test cases first (" 1 st" under "Saw Tests"), and who saw the test cases after having seen the code ("2nd" under "Saw Tests"); and numbers and percentages of all people who solved the problem using test cases ("Lotal under "Got Using Tests"), who solved the problem using only test cases "Only" under "Got Using Tests"), and people who solved the problem using test cases after they had seen code ("2nd" under "Got Using Tests"). Interviews are interview in each semester was divided into sentence questions ("Sents") and list questions ("Lists").

| Semester | \# Students | \# Correct, \% Correct | Saw Code |  |  | Got Using Code |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  | $\begin{aligned} & \text { \# Total, } \\ & \text { \% Total } \end{aligned}$ | $\begin{aligned} & \# 1^{\text {st }} \\ & \% 1^{\text {st }} \end{aligned}$ | $\begin{aligned} & \# 2^{\text {nd }} \\ & \% 2^{\text {nd }} \end{aligned}$ | $\begin{aligned} & \text { \# Total, } \\ & \text { \% Total } \end{aligned}$ | $\begin{aligned} & \text { \# Only, } \\ & \text { \% Only } \end{aligned}$ | $\begin{aligned} & \# 2^{\text {nd }} \\ & \% 2^{\text {nd }} \end{aligned}$ |
| SpA 12 | 34 | 19, $56 \%$ | 25, $74 \%$ | 16, 47\% | 9, $26 \%$ | 6, 24\% | 5, 31\% | 1, 11\% |
| SpA I3 Sents | 17 | 15, $88 \%$ | 3, 18\% | - | 3, 18\% | 1, $33 \%$ | - | 1, $33 \%$ |
| SpA 13 Lists | 17 | 16, 94\% | 3, 18\% | - | 3, 18\% | 2, $67 \%$ | - | 2, $67 \%$ |
| SuA I2 | 6 | 4, $67 \%$ | 3, 50\% | 2, $33 \%$ | 1, 17\% | 1, $33 \%$ | 1, $50 \%$ | 0, $0 \%$ |
| SuA I3 Sents | 5 | 5,100\% | 0, 0\% | - | - | - | - | - |
| SuA I3 Lists | 6 | 5, $83 \%$ | 3, 50\% | - | 3, 50\% | 2, $67 \%$ | - | 2, $67 \%$ |
| FaA I2 | 34 | 21, $62 \%$ | 23, $68 \%$ | 13, 38\% | 10, $29 \%$ | 11, $48 \%$ | 10, $77 \%$ | 1,10\% |
| FaA I3 | 34 | 31, $92 \%$ | 22, $65 \%$ | 18, $53 \%$ | 4, 12\% | 17, $77 \%$ | 14, 78\% | 3, $75 \%$ |
| FaA I4 Sents | 25 | 18, $72 \%$ | 8, $32 \%$ | - | 8, $32 \%$ | 1, $13 \%$ | - | 1, 13\% |
| FaA I4 Lists | 26 | 17, $65 \%$ | 11, $42 \%$ | - | 11, $42 \%$ | 2, 18\% | - | 2, 18\% |
| SpB I2 | 20 | 14, 70\% | 14, 70\% | 9, $45 \%$ | 5, $25 \%$ | 7, 50\% | 5, 56\% | 2, 40\% |
| SpB I4 Sents | 10 | 9, 90\% | 2, 20\% | - | 2, 20\% | 1,50\% | - | 1,50\% |
| SpB I4 Lists | 7 | 7, 100\% | 0, $0 \%$ | 0, 0\% | 0, 0\% | - | - | - |

Table 4.7: Numbers and Percentages students who saw and correctly answered the divide-by-largest question using code. This table represents only part of the students who answered O6. For the rest, see Table 4.6. Categories are the total numbers of students interviewed ("\# Students"); numbers and percentages of people
who solved the problem in any way; numbers and percentages of people who saw the code in the course of trying to solve the problem ("Total" under "Saw Code"), who saw the code first (" 1 st" under "Saw Code"), and who saw the code after having seen the test cases (" 2 nd" under "Saw Code"); and numbers and percentages of all people who solved the problem using code ("Total" under "Got Using Code"), who solved the problem using only code ("Only" under "Got Using Code"), and people who solved the problem using code after they had seen test cases ("2nd" under "Got Using Code"). Interviews are written as the semester "SpA" for Spring A, "SuA" for Summer A, "FaA" for Fall A, and "SpB" for Spring B; followed by " I " and the interview number. The final interview in each semester was divided
into sentence questions ("Sents") and list questions ("Lists").

Table 4.6 shows the total number of students in the interview ("\# Students"); numbers and percentages of students who found the bug using either tests or code ("Correct"); as well as the numbers and percentages of students who saw the test cases while solving the problem("Total" under "Saw Tests"), who were given the test cases first (" 1 st" under "Saw Tests"), and who saw the test cases only after they had tried and failed to solve the problem with code ("2nd" under "Saw Tests"); and the percentages of each of these groups who actually solved the problem while using test cases ("Total," "Only," and "2nd" under "Got Using Tests"). Table 4.7 shows in the same format the results for students using code. Both tables show the total numbers of students who answered and the total numbers and percentages who got the right answer, but the remaining columns deal only with the students who saw the tests or code, respectively. In general, students who did not find the bug the first time were unlikely to find it when given all of the information.

There were three keys to finding the error in the divide-by-largest problem. The first was realizing that sent-max ${ }^{2}$ returned a one-word sentence instead of a number. The second was noticing that the result from sent-max was used in arithmetic. The third was realizing that a one-word sentence would cause an error if it was used in arithmetical operations (in particular, that dividing a number by a sentence would be an error).

The way to see that sent-max produced a number depended on whether students were given test cases or code. When given test cases, students had to notice the parentheses around the numbers that sent-max returned and recognize that these meant "sentence." When given code, students had to notice that the code said "If there is exactly one number left in the sentence, return the whole sentence." This was written in the code as ( $(=$ (count sent) 1) sent). Students had to recognize that sent meant a sentence and not the word, and that Scheme would not convert the sentence to a word before finishing sent-max.

The way to see that the sentence returned by sent-max would be used in a numerical calculation and would thus lead to an error message was the same whether students were

[^7]given tests or code. First, students had to remember that sent-max returned a sentence as they read through the code to divide-sentence-by. Second, they had to know that Scheme would not convert a one-number sentence to a number even if it might be convenient, and that a sentence with only one number inside did not work just like that number.

Almost all of the students who demonstrated that they had noticed sent-max returning sentences eventually decided that dividing by that sentence would cause an error message. After the interview was over, students who had not found the error were shown the test cases again and asked what (sent-max '(12lllll $\left.\boldsymbol{1}_{1}^{2} \begin{array}{lllll}3 & 4 & 3 & 2 & 1\end{array}\right)$ should return. They all said it should return "four." Then they were asked what it actually returned. Some noticed the sentence at this point, but others said it would return "four." When these students were asked to look again, they all noticed that it was a sentence. When asked whether they had noticed the sentence before, a few said that they had but had not considered it important. The majority said that they had not noticed the parentheses. However, because they were asked to remember whether they had noticed something they considered unimportant, the reliability of their answers is suspect.

The reason many students gave for having missed the sentences was that they were only looking at what they called "the answer" (the number). This explanation was common through the semesters, both on this problem and on others. This suggests that, whether or not students believe in one-word sentences, many of them believe that the values of the words or numbers are far more important than the quality of "being in a sentence." This is not entirely surprising, as it usually takes more code to pick the right words or calculate the right numbers than it does to put them in a sentence.

The "Correct" columns in Tables 4.6 and 4.8 show a general increase in the frequency of right answers both within Spring A and between Spring A and later semesters. Students were $57 \%$ more likely to solve this problem at the end of Spring A than they were at the beginning; this increase is significant ( $p<.01$ ). Students in semesters after Spring A were not significantly more likely to identify the error when given tests than were students in

| Semester | \# Students | \# Correct, <br> \% Correct | Tests |  | Code |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  | $\begin{aligned} & \text { \#Saw, } \\ & \text { \% Saw } \end{aligned}$ | \# Correct, <br> \% Correct | $\begin{aligned} & \text { \#Saw, } \\ & \text { \% Saw } \end{aligned}$ | \# Correct, <br> \% Correct |
| SpA Sents | 51 | 34, $67 \%$ | 46, 90\% | 27, 59\% | 28, $55 \%$ | 7, 25\% |
| SpA Lists | 17 | 16, 94\% | 17, 100\% | 14, $82 \%$ | 3, 18\% | 2, $67 \%$ |
| SuA Sents | 11 | 9, $82 \%$ | 10, 91\% | 8, 80\% | 3, $27 \%$ | 1, 33\% |
| SuA Lists | 6 | 5, 83\% | 6,100\% | 3, $50 \%$ | 3, 50\% | 2, $67 \%$ |
| FaA Sents | 93 | 78, $84 \%$ | 65, 70\%\% | 46, 71\% | 53, $57 \%$ | 29, $55 \%$ |
| FaA Lists | 26 | 17, 65\% | 26, 100\% | 15, 58\% | 11, $42 \%$ | 2, 18\% |
| SpB Sents | 30 | 23, $77 \%$ | 25, $83 \%$ | 15, 60\% | 16, 53\% | 8, 50\% |
| SpB Lists | 7 | 7, 100\% | 7, 100\% | 7, 100\% | 0, 0\% | 0, - |

Table 4.8: Semester summary for O6, finding the error in the divide-by-largest procedure. Semesters are written as SpA for Spring A, SuA for Summer A, FaA for Fall A, and SpB for Spring B, followed by either "Sents" for all of the sentence questions from all interviews or "Lists" for the list questions from the final interview.

Spring A, but students in semesters after Spring A were more likely to identify the error when given code. In fact, students given code in the second rounds of interviews for Fall A and Spring B were approximately twice as likely to find the error as were students given code in Spring A ( $p<.01$ for Fall A and $p<.05$ for Spring B). While students were not much more likely to notice that (4) was a sentence, they were more likely to notice that sent meant sentence. This difference is consistent with an increased emphasis after Spring A that sentences are not words and no additional emphasis that sentences need parentheses.

Students in Spring A also said they related parentheses in Scheme to parenthetical phrases in English. One said that because he did not think that parenthetical phrases were important in English, s/he did not think parentheses were important in Scheme: "When I read, I read for content. I don't remember, 'Oh, that part was in parentheses.' "3 Students in later semesters still ignored parentheses, but none was able to explain why.

## One-Word Sentence Subscale

I attempted to construct a subscale to measure students' overall understanding of oneword sentences using O1, O3, O5, and O6, as these were in every semester's interview. Only the combination of O1 and O3 were coherent (Cronbach's alpha $>.95$ ), and only after the Spring A semester. Unfortunately, there are no significant improvements in these scales

[^8]```
E1: What is (bf '(1))?
E2: What is (bl '(1))?
```

Figure 4.4: Questions E1 and E2, creating empty sentences.
from semester to semester. The general lack of coherence supports the idea that students are using everyday reasoning to answer these questions, because while these questions are all the same from a Scheme perspective, they are different from a real-world perspective.

### 4.2 Empty Words and Sentences

## E1 and E2: Creating Empty Sentences

Students' answers to questions about the creation of empty sentences (E1 and E2, described on page 49) are summarized in Table 4.9. The numbers and percentages of students who answered E1 and E2 correctly are given in the "Correct" columns. The numbers and percentages of students who missed E1 or E2 and said the result would be an error (the predicted mistake) are given in the "Error" columns. Few students actually missed E1. Most of the ones who did thought that (bf '(1)) would cause an error because there would be nothing left. Several of these claimed that they remembered having been told this, and the rest seemed to think it was logical.

Students who were asked questions O5 and O6 (negate-all and sent-max) had to know the right answer to trace those programs; only a very few of the students who thought that (bf '(1)) caused an error in E1 said that it would cause an error in O5 or O6. Most of the students who missed E1 never appeared to notice this when solving O5 or O6, but at least two students did notice and changed their answers to E1.

While at least $80 \%$ of the students in each round of interviews eventually reached the correct answer, most had to think about it first. These people were generally trying to decide between empty sentences and errors, just like the students who gave wrong answers.

| Semester | \# Students | (bf '(1)) |  | (bl ' (1)) |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | \# Correct, \% Correct | \# Said Error, \% Said Error | \# Correct, <br> \% Correct | \# Said Error, \% Said Error |
| SpA I1 | 37 | 34, 92\% | 3, 100\% | 30, 81\% | 6, 86\% |
| SpA 12 | 34 | 31, 91\% | 2, $67 \%$ | - | - |
| SpA 13 Sents | 17 | 15, $88 \%$ | 2, 100\% | - | - |
| SpA 13 Lists | 17 | 17, 100\% | - | - | - |
| SuA I1 | 12 | 10, $83 \%$ | 0, $0 \%$ | 9, $75 \%$ | 0, 0\% |
| SuA 12 | 6 | 6, 100\% | - | - | - |
| SuA I3 Sents | 5 | 5, 100\% | - | - | - |
| SuA I3 Lists | 6 | 6, 100\% | - | - | - |
| FaA I1 | 37 | 37, 100\% | - | 27, $73 \%$ | 0, $0 \%$ |
| FaA 12 | 34 | 33, 97\% | 1, 100\% | - | - |
| FaA 13 | 34 | 31, 91\% | 3, 100\% | - | - |
| FaA I4 Sents | 25 | 21, $84 \%$ | 2, $50 \%$ | - | - |
| FaA I4 Lists | 26 | 21, $81 \%$ | 2, $40 \%$ | - | - |
| SpB I1 | 19 | 14, 74\% | 1, $20 \%$ | 13, $68 \%$ | 1, $17 \%$ |
| SpB 12 | 20 | 20, 100\% | - | - | - |
| SpB I4 Sents | 10 | 9, $90 \%$ | 1, 100 | - | - |
| SpB I4 Lists | 7 | 6, 89\% | 1, 100\% | - | - |

Table 4.9: Numbers and percentages of students who gave the correct result for (bf '(1)) and (bl '(1)), as well as the percentage of those students who thought the expression would produce an error message. Interviews are written as the semester "SpA" for Spring A, "SuA" for Summer A, "FaA" for Fall A, and "SpB" for Spring B; followed by "I" and the interview number. The final interview in each semester was divided into sentence questions ("Sents") and list questions ("Lists").

Furthermore, students' explanations for this problem were different from their explanations of O1 and O3. When explaining their answers at the end of the interviews, students often gave a general explanation of how butfirst worked when discussing (bf '(12)) and said that the same concepts applied to (bf ' (1 22344$)$ ). However, less than one in five of these students said that the same concepts also applied to (bf '(1)). Instead, they gave specialized explanations of butfirst that applied only to one-word sentences. General explanations of butfirst were along the lines of "Butfirst takes a sentence, gets rid of the first thing, and returns the rest of the sentence." Specialized explanations were along the lines of "Butfirst takes a one-word sentence and, since there is only one word, it gives you back an empty sentence." It appears that the majority of these students had not seriously considered how butfirst might work on one-word sentences or, at least, did not think of it in the same way they thought of the butfirst of longer sentences.

Students had more trouble with E2, although not as many of them believed that the butlast of a one-word sentence would cause an error message. This appears to be due to confusion over the meaning of "last." While most of the students did give the correct
answer and most of the rest said (bl '(1)) would produce an error, students provided many additional answers and explanations. Some thought that the sentence (1) was composed of two elements, 1 and (). The first was 1 and the last was (), so the butlast would produce 1. These students may have reasoned that because the first was 1 , the butfirst was (), and neither could be broken down any more, (1) must be made of the elements 1 and (). Others thought that the 1 was followed by some kind of invisible space or empty word, and that butlast removed the blank space and left (1). Some thought that taking the butlast again would remove the 1 , since the blank space was now gone, but others did not. Several said that because there was no last, the sentence would be returned unchanged.

Like E1, even students who correctly said that (bl '(1)) was () struggled before reaching a conclusion. However, instead of trying to decide whether there was anything left after taking the butfirst, students tried to decide whether there even was a last element to get rid of. From this, it appears that students have very unexpected models for exactly what a sentence is and what last and butlast do.

At the start of the Fall B semester, I tried an impromptu experiment with a lab section of approximately 25 students. I asked one student to start a line at the door and then asked the rest of the class who the last person in line was. Students said they were not sure that a line with only one person could have a last person. Thus, students could be relying on more real-world models to understand what last and, by extension, butlast do. ${ }^{4}$

As Table 4.10 shows, students did about equally well on (bf '(1)) in every semester. The percentage of students who correctly answered (bl '(1)) dropped every semester, although not significantly. The only interesting change was that students in later semesters were much less likely to think that taking the butlast of a one-word sentence would result in an error message. In Spring A, one student said that the sentence (1) had blank space on the end that was removed by the butlast procedure, while the other six who gave the wrong answer expected an error message. Only one student in all of the later semesters

[^9]| Semester | \#Students | \# Correct, <br> \% Correct | \# Said Error, <br> \% Said Error |
| :--- | :--- | :--- | :--- |
| SpA Sents | 88 | $80,91 \%$ | $7,88 \%$ |
| SpA Lists | 17 | $17,100 \%$ | - |
| SuA Sents | 23 | $21,91 \%$ | $0,0 \%$ |
| SuA Lists | 6 | $6,100 \%$ | - |
| FaA Sents | 130 | $122,94 \%$ | $5,71 \%$ |
| FaA Lists | 26 | $21,81 \%$ | $2,40 \%$ |
| SpB Sents | 49 | $43,88 \%$ | $2,33 \%$ |
| SpB Lists | 7 | $6,89 \%$ | $1,100 \%$ |

Table 4.10: Semester summary for E1, taking the butfirst of a one-word sentence. Since E2 was only asked in the first round of interviews each semester, it was not included in this table. Interviews are written as the semester "SpA" for Spring A, "SuA" for Summer A, "FaA" for Fall A, and "SpB" for Spring B, followed by either "Sents" for all of the sentence questions from all of the interviews or "Lists" for the list questions from the final interview.

```
E3: What is (empty? '(""))?
E4: What is (first '(""))?
```

Figure 4.5: Questions E3 and E4, empty words and sentences.
said that (bl '(1)) should produce an error. This is not an improvement, since thinking that there is no last element or that there is invisible space at the end of a sentence is at least as bad as thinking that getting rid of the last element of a sentence causes an error. None of the interventions or changes in the course material focused on last or butlast; the reasons for this change are unknown.

## E3 and E4: Empty Words and Empty Sentences

Table 4.11 shows students' successes with questions E3 and E4. Students were asked to tell whether the sentence ("") was empty and what the first of it would be. During the first round of interviews in each semester, fewer than half of the students correctly stated that ("") was not empty (as seen in the first "Correct" column). However, students did improve on E4 over a semester. In Spring A, for example, students were $49 \%$ more likely to answer E4 correctly at the end of the semester than they were at the beginning.

| Semester | \#Students <br> empty? | first <br> \# Correct, | Correct, <br> \# Erst | Both <br> \# Error, | \# Right, <br> \# Empty Has First <br> \% Empty Has First, |  |
| :--- | ---: | ---: | ---: | ---: | ---: | ---: |
| SpA I1 | 37 | $18,49 \%$ | $19,51 \%$ | $8,44 \%$ | $13,35 \%$ | $6,16 \%$ |
| SpA I2 | 34 | $18,53 \%$ | $23,68 \%$ | $9,82 \%$ | $17,50 \%$ | $6,18 \%$ |
| SpA I3 Sents | 17 | $10,59 \%$ | $13,76 \%$ | $3,75 \%$ | $8,47 \%$ | $5,29 \%$ |
| SpA I3 Lists | 17 | $13,76 \%$ | $16,94 \%$ | $1,100 \%$ | $13,76 \%$ | $3,18 \%$ |
| SuA I1 | 12 | $7,58 \%$ | $6,50 \%$ | $3,50 \% \%$ | $5,42 \%$ | $2,17 \%$ |
| SuA I2 | 6 | $5,83 \%$ | $5,83 \%$ | $0,0 \%$ | $5,83 \%$ | $1,17 \%$ |
| SuA I3 Sents | 5 | $5,100 \%$ | $5,100 \%$ | - | $5,100 \%$ | $0,0 \%$ |
| SuA I3 Lists | 6 | $5,83 \%$ | $5,83 \%$ | $1,100 \%$ | $5,83 \%$ | $0,0 \%$ |
| FaA I1 | 37 | $13,35 \%$ | $28,76 \%$ | $3,33 \%$ | $12,32 \%$ | $16,43 \%$ |
| FaA I2 | 34 | $20,59 \%$ | $26,76 \%$ | $8,100 \%$ | $19,56 \%$ | $7,21 \%$ |
| FaA I3 | 34 | $28,82 \%$ | $27,79 \%$ | $4,57 \%$ | $25,74 \%$ | $2,6 \%$ |
| FaA I4 Sents | 25 | $19,76 \%$ | $21,84 \%$ | $1,25 \%$ | $18,72 \%$ | $3,12 \%$ |
| FaA I4 Lists | 26 | $21,81 \%$ | $24,92 \%$ | $2,100 \%$ | $21,81 \%$ | $3,12 \%$ |
| SpB I1 | 19 | $7,37 \%$ | $11,58 \%$ | $0,0 \%$ | $7,37 \%$ | $4,21 \%$ |
| SpB I2 | 20 | $17,85 \%$ | $17,85 \%$ | $1,33 \%$ | $18,80 \%$ | $1,5 \%$ |
| SpB I4 Sents | 10 | $9,90 \%$ | $10,100 \%$ | - | $9,90 \%$ | $1,10 \%$ |
| SpB I4 Lists | 7 | $4,57 \%$ | $6,86 \%$ | $1,100 \%$ | $4,57 \%$ | $2,29 \%$ |

Table 4.11: Numbers and percentages of students who gave the correct results for (empty? ("")) and (first '("")), as well as of students who said that the answer to the second should be an error, who correctly answered both questions, and who gave an incorrect but predicted set of answers ("Empty Has First"). The percentages in the "Error" column are of students who got the question wrong, not of all students. Interviews are written as the semester "SpA" for Spring A, "SuA" for Summer A, "FaA" for Fall A, and "SpB" for Spring B; followed by "I" and the interview number. The final interview in each semester was divided into sentence questions ("Sents") and list questions ("Lists").

Students did better when asked what the first of it would be (as seen in the second "Correct" column). As shown in the "Both Right" column, few of the students in the Spring A semester or in the early interviews of other semesters answered both questions correctly.

The "Error" column shows the numbers and percentages of students who missed E4 and said it would produce an error message. Most of the students who said it this also said that the sentence was empty, so they reasoned (correctly) that taking the first of an empty sentence should cause an error message. While a majority of these students (correctly) thought that the empty word remained in the empty sentence, some believed that the empty word actually disappeared.

Some students who said that the sentence was not empty still said that taking the first of it would produce an error message. These students seemed to believe that while the empty word was somehow good enough to fit in a sentence, it was not good enough to stand on its own.

Approximately one student in five gave the predicted combination of answers, that (" ") is empty but it has "" as its first element (the "Empty Has First" column). Nearly every student who gave these two answers explained that, while they knew there was something inside the sentence (and thus they could take the first), that something was not enough to make the sentence non-empty. An alternate explanation given by a very few students was that taking the first of any empty sentence produced an empty word, either because empty sentences always contain hidden empty words, or there is nothing in the sentence and thus the first must be nothing (which they wrote as an empty word). Students who believed that ("") had something in it but was still empty usually said that empty? was looking for letters or "real" words in the sentence, and the empty word had no letters and was not, by their definition, a real word.

Some students, even those in later rounds of interviews, had trouble with these two problems because they did not know that "" was an empty word. Instead, they assumed it was a word made of two " characters. All but one of these students knew what an empty word was, but they did not think of it when reading the problems. These students gave the correct answers, that ("") was not empty and that the first was " ". Interviewers pointed out the empty word and nearly all of these students changed their minds, at least on E3. They justified this in the same way that the other students did, saying that the empty word was not a real word.

The percentage of students who missed E4 and said that (first '("")) produced an error fluctuated wildly after Spring A. In three rounds of interviews, every student who missed E4 said the answer should cause an error message. However, in five other rounds of interviews, half or fewer of the students said this. These changes appear to be due to several things. First, fewer students missed E4. Second, those who did tended to be extremely lost. Approximately half of the students who missed E4 did so because they did not know that "" was the empty word or because they did not understand how first worked. The two answers given by those confused by first were that (first '("")) returns " or ("). These
students believed that first operated on the letters of a sentence rather than the words of a sentence, so (first '(ab)) should return either a or (a). However, most of the students in the fourth round of interviews in Fall A who missed E4 did so because they were not sure how to write that (first '("")) should return an empty word rather than because they had the wrong kind of answer. Students in other semesters who were not lost tended to have missed E3, believing that the sentence was empty, and thus that taking the first would cause an error message.

In general, students improved after the Spring A semester. Students in the first round of interviews each semester did worse on E3, but they made improvements as the semester went on. Scores on both questions regularly increased from semester to semester (see Table 4.12). For E3, students in Summer A and Spring B outscored those in Spring A ( $p<.05$ for both), and those in Fall A may have ( $p<.1$ ). For E4, students in Fall A and Spring B outscored those in Spring A ( $p<.01$ for Fall A and $p<.05$ for Spring B); scores were not significantly different between Spring and Summer A. Students in Fall A and Spring B were, of course, more likely to get both questions right than were students in Spring A ( $p<.05$ for Fall A and $p<.01$ for Spring B), and those in Spring B may have been more likely to get both correct than students in Fall A $(p<.1)$. However, the percentage of students who missed E3 but got E4 did not change significantly across semesters.

Several changes to the curriculum may have enhanced student understanding. The $\mathrm{Pez}^{\circledR}$ analogy gave students a model that suggested that empty words and sentences were actual things and had some sort of value. The number-spelling project forced many students to deal with empty words in sentences. WebScheme activities in Spring B interrupted students who answered questions about empty words or sentences with "nothing" or who left the answers blank, encouraging them to talk to TAs or lab assistants and making them confront the idea that empty words and sentences do not just vanish. It seems likely that some combination of these activities contributed to student understanding.

| Semester | \#Students | empty? <br> \#Correct, | first <br> \# Correct, | first <br> \# Error, | Both <br> \# Right, | \# Empty Has First <br> \# Empty Has First, |
| :--- | ---: | ---: | ---: | ---: | ---: | ---: |
| SpA Sents | 88 | $46,52 \%$ | $55,63 \%$ | $20,61 \%$ | $38,43 \%$ | $17,19 \%$ |
| SpA Lists | 17 | $13,76 \%$ | $16,94 \%$ | $1,100 \%$ | $13,76 \%$ | $3,18 \%$ |
| SuA Sents | 23 | $17,74 \%$ | $16,70 \%$ | $3,43 \% \%$ | $15,65 \%$ | $3,13 \%$ |
| SuA Lists | 6 | $5,83 \%$ | $5,83 \%$ | $1,100 \%$ | $5,83 \%$ | $0,0 \%$ |
| FaA Sents | 130 | $80,62 \%$ | $102,78 \%$ | $16,57 \%$ | $74,57 \%$ | $28,22 \%$ |
| FaA Lists | 26 | $21,81 \%$ | $24,92 \%$ | $2,100 \%$ | $21,81 \%$ | $3,12 \%$ |
| SpB Sents | 49 | $33,67 \%$ | $38,78 \%$ | $1,9 \%$ | $34,69 \%$ | $6,12 \%$ |
| SpB Lists | 7 | $4,57 \%$ | $6,86 \%$ | $1,100 \%$ | $4,57 \%$ | $2,29 \%$ |

Table 4.12: Semester summary for questions E3 and E4, (empty? '("")) and (first '("")). The percentages in the "Error" column are of students who got the question wrong, not of all students. Interviews are written as the semester "SpA" for Spring A, "SuA" for Summer A, "FaA" for Fall A, and "SpB" for Spring B, followed by either "Sents" for all of the sentence questions from all interviews or "lists" for the list questions from the final interview.

| Mistake | Least Serious <br> $(\#, \%)$ | Most Serious <br> $(\#, \%)$ |
| :--- | :--- | :--- |
| (eleven thousand zero zero zero) | $7,21 \%$ | $10,29 \%$ |
| (eleven thousand zero) | $3,9 \%$ | $3,9 \%$ |
| (eleven thousand "") | $26,76 \%$ | $2,6 \%$ |
| (ten one thousand) | $4,12 \%$ | $20,59 \%$ |

Table 4.13: Mistakes for the number-spell program and the numbers and percentages of students who thought each was least or most serious. This data is from the 34 students in the second round of interviews in Spring A.

## E5: number-spell and Empty Words in Sentences, Spring A

Table 4.13 shows which of the four possible mistakes for number-spell (page 51) students found to be least and most serious for students in Spring A. While the table only shows the results of the 34 students in Spring A, five students rated several mistakes as either most or least serious. Three of the six students in the second round of interviews in Summer A were also asked this question.

All three Summer A students felt that (ten one thousand) was the most serious mistake. One student said that all of the rest of the mistakes were equally minor. Another initially said (eleven thousand "") was the least serious but decided (eleven thousand zero) was equally minor. The third said that (eleven thousand "") was the least serious, since it could be easily fixed by replacing the empty word with an empty sentence. The remainder of this section will focus on the Spring A students.

Nearly three-quarters of the students felt that (eleven thousand "") was the least serious. Over over half of the students felt that the most serious mistake was (ten one
thousand). About a third felt that the worst mistake was was (eleven thousand zero zero zero). Only two students felt that the worst mistake was (eleven thousand ""). Just three students said that (ten thousand zero) was the least bad; one of those said it was about equal to (eleven thousand "") and another said that everything except (eleven thousand zero) was equally okay.

One of the two students who felt that the extra empty word was the worst mistake explained that $\mathrm{s} / \mathrm{he}$ did not see any reason for an empty word to be a part of the code, so whoever wrote the code must have gone out of the way to add it. The second said that zeroes, unlike the empty word, have absolutely no value.

Thirty-eight percent of those who thought that (eleven thousand "") was the least serious mistake explained that it should be the easiest to fix. Thirty-one percent said the empty word was not really anything - it "doesn't represent anything," "says there is nothing after 'eleven thousand,'" "knows not to say anything," is "not really there," is a "superficial error," or is just junk after the right answer. Twelve percent said that the empty word was less confusing to look at, either because it was clearly not a number or because it could be ignored while reading the sentence.

The first reason, that the empty word is easier to fix, might also apply to (eleven thousand zero). However, only three students said that (eleven thousand "") was one of the least serious errors, and a total of five students in the entire interview said that the extra zero and the extra empty word were about the same. Seven of the ten students who thought that the empty word was easier to fix gave reasonable answers - it should be easier to search the code for " " than for zero; it should be easier to go through the sentence and remove " ", since while it is possible for zero to belong in the sentence, it is not possible for the empty word to belong there; or the code could be modified to return the empty sentence instead (the best solution). These are all justifiable reasons to pick the empty word as being caused by the least serious bug. The other three said that it would be easy to make it go away, but they had no idea why.

```
Here is something that you might have been asked to do in homework or on a test:
Write a procedure called sum-of-square-roots, which takes two numbers, x and y, finds the
square root of each, and returns the sum of those square roots. }x\mathrm{ and }y\mathrm{ will not be negative
numbers.
One student writes this as an answer on the test:
(define (sum-of-square-roots x y)
    (if (or (< x 0) (< y 0)) ;;line 1
        '() ;;line 2
        (+ (sqrt x) (sqrt y)))) ;;line 3
Why did this person include lines 1 and 2?
```

Figure 4.6: Question E6, empty words as errors.

Forty-two percent of all of the students who said that the empty word was one of the least serious errors, half of the students who felt that it was the single least serious error, or approximately one-third of all of the students in the interview, felt that the empty word would vanish, know "not to say anything," not really be there, or would somehow be easier to remove than the extra zero.

Thirty-five percent of the students who said that the empty word was least serious made it clear that they were not being forced to choose between it and the extra zero. One-third of these students rated another mistake as equally minor. The other two-thirds made it clear either by explicitly comparing the two or by ranking the four answers and putting another answer between the empty word and the extra zero. Of those who explicitly compared the two, one-third said it was because the empty word was less confusing to read, while the rest thought the empty word would vanish or otherwise be unimportant.

## E6: Empty Sentences as Errors

None of the students who were asked this question had a ready explanation. When given some time to think, they all decided that the empty sentence was used to signal an error, but they were still unable to explain why the empty sentence might be a logical choice to convey this.

| Semester | \#Students | \# Correct, <br> \% Correct | \# Forgot, <br> \% Forgot | \#Vanishes, <br> \% Vanishes | \#Sentence, <br> \% Sentence | \# Error, <br> \% Error |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- |
| FaA I3 | 34 | $19,56 \%$ | $3,9 \%$ | $8,24 \%$ | $2,6 \%$ | $0,0 \%$ |
| FaA I4 Sents | 25 | $13,52 \%$ | $1,4 \%$ | $5,20 \%$ | $1,4 \%$ | $4,16 \%$ |
| FaA I4 Lists | 26 | $13,50 \%$ | $1,4 \%$ | $2,8 \%$ | 5,19 | $2,8 \%$ |
| SpB I4 Sents | 10 | $7,70 \%$ | $0,0 \%$ | $1,10 \%$ | $1,10 \%$ | $1,10 \%$ |
| SpB I4 Lists | 7 | $3,43 \%$ | $0,0 \%$ | $0,0 \%$ | $4,57 \%$ | $0,0 \%$ |

Table 4.14: Numbers and percentages of students who gave the correct result for (every bf ' (1 2434$)$ ) as well as for those who gave certain kinds of incorrect answers. Not all student responses are counted here, since some mistakes did not fit into any of these categories. Interviews are written as the semester "FaA" for Fall A "SpB" for Spring B, followed by "I" and the interview number. The final interview in each semester was divided into sentence questions ("Sents") and list questions ("Lists").

E7: What is (every bf '(1 22333 4444))?

Figure 4.7: Question E7, placing an empty word in a sentence.

## E7: Placing an Empty Word in a Sentence

Table 4.14 summarizes the results of question E7. More than half of all students got the right answer, that (every bf ' (1 22333 4444) ) produces ("" 233 444). There were three general categories of incorrect answers. First, students left out the empty word (the "Forgot" and "Vanishes" columns). Second, students said that the butfirst of 1 was the empty sentence rather than an empty word ("Sentence"). Third, students said that taking the butfirst of 1 would produce an error ("Error").

Students who said that (every bf ' (1 22333 4444) ) produced (2 33 444) gave one of three reasons. Some students said they forgot. Some incorrectly thought that (bf 1) produced an empty sentence, but they correctly reasoned that the empty sentence would not appear in the final product. Three said that (bf 1) was literally nothing, so it would have no contribution to the final product. Most of the students who omitted the empty word correctly said that (bf 1) produces an empty word, but incorrectly said it would not appear in the final product.

Most of the students who thought that empty words disappeared said they thought empty words vanished because they worked just like empty sentences. When asked, most of these students could not explain why empty sentences did not appear in sentences. A few students who thought that empty words disappeared said that empty words were nothing.

P1: What do you think are the essential parts of an English sentence?
P2: What do you think are the essential parts of a Scheme sentence?

Figure 4.8: Questions P1 and P2, essential elements of English and Scheme sentences.

A few others said that empty words were actively removed when sentences were constructed. The rest of the students could not explain why empty words disappeared.

## Empty Word/Sentence Subscale

I attempted to construct a subscale to measure students' overall understanding of empty sentences and words using E1, E3, and E4, as these were in every interview. Unfortunately, no combination of these items was coherent (Cronbach's $\alpha<.8$, and usually $<.5$ ). Again, this supports the idea that students are relying on everyday knowledge, since none of these questions would be related in the real world.

### 4.3 Essential Elements of English and Scheme Sentences

Table 4.15 shows the percentages of students in each round of interviews who included starting with capital letters or ending with punctuation among the essential elements of English sentences, as well as the percentages of students who included parentheses among the essential elements of Scheme sentences. The three dichotomous variables "Capital," "Punctuation," and "Parentheses" could not be used to form a consistent subscale (Cronbach's $\alpha<.7$ ), and students who gave these responses did no better on questions O1-O6 (bf and bl of one-, two-, and four-word sentences) than anyone else (Chi-square tests showed $p>.1$ in all cases). These students may have had some association between sentences and parentheses in their minds, but they had not integrated their knowledge to the point that they automatically treated parentheses as delimiters for sentences.

The fourth column shows the percentages of students who explicitly compared or contrasted English and Scheme sentences. Available data probably underestimates the number

| Semester | \# Students | English |  | Scheme | \# Compare, <br> \% Compare | \# Confuse, <br> \% Confuse |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  | \# Capital, <br> \% Capital | \# Punctuation, \% Punctuation | \#Parentheses, \%Parentheses |  |  |
| FaA I1 | 37 | 2, 5\% | 15, 41\% | 27, $73 \%$ | 3, $8 \%$ | 5, 14\% |
| FaA I2 | 34 | 7, $21 \%$ | 17, $50 \%$ | 23, $68 \%$ | 2, $6 \%$ | 3, $9 \%$ |
| FaA I3 | 34 | 1, $3 \%$ | 10, $29 \%$ | 20, $59 \%$ | 3, $9 \%$ | 2, 6\% |
| FaA I4 Sents | 25 | 4, 16\% | 11, $44 \%$ | 17, $68 \%$ | 6, $24 \%$ | 5, $20 \%$ |
| FaA I4 Lists | 26 | 5, 19\% | 8, 31\% | 14, $54 \%$ | 1, $4 \%$ | 3, 12\% |
| SpB I1 | 19 | 0, $0 \%$ | 4, $21 \%$ | 14, $74 \%$ | 2, 11\% | 9, $47 \%$ |
| SpB I2 | 20 | 2, 10\% | 10, $50 \%$ | 17, $85 \%$ | 6, 30\% | 4, $20 \%$ |
| SpB I4 Sents | 10 | 1, 10\% | 4, 40\% | 2, $20 \%$ | 1, 10\% | 3, 30\% |
| SpB I4 Lists | 7 | 1, 14\% | 5, $71 \%$ | 4, $57 \%$ | 1, 14\% | 1, 14\% |

Table 4.15: Numbers and percentages of students who mentioned capital letters or punctuation as essential elements of English sentences, parentheses as essential elements of Scheme sentences, who explicitly compared or contrasted English and Scheme, or who confused Scheme sentences and code. Interviews are written as
"FaA" for Fall A and "SpB" for Spring B, followed by "I" and the interview number. The final interview in each semester was divided into sentence questions ("Sents") and list questions ("Lists").
of students who actually compared or contrasted English and Scheme. Many students gave answers that sounded like they might have been contrasts, but because interviewers did not ask follow-up questions, only those students who wrote down explicit comparisons or contrasts were counted. The data does show that very few students actually compared English and Scheme.

The final column shows the percentages of students who thought that a "Scheme sentence" meant a "Scheme procedure" or "Scheme code." These students were told by the interviewers that a "Scheme sentence" was intended to mean "the kind of data in Scheme that they had been calling a 'sentence' for the entire semester;" some of these students then changed their answers. One student commented that it seemed more logical for a sentence to refer to actual Scheme code, while others made a number of connections between English sentences and Scheme code (procedures as verbs, rules of grammar and syntax, etc.)

### 4.4 What Students Did Not Say About English, Scheme, and Collections

Students mentioned most inferences given in Tables 3.7 and 3.8, so this section will discuss those they mentioned rarely or actively disagreed with. Students had trouble understanding "meaning" in terms of Scheme sentences. Students answering P1 and P2 often said that English sentences had meaning, but Scheme sentences did not. They said that the order in an English sentence was governed by the rules of grammar, while Scheme sentences could be in any order. One student, on the other hand, thought that Scheme sentences had more meaning and order than English sentences, since the precise value and position of each word was important to procedures that operated on sentences.

Students did not treat words as being made of one-letter words or as being collections of letters in the same way sentences are collections of words. Students expressed some confusion over this idea when it was mentioned in lab.

|  | Percent Correct (Sentences/Lists) |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Semester | O1 | O3 | O5 | O6 | E1 | E3 | E4 | E7 | Mean |
| Spring A | 94/100 | 100/100 | 29/47 | 88/94 | 88/100 | 59/76 | 76/90 | -/- | 76/87 |
| Summer A | 100/67 | 100/83 | 60/60 | 100/83 | 100/100 | 100/100 | 100/83 | -/- | 94/79 |
| Fall A | 96/82 | 96/92 | 44/42 | 72/65 | 84/81 | 76/81 | 84/92 | 52/50 | 79/77 |
| Spring B | 100/86 | 100/100 | 20/43 | 90/100 | 90/86 | 90/57 | 100/86 | 70/43 | 84/80 |

Table 4.16: Percentages of students who got each question correct on the sentences/lists questions in the final round of each semester, as well as the average score for all but E7.

Students did not mention one-word sentences on questions P1 and P2, and they generally avoided talking about one-word English sentences.

Students did not compare or contrast words and sentences, but when they were asked in discussion or in lab, they did not see many similarities between the two. In particular, they did not feel that sentences were composed of words in the same way that words were composed of one-letter words.

### 4.5 Comparison of Sentences and Lists

Table 4.16 shows the differences in total scores for students given sentence questions and students given list questions in the final round of interviews each semester. The first eight columns show the percentages of students who got each of the eight questions (seven in Spring A) correct, while the last column shows the average over the seven questions (O1, O3, O5, O6, E1, E3, and E4) that were on all three rounds of interviews.

Students in the Spring A semester did better with lists than with sentences on all questions. When students were given one point for each correct answer, the average score for students answering sentence questions was 6.1, while the average score for students answering list questions was 6.94. The average score for students answering list questions was $14 \%$ higher than for students answering sentence questions. A one-way ANOVA confirmed that this was significant ( $p<.05$ ).

Students in the Spring A semester had the least practical experience with lists out of all the semesters. They had seen a basic introduction to them in lab, had studied lists for the third midterm, and had been given further list activities several weeks later. However, the further list activities were given as lab assignments during final projects, and very few students even looked at them until projects were complete. Furthermore, their projects did not make use of lists, so all of their experience with lists came from one week in lab and their own studying. Many of these students claimed not to be very familiar with lists-they had to remind themselves of what car, cdr, and null? meant, and they usually resolved their confusion by comparing a list procedure to the equivalent sentence procedure. Whether or not these students were comfortable and familiar with lists, they had been working with sentences far longer than with lists, and their most recent midterm included more questions that dealt with sentences than with lists. The significantly higher scores for list questions strongly suggest that they thought about sentences and lists in different ways. ${ }^{5}$

Several of these students admitted this. One student became upset with the (empty? '("")) question, saying that it was a silly corner case like the ones on the exams. When asked what (null? '(())) was, he said it was obviously not null. Then he looked at the two questions more carefully, said, "Oh!" and began asking about the goals of the interviews. Students in the other semesters did not appear to make this distinction.

In later semesters, most or all of the list activities were given in lab before the projects started, and the projects made use of lists. Students in these semesters had actual hands-on experience using lists. However, scores on sentence questions increased relative to scores on list questions in the later semesters. Students in the Fall A semester did better with sentences than with lists on all questions except E3 and E4, which dealt with (""), but

[^10]none of these differences was significant. Students in Spring B did better with sentences than with lists for all questions except O5 and O6. While only two students answering the sentence questions correctly said that the negate-all code did not work, the majority at least noticed that it did not behave consistently; most of the students answering list questions did not notice this. Only one student answering sentence questions missed O6, divide-by-largest, while none of the students answering list questions did so. It is thus not unreasonable to say that students in Spring B did at least as well on sentence questions as they did on list questions. They did at least as well as students answering sentence questions in Spring A, except for question O6. They did significantly better on questions E3 and E4 than students in Spring A ( $p<.05$ for both, using one-tailed $t$ tests), and slightly better than students answering sentence questions.

### 4.6 Other Observations

## Observations from Interviewers

Interviewers noticed several differences in student behavior between the Spring A semester and later semesters. First, students in the Spring A semester made explicit references to English when explaining their answers to the interviewers or when defending their answers after being told they were wrong. While students relied less and less on English as the semester went on, they still tended to defend their answers. They gave essentially the same kinds of explanations, but without explicitly mentioning English. Students in Summer A, Fall A, and Spring B almost never used English when explaining their answers, and they did not try as hard to defend their answers after being told they were wrong. In fact, students in later semesters often noticed that they had made a mistake when they were asked to explain how they solved that problem.

The other major difference interviewers noticed was the speed with which students solved problems. In Spring A, interviews generally took between 30 and 60 minutes. Stu-
dents spent much of this time trying to solve question O6, divide-by-largest. One interviewer described this as "painfully slow to watch." Even those students who solved O6 spent a long time on it. In the the third round of interviews, students were asked a second set of questions after they finished the Scheme questions. These questions took between 15 and 20 minutes to complete, and the whole interview typically took an hour or slightly more. In later semesters, interviews generally took between 15 and 40 minutes, although some took a full hour. The same second set of questions was asked after the Scheme questions in Fall A, and most interviews finished in less than an hour. Interviewers felt that much of this difference was due to the speed with which many students solved O6. In at least five cases, students found the solution before the interviewers had a chance to put the handout with test cases down on the table.

## Observations of TAs

I have noticed that, when explaining Scheme code to students, TAs often ignore sentences in favor of the words they contain. For example, consider a Scheme procedure that returns a sentence of all perfect squares between two numbers. If asked to find the perfect squares between 1 and 10 , it would return the sentence (149). If asked to find the perfect squares between 26 and 35 , it would return (). The technically correct way to describe ( 1 49 ) is "a sentence of one, four, and nine," while the technically correct way to describe () is "an empty sentence." However, TAs generally say that the procedure returns "one, four, and nine" or "nothing." This is a reasonable way to describe what is happening, since the students will probably be more interested in how the program finds the right numbers than in how they end up in a sentence. Unfortunately, this encourages students to ignore sentences without helping them learn when doing so is safe and when it is not.

## Chapter 5

## Discussion

### 5.1 Summary of Results

This research has several significant results. First, students had difficulty understanding one-word sentences and empty words and sentences. Some students do not recognize empty words at all. When asked what should be simple questions, some students struggled with several possible answers, while others confidently gave incorrect answers and were prepared to explain why they felt they were correct. Second, students did not have the same difficulty understanding null or one-element lists. Even though students had been working with sentences for three months and lists for less than a month, scores on a series of sentence questions were significantly lower than scores on otherwise-identical list questions. Third, while treatments did appear to improve students' performance relative to lists at the end of the semester, they did not help students with all of their problems. These findings are summarized in Tables 5.1, 5.2, and 5.3 (the last on page 109).

As shown in Table 5.1, students in the Spring A semester were more likely to write parentheses around sentences and believe that the butfirst of a two-word sentence should be a one-word sentence by the end of the semester. They were also more likely to notice
when something was a word or a one-word sentence when they wrote or read it. They made no significant improvements on any of the three problems with empty words and sentences. In contrast, students in Spring B improved on both parentheses and one-word sentences containing empty words ( $p<.01$ for all except parentheses around sentences in general, for which $p<.1$ ).

Students in Spring B did not generally do as well as students in other semesters on the first round of interviews. Furthermore, they often did not do as well in later rounds as students in Fall A. Some of this may be related to the low number of EECS majors in Spring B or their lower exam scores when compared to other semesters. However, neither the exam scores nor the number of EECS majors predict results overall. Spring B students did make some significant improvements over Spring A students. If the first round of interviews is ignored, they may have done better on putting parentheses around sentences and treating the butfirst of a two-word sentence as a one-word sentence. Furthermore, even including the first round of interviews, they did significantly better on both questions related to one-word sentences that contain empty words.

### 5.2 Sources of Mistakes

## Sentences: Generative or Surface Analogies?

Students in Spring A did use analogies between English and Scheme, at least when explaining their answers to interviewers. However, the students might have been using English only as a surface analogy (used only when explaining their results) rather than as a generative analogy (used when generating their results). The evidence appears to support generative analogies.

However, interviewers observed that students in Summer A, Fall A, and Spring B were more likely to notice that they had made mistakes when they were asked to explain how they got their answers, and they generally did not argue when they were told their answers

| Topic | Problem | Improvement |
| :--- | :--- | :--- |
| Sentences | Students forget to/believe they do not need to write parentheses <br> around sentences of more than one word | Most $(p<.1)$ |
| One-word <br> sentences | Students forget to/believe they do not need to write parentheses <br> around one-word sentences when taking the butfirst of a two- <br> word sentence, or they feel that butfirst should logically produce <br> a word instead of a one-word sentence | Most $(p<.05)$ |
| One-word <br> sentences | Students fail to notice that they have written a word rather than <br> a one-word sentence | Some $(p<.1)$ |
| One-word <br> sentences | Students fail to notice that they have read a one-word sentence <br> rather than a word | Some $(p<.01)$ |
| Empty <br> sentences | Students believe that taking the butfirst of a one-word sentence <br> will generate an error because there would be nothing left | None |
| Empty <br> words <br> and <br> sentences | Students believe that a one-word sentence containing only the <br> empty word is empty | None |
| Empty <br> words <br> and <br> sentences | Students believe that a one-word sentence containing only the <br> empty word is empty but that they can take the first of that <br> sentence | None |

Table 5.1: Students' Problems in the Spring A Semester. The "Improvement" category tells whether students improved on this issue over the semester. A value of "Most" indicates that the percentage of students who made this mistake dropped to near zero by the final round of interviews. A value of "Some" indicates that the percentage of students who made this mistake decreased by the end of the semester, but was still significantly greater than zero. A value of "None" indicates that the percentage of students did not decrease by the end of the semester.

| Problem | Second | Final | Average |
| :--- | :--- | :--- | :--- |
| Students forget to/believe they do not need to write parentheses <br> around sentences of more than one word | $p<.1$ | $p<.1$ | - |
| Students forget to/believe they do not need to write parenthe- <br> ses around one-word sentences when taking the butfirst of a <br> two-word sentence, or they feel that butfirst should logically <br> produce a word instead of a one-word sentence | - | $p<.01$ | - |
| Students believe that taking the butfirst of a one-word sen- <br> tence will generate an error because there would be nothing <br> left | $p<.1$ | - | - |
| Students believe that a one-word sentence containing only the <br> empty word is empty | $p<.01$ | $p<.05$ | $p<.05$ |
| Students had trouble predicting what the first element of a sen- <br> tence containing only the empty word would give | $p<.1$ | $p \approx .05$ | $p<.05$ |

Table 5.2: Improvements from Spring A to Spring B. The "Second" column compares the results of the second rounds of interviews in Spring A and B. The "Final" column compares the results from the sentence versions of the final rounds of Spring A and Spring B. The "Whole" column compares the results from the sentence versions of all rounds of the Spring A and B semesters.
were wrong. When Spring A students were asked to think about their reasoning, they did not see any problems. In fact, they were sure they were correct and sometimes resisted when the interviewers tried to correct them. Students in later semesters, on the other hand, often noticed that they were wrong when they were asked to examine their reasoning. Of those who did not notice their own mistakes, almost none put up a fight when interviewers pointed out the errors. The most logical explanation for this behavior is that the students in Spring A had what they considered to be a very good reason to give and defend the wrong answer, while other students did not.

Finally, the differences between list and sentence scores in the final interviews of each semester argue that Spring A students were operating under a misconception that later students did not share. Spring A was the only time that students performed better with list questions than with sentence questions. While students in Spring A had studied lists in preparation for a midterm, they should not have scored better on list questions than on sentence questions unless there was some reason for them to develop an understanding of car, cdr, and null? in a few weeks that was better than the understanding of first, butfirst, and empty? that they had developed throughout the semester. As Ryan (1985b) points out, familiar words do not force students to learn technical meanings. Unfamiliar words do.

Taken together, these observations and data strongly suggest that students in Spring A had been using English rules for parentheses as a generative analogy. However, this does not explain why students continued to make similar mistakes in later semesters, when they had been exposed to several treatments and were no longer using explicit references to English to justify their answers.

## The Collection Model

Students using Simply Scheme in Berkeley's CS 3 class are not the only ones to forget parentheses. Oliver Grillmeyer's Exploring Computer Science with Scheme (1997) features

## Mistakes to Avoid

Remember that rest returns a list with all but the first element. A common mistake is to think that
(rest ' (a (b)))
returns (b) instead of the true value returned: ((b)).

Figure 5.1: Warning from Exploring Computer Science with Scheme, page ix.
a warning in a box in the preface shown in Figure 5.1. Exploring Computer Science with Scheme uses rest to protect students from cdr in the same way Simply Scheme uses butfirst, so it is possible that Grillmeyer had noticed a similar phenomenon. However, Van Someren (1990) noted similar behavior among students learning Prolog. Lists in Prolog are enclosed by square brackets, "[" and "]," which students often dropped. Additionally, some of them believed that [ ] was nothing. Because these mistakes have been observed in several different languages, only one of which involves sentences, it is not unreasonable to suspect they have a common cause.

The author was not familiar with the collection model until after the Spring B interviews were concluded, so there were neither questions specifically designed to identify students who used it nor treatments specifically designed to help students with it. In addition, because sentences can be thought of as collections of words with just a few additional rules, it is difficult to say that students are relying on some sort of collection analogy.

The one case in which students using the collection model were predicted to give different answers than those using analogies to English was question E4, (first '("")). Students relying on analogies were expected to say that the sentence was empty and that the first would produce either the empty word or an error message. Students relying on the collection model, on the other hand, were expected to say that the sentence was empty and that the first would produce an error message.

Unfortunately, the results from question E4 only offer limited support for the collection model. Every student who gave the wrong answer to the list version of this question said that the answer would be an error, but since so few students answering the list version
actually missed this question, this finding is not reliable. Furthermore, students who gave wrong answers to the sentence version of this question were often very confused about how first worked or what the empty word was.

Nevertheless, most of the mistakes and comments made by students throughout the three later semesters can be explained by the collection model. Students tended to focus on the words in a sentence, often to the exclusion of the sentence itself. When asked about the elements of a sentence, they gave much higher priority to the words than to the presence of parentheses. When explaining their answers, they focused almost entirely on the words in the sentence and dismiss the parentheses. Some students differentiated between "the answer" and "what Scheme would give." "The answer" to (bf '(12)) is just 2, while "What Scheme would give" is (2). Likewise, many students who said that (negate-all '(1)) returned -1 but showed no indication they knew it was a sentence later explained that they had only been looking at "the answer." Students who had trouble finding the error in divide-by-largest often said the same about the test cases for sent-max, which clearly showed sentences rather than words. Furthermore, while students contrasting English and Scheme often pointed out that English sentences could never be empty, they did not mention one-word sentences. This suggests that they have other reasons for their difficulties with one-word sentences.

Additionally, TAs and lab assistants reinforced the collection way of thinking when explaining how some Scheme procedures worked.

More work needs to be done to determine whether the collection model is a significant source of confusion for students trying to understand sentences and lists. Students could be asked to create their own analogies for words and sentences. They could also be asked to compare and contrast either Scheme sentences and English sentences or Scheme sentences and some easily aligned collection. People find it easier to identify differences between two objects if they can align those objects easily; if students can find more differences between

Scheme sentences and a collection than between Scheme sentences and English sentences, they are likely using a collection model.

## Other Explanations

Several other explanations have been proposed. One is that students confuse "returning a value" with "printing the answer." Returning a value is something formal and implies that Scheme might want to use that value, while printing an answer is relatively informal and implies that the results are for human use only. In the former case, the difference between 3 and (3) is significant, while in the latter, the difference is not. When explaining their programs in lab, many students say that Scheme "prints" something when they actually mean that Scheme returns something. This confusion would explain many of the problems students had with question O5, as well as those students who argued that there was a difference between "the answer" and "what Scheme would return." However, it cannot explain students' difficulty detecting sentences in question O6, where the value that Scheme "printed" was immediately used by another procedure and never shown to a human. It also cannot explain the students who genuinely thought that (bf '(12)) produced a number or who thought that ("") was empty. Furthermore, it cannot explain the difference in scores between students answering sentence and list questions in Spring A.

Some of these problems may also be due to the flexibility of the procedures that deal with words and sentences. Almost all of these procedures will work with either words or sentences. The sentence procedure will make a sentence out of any combination of words and sentences, so whether one has the words phonetic and punctuation, the one-word sentences (phonetic) and (punctuation), the word phonetic and the sentence (punctuation), or the sentence (phonetic) and the word punctuation, they can all be combined into the sentence (phonetic punctuation) in the same way. This versatility may de-emphasize the differences between words and sentences, in part because students do not have to pay attention to whether they are working with words or sentences, and in part because if they
write code that should produce sentences but sometimes produces words (such as question O5, negate-all), they can safely add a call to sentence without thinking about why their code does what it does. The procedures that are used to construct lists, on the other hand, are much more picky about what they will work with. Students are forced to learn the difference between combining two one-word lists and two words, and they may pay more attention to data types when they are thinking about sentences. This is a reasonable explanation for many of the students who did not find the difference between words and sentences important, especially in O5 (negate-all) and O6 (divide-by-largest), but also for those in O1 and O3 who were not sure whether 2 or 234 were numbers or sentences. However, this does not explain why many students believed that (bf ' $\left(\begin{array}{ll}1 & 2) \text { ) was actually the }\end{array}\right.$ number 2 or why others had trouble with empty words and sentences.

A possible explanation for students' troubles with empty words is that, while the two parentheses of empty sentences imply a sentence with no words left, empty quotation marks do not imply a word with no letters left. Repeatedly taking the butfirst of a sentence makes it get smaller, while repeatedly taking the butfirst of a word makes it get smaller only until it becomes empty and (from the students' point of view) bigger than a one-letter word. This idea is supported by the number of students who did not recognize "" as the empty word when they dealt with ("") in questions E3 and E4, but it cannot explain the students who recognized the empty word in question E5 but said that it "knows not to say anything." Nevertheless, this explanation is compatible with the use of analogies or the collection model.

During interviews, several students proposed that the reason they had trouble with question O1 was that they thought of numbers and words differently. While they might have thought (bf '(12)) was the number 2, they would have thought (bf ' (how now) ) was the sentence (now). This was never tested during any of the interviews, primarily because students encountered this situation in lab and made the same mistakes.

### 5.3 Success or Failure of Treatments

The results of the various interventions appear to be mixed. The students of Summer A appeared to do better than students of other semesters on sentence questions during the final round of interviews. Their mean scores were significantly higher than those in Spring A $(p<.01)$. The difference was nearly significant for Fall A $(p=.056)$ and Spring B ( $p=.082$ ). It is possible that this difference is due to the small number of students who participated in the last round of interviews, or because the summer students had more time to focus on CS 3. However, one key difference between the Summer and Fall A semesters was that approximately half of the first midterm review session for the summer focused on the differences between Scheme and English and the behavior of procedures such as first, butfirst, last, butlast, and empty?, while little of the fall midterm review dealt with these topics. Summer A was not the first time students had been exposed to a long "Scheme is not English" discussion. I had tried much the same thing in several previous semesters, and I had no success. It is possible that, having done the "Comparing English and Scheme" homework, the students were at least willing to consider alternate lines of reasoning, and that the review session provided one. Something similar may have happened with empty words in Spring B. Students in the second round of interviews in Spring B did significantly better than students in the second rounds of either Spring or Fall A ( $p<.05$ ), and they appeared to do better in the final round than either students in Spring A ( $p<.05$ ) or Fall A ( $p>.1$, probably due to the small number of students in the final round of Spring B). On the first day that students worked with sentences, they encountered WebScheme activities that convinced them their initial suspicions about empty objects were wrong. These activities were followed perhaps an hour later by the $\mathrm{Pez}^{\circledR}$ analogy, which provided students with a model for empty words and sentences, something that neither English nor real-world collections could do.

Students in the final rounds of interviews in Summer A, Fall A, and Spring B did about as well when given sentence questions as when given list questions, and while those in Fall

A and Spring B did not do significantly better than those in Spring A, the results were close to significance.

Students in later semesters did significantly better on questions about a sentence with an empty word. This suggests that the $\mathrm{Pez}^{\circledR}$ may have given students a working analogy for empty words and sentences.

There are several potential reasons for the generally limited success of the treatments. First, it is possible that students did not really refine their use of English analogies. Even after the various interventions, some students spontaneously contrasted English and Scheme sentences, probably because they saw a good connection between the two. Instead of learning to be careful when using analogies, students may have learned to be careful when talking to TAs and other CS 3 staff. On several occasions, students who were asked about elements of English and Scheme said something like, "Oh, not this again!" Students may have been aware of the emphasis on this topic and thus may have taken care that their explanations during the interview did not involve analogies to English. This is unlikely, however, as students proved to have enormous difficulty remembering a series of activities they had done in lab just one week before the interviews. Students answering question E7 were asked whether they remembered certain activities that were directly related to the question. Even among the student who gave the right answer to E7, very few remembered having seen any of the activities.

Another option is that students were more careful with English analogies, but they found other reasons to make the same mistakes. It may be that students need more than to be told to think about the differences between words and sentences. Homework activities like "Comparing English and Scheme" may help students realize that analogies to English are not as useful as they might seem, but they do not help the students find other, more useful models. This is a more likely option, since similar mistakes have been seen in students taking classes very different from Berkeley's current CS 3. The collection model is one possibility, although there could be others.

| Treatment | Success | Possible Reason |
| :--- | :--- | :--- |
| Pez $^{\circledR}$ | Probable | This may have given students an analogy for empty words. <br> Students said it did not help with other sentence issues. It <br> was the only activity to target empty words and sentences in <br> Fall A. |
| Comparing English <br> and Scheme (Sum- <br> mer A) | Unknown | This may have encouraged students to consider the normative <br> explanations given during the midterm. However, it was only <br> used in one semester. |
| Comparing English <br> and Scheme (Fall A <br> and Spring B) | Probable | This may have encouraged students to pay attention to their <br> use of English and Scheme. It was the only activity in Fall A <br> that targeted English and Scheme. |
| Defining Terms | Unknown | This was only used in Spring B. Students in this semester only <br> outperformed students in Fall A on empty words and sen- <br> tences, and this activity does not specifically focus on them. <br> Given that students sometimes failed to think about empty <br> words and sentences at all, it is unlikely that they found this <br> activity very useful. |
| WebScheme | Probable | The WebScheme activities encouraged students who thought <br> that empty words or sentences were literally nothing to talk to <br> their TAs. This may have primed students to respond well to <br> the Pez ${ }^{\circledR}$ analogy. Students in Spring B did better with one- <br> word sentences containing empty words than either Spring or <br> Fall A. |

Table 5.3: Success and Failure of Treatments.

A third option is that the activities were not properly integrated. Perhaps students should be asked to think about definitions before a TA conducts a discussion on the normative meanings for these terms, as suggested by Linn and Eylon (in press).

### 5.4 Limitations and Weaknesses

The findings described here are limited for several reasons. The most important is that the interviews were not audiotaped. Data comes from students' written work and notes the interviewers made. Interviewers were given a list of utterances to listen for and write down, and they wrote additional notes whenever a student said something they considered interesting. As a result, it is impossible to confirm the interviewers' impressions about the speed with which students solved problems or their willingness to believe they had made mistakes. It is also impossible to measure the frequency of certain utterances that only
became significant after they were said by so many students. Finally, it is possible that the interviewers missed some instances of the things they were supposed to mark down.

Individual interviewers may have biased students' responses, possibly by the way they read instructions, asked or answered questions, or corrected students. For example, giving instructions in a certain way for question O6, divide-by-largest, might persuade a student to focus on (or avoid) the test cases. However, two of the interviewers conducted a substantial majority of the interviews, and answers given by the students in each group do not differ significantly. Unless both interviewers biased students in the same direction, it is unlikely that interviewers significantly skewed the results.

The main source of data for this project is the students' scratch work and written answers. While there are many questions that cannot reliably be answered because of weaknesses in coding or transcription, the students' written work, and thus the finding of a difference in scores between students answering list or sentence questions, should be reliable.

## Chapter 6

## Conclusions and Future Work

### 6.1 Conclusions

This research has examined some of the mistakes of students learning Scheme in UC Berkeley's CS 3 class. Like many students, those I studied had trouble understanding collections of objects. Simply Scheme describes two kinds of collections. Scheme sentences are collections of words, and words are collections of one-letter words. Sentences, words, and many of the procedures in Simply Scheme were given familiar names to make them friendlier and more understandable than standard Scheme. However, because students recognize these familiar, friendly names, they assume they know how everything works. English definitions for sentences and words do not adequately explain the behavior of Scheme sentences and words. In particular, they do not explain parentheses or empty words and sentences.

Students often hold fast to misconceptions (Chi, 2005; Fischbein, Deri, Nello, \& Marino, 1985; Groves \& Pugh, 2002; Kim \& Pak, 2002; McCloskey, 1983; Taylor, 1990), possibly because their so-called misconceptions are perfectly valid in other places and at other times (Smith et al., 1993). English and other natural languages are useful when trying to understand programming languages, but only under the right circumstances. Merely telling
students they are wrong when they write sentences without parentheses or misunderstand empty words or sentences is the wrong approach. Most students can think of instances when they successfully used natural language to make sense of Scheme, so they probably will not understand why it failed this time. Instead, these students need to be confronted with evidence that their assumptions are wrong and encouraged to think about the limitations of their understanding. In this way, they will develop a better idea of when they can and cannot safely use natural language to understand Scheme.

Unfortunately, convincing students not to rely too much on natural language does not correct every mistake. Students appear to fall back on the collection model, another idea that works well in the real world but not so well in a formal situation like Scheme. If we want students to develop a solid understanding of parentheses, empty words and sentences, and how to successfully reason about Scheme, we need to encourage them to think more about these topics and, when they are ready, provide them with more details.

### 6.2 Implications for Instruction

Both teachers and students need to be aware of those aspects of a programming language which overlap with students' real-world experiences. While it is tempting to assume that familiarity will help students learn the material, familiarity may also convince them that there is nothing new to learn. Instructors should not completely discourage the use of real-world knowledge, since analogies and natural language are powerful tools for problem solving. Linn and Eylon (in press) suggest four steps for helping students:

1. elicit students' ideas
2. introduce new ideas
3. help students develop criteria for evaluating ideas
4. help students use their criteria

Following these steps, students should be asked to think of real-world meanings for the terms they will learn. What connotations do the words have? What denotations do they have? The term "empty," for example, may mean "meaningless," "lacking in some specified quality or object," or "containing nothing," while it may also be closely associated with "nothing" or "zero." This could be done through a TA-coordinated in-class discussion or as an online discussion during lab. An example of this kind of activity is the "Defining Terms" homework, although it would work better as an in-class assignment.

Once students have listed possible meanings for key terms, they should be given a basic introduction to these terms in the context of the class. A complicated or difficult introduction may encourage students to quickly adopt real-world meanings rather than consider the new, technical meanings (Waltz et al., 2000; Tohill \& Holyoak, 2000). The introduction should be enough to allow students to complete the next two steps, but it should also leave them with enough time to do those steps before class is over.

Once students have both real-world and technical meanings, they need to learn how to tell which to use. Instruction should help them develop useful heuristics for predicting the behavior of code and criteria for differentiating between good and bad outcomes in a program. ${ }^{1}$ Some of these might be

- Given the choice between giving an error message and producing a reasonable answer, a program should (probably) produce a reasonable answer.
- Given the choice between giving an error message and producing an answer that is confusing or wrong, a program should (probably) give an error message.
- It is (usually) better for a program to behave in a consistent way.
- A procedure should do exactly what it is asked to do, not more or less.

[^11]- Data that is worthless in one situation may be valuable in another.

Students could be given small programming problems that each highlight one or two heuristics. Once the students have finished, or at least worked on, all of them, the TA could conduct a discussion that makes these points explicit.

Finally, once students have real-world and technical definitions and criteria to help them decide which meanings are most reasonable, instruction should help them understand the implications of each possible meaning. Students may not realize that there are many reasonable definitions for a familiar term or that different definitions can be used to reach very different conclusions about the behavior of a programming language. Depending on which meaning(s) of empty a student considers, ("") might be empty because it contains no letters (lacking a specified quality or object), because it conveys no useful information (meaningless), or because it contains only "", which is empty (empty is like nothing, and something is empty if it contains nothing). Similarly, it could be non-empty because "" has some strange meaning to Scheme (not meaningless) or because it contains a word (not lacking in some specified quality or object, or not containing nothing). This can be done through a TA-coordinated class discussion in which students are presented with segments of code and asked to list possible answers and decide which are the most likely. I conducted one of these discussions after the Spring B semester, and my students were unusually active.

All of this needs to be done early. Many of the students in the Spring A semester argued when interviewers told them they were wrong and were able to give reasons for some of their mistakes. While the changes to the Fall A curriculum were not particularly effective at helping students reach the right answer, they did appear to leave students willing to consider alternatives. Intervention needs to occur before students have decided that all of their real-world knowledge is applicable to programming.

Instructors should also give careful consideration to analogies they wish to use in class. While an analogy may be designed to explain a few points, students might apply it in other situations and thus reach very non-normative conclusions that, by the instructor's
own explanation, should be correct. An instructor planning to make heavy use of one or more analogies should make sure students understand when the analogies break down.

### 6.3 Future Work

There are three primary directions for continued research. First, I could continue searching for models students use to understand sentences and empty objects. Second, I could examine other issues that I noticed in the interviews but did not actively pursue. Third, I could examine similar problems in other conditions.

## Continuing This Research

Several questions still need better answers. Are students using the collection model? Is this the only other model they are using? Do they still use any of the English model? There are several ways to examine students' preferred analogies during interviews. First, students could be asked to generate, explain, and work with their own analogies. How accurate and complete are student-generated analogies? How comfortable are students with them? How much do their analogies overlap with either English or collections?

Second, students could be asked to compare and contrast English sentences and Scheme sentences, some suitable instance of a collection and Scheme sentences, and possibly a student-generated analogy and Scheme sentences. Students should be able to list more differences between the two that are most alignable (Gentner \& Markham, 1994). While explicitly asking students to compare and contrast will show which pair has the best alignment, it will not show which analogies students spontaneously use. Asking students to describe Scheme sentences and one or more of the alternatives in two separate questions would allow students to compare and contrast when they feel it is reasonable.

Third, students could be given a set of short problems and asked to predict all reasonable answers. When I tried this early in the semester, I found that students could list a
number of reasonable answers but could not decide which were correct without some sort of heuristic (see page 113). Students should be able to pick answers if they are told to use a specific analogy. Does the pattern of answers generated by using English sentences differ significantly from the pattern generated by using a collection? If the two are essentially the same early in the semester, will they be different at the end of the semester?

For that matter, if students abandon the English model, what are the results? Smith et al. (1993) would argue that it is not always wise to get students to throw out a misconception, since many misconceptions are good ideas that have not been properly applied. Collins and Gentner (1987) and M. D. Williams et al. (1983) give instances of students making good use of multiple incorrect mental models by considering the predictions of all of their models, even if each is wrong in some way. One secret to building a superior mental model appears to be thoughtfully combining multiple inferior models rather than trying to build up one perfect model at the expense of all others.

Another question unanswered by this research is how students' native cultures and languages affect their understanding of sentences and empty objects. One-word sentences are completely acceptable in Japanese, for example, because the subject can often be implied by the form of a verb. How are the concepts "empty," "nothing," and "collection" treated in other languages and cultures?

Students interviewed after the first rounds of interviews in Fall A and Spring B appeared much more willing to consider our explanations than students from Spring A. In several cases, when interviewers explained how sentences and words worked, students were pleasantly surprised and asked why this had not been mentioned before. Clearly, we need to take advantage of this. If we wait until students have refined their use of English and then give them a quick review of the details of words and sentences, will they do better?

What would be the results of giving students different models for empty sentences and words? It would be good to develop an analogy that, unlike the bagel or $\mathrm{Pez}^{\circledR}$ analogies, meshes well with words and sentences. Empty words and sentences do not show up in
literature. Rather than writing, "He said '," a good author might write "He said nothing." However, there is a precedent for empty words and sentences in sequential art (comic books and similar media). When drawing a scene in which a character is quite clearly unable to find anything to say, some artists will draw a full-size word balloon and put only "..." in it. Space on the page is precious, so the ellipsis must be quite significant. Writer/artist Lea Hernandez says that she uses them "when the only other thing I could say would involve the word 'Fuck!'" (personal communication, 21 February 2004). Clearly, while an ellipsis means the character is saying nothing, the ellipsis itself says a great deal. Two other possible examples of real-world empty sentences are someone who is completely speechless or who uses body language to convey a whole thought.

Could the editor students use to write Scheme and the interpreter they use to run Scheme be modified to help them see sentences and empty words as things? Both of these programs are capable of color-coding different parts of a Scheme program. If they used one background color for sentences and another for words, so that a sentence would be a visually distinct entity, would students be more likely to think of sentences as objects rather than as collections of objects? Similarly, since empty words or sentences would have the same background color as other words or sentences, would students think of them as regular words or sentences instead of special cases? Would the common colors encourage them to see " " as still a genuine word and () as a genuine sentence?

## Examining Related Topics in CS 3

In English, the statement "If today is Saturday or today is Sunday, today is a weekend. Otherwise, it is a weekday." means the same things as "If today is Saturday or Sunday, it is a weekend. Otherwise, it is a weekday." The question "Is today Saturday or is today Sunday?" is written in Scheme as (or (equal? today 'saturday) (equal? today 'sunday)). This is as close to a word-for-word translation as is likely to be found in programming. Many students, however, assume that a similar word-for-word translation would render the
shorter "Is today Saturday or Sunday?" as (equal? today (or 'saturday 'sunday)). These students are wrong, because while and and or determine whether they have been given true statements, the true statements are (equal? today 'saturday) and (equal? today 'sunday), not just the words saturday and sunday. When students who had made a mistake like this were asked to explain their code, they gave the short English translation. These students had only been taught to use and and or with true and false statements, and they had never seen either used in any situation like this. They came up with this use of and and or on their own, presumably because it sounded logical in English.

Several of the warm-up questions in the first set of interviews in Spring A included questions that dealt with numbers as words. Students did not do very well on these questions. Like all other words, numbers can be taken apart with first and butfirst and put together with word. By and large, students can go through CS 3 without ever caring that numbers are words. However, there are certain projects that require numbers to be words. For example, see the description of the number-spelling program on page 51. Do students have trouble because they do not think of numbers as words in these cases?

A third problem in CS 3 involves the procedures keep and every. These procedures are introduced in the same chapter, but they do very different things. keep acts as a filter, keeping certain words in a sentence and discarding others. (keep weekday? '(sunday monday tuesday wednesday thursday friday saturday)) returns the sentence (sunday saturday), assuming that someone has written weekday?. The procedure every, on the other hand, does something to every word of a sentence. (every square ' (1 2434 ) ) returns the sentence (1 49 16). Students run into trouble when they translate "Keep every weekday" into (keep every weekday? '(sunday monday tuesday wednesday thursday friday saturday)).

## Examining Similar Topics in Other Areas

How do students learning Scheme with other textbooks or different course styles understand empty and one-element lists? Do students who use Simply Scheme in other universities have the same problems? Do the students in Berkeley's next computer science class, CS 61A, have these problems? What differences are there among CS 61A students who have also taken CS 3 and those who started with 61A?

As Fischbein and Baltsan (1998-1999), Grillmeyer (1997), Van Someren (1990), and Zazkis and Gunn (1997) have found, students in many situations experience problems similar to Berkeley's CS 3 students. To see how much influence words and sentences might have on students, someone must study students in other Scheme classes. Do students who use Simply Scheme tend to have more trouble with these topics than students who start with raw, unpronounceable Scheme? What models do other groups of students use to understand parentheses or empty objects?

Scheme is not the only language to have empty objects. Most languages can have empty strings, and many can have empty lists. However, some languages treat empty or one-element lists differently. Empty words, sentences, and lists are considered "true" in Scheme, but empty lists are false in languages such as Lisp, Perl, and Matlab, and empty strings are false in Perl and Matlab. In no case does this make an empty list or string a non-thing, but it may encourage students to think otherwise. Java has both empty (length zero) lists and null lists, and the two are quite different.

How do students learning other languages think about one-element lists or strings? Does someone learning C grasp the difference between the string "a" and the character 'a'? How do students learning Matlab handle the fact that a number, a vector (a list of numbers), and a matrix (a list of vectors) are all the same? This would seem to avoid confusion over one-element vectors, but does it introduce different confusion, above and beyond that of linear algebra itself?

What other related problems do students have with other languages? I have seen some of the problems students have with "and" and "or" in Java. In Java, "and" is \&\& and "or" is II, so code does not look much like English. However, these students thought about it as though it did. They made a number of mistakes with "and" that worked perfectly well in English. For example, one wrote something like if ( $\mathrm{x}>3 \& \&<7$ ), which he said meant "If x is greater than three and less than seven." Another asked if he could write something like if ( $\mathrm{x} \| \mathrm{y}<0$ ) to express "If x or y is less than zero." After a little bit of thinking, he decided that it probably would not work, even though he felt it should.

Problems with empty words and sentences may be closely related to problems with zero that many students have. Students have difficulty understanding zero as a concept (Bialystok \& Codd, 2000; Pepperberg \& Gordon, 2005). Furthermore, students who can (presumably) grasp the idea of zero may have trouble with it in other contexts. For example, beginning algebra students who can solve $y+3 x=5 x$ by moving the $3 x$ to the other side and subtracting (to get $y=5 x-3 x$ ) may have trouble with $y+5 x=5 x$, since the intermediate step is $y=5 x-5 x$, and $5 x-5 x$ is "nothing."

Schoenfeld (1998) describes the problems one class had understanding division with exponents. The students were fine with something like $x^{5} / x^{3}$, which comes out to $x^{2}$, but they had trouble with problems like $x^{5} / x^{5}$, which is $x^{0}$ or 1 . Students thought the answer should be zero, since there was nothing left.

Changing the scale somewhat, how do expert programmers use analogies? Do they use them frequently or sparingly? What kinds of analogies do they favor? Do they regularly switch to natural language when trying to make sense of difficult problems, as do Taylor's (1990) students? Under what circumstances are they led astray by their analogies? A better understanding of experts' analogies could help us understand students' use of analogies.

## References

Abelson, H., Sussman, G. J., \& Sussman, J. (1996). Structure and interpretation of computer programs (2nd ed.). Cambridge, MA: The MIT Press.

Bialystok, E., \& Codd, J. (2000). Representing quantity beyond whole numbers: Some, none, and part. Canadian Journal of Experimental Psychology, 54(2), 117-128.

Bonar, J., \& Soloway, E. (1989). Preprogramming knowledge: A major source of misconceptions in novice programmers. In E. Soloway \& J. Spohrer (Eds.), Studying the novice programmer. Hilsdale, NJ: Lawrence Erlbaum Associates.

Borge, V. (2001). Phonetically speaking - and don't forget the piano [CD]. Jasmine Music.
The Burlingame Pez Museum. (n.d.). (http://www.burlingamepezmuseum.com)
Chi, M. T. H. (2005). Commonsense conceptions of emergent processes: Why some misconceptions are robust. Journal of the Learning Sciences, 14(2), 161-199.

Clancy, M., Titterton, N., Ryan, C., Slotta, J., \& Linn, M. (2003). New roles for students, instructors, and computers in a lab-based introductory programming course. In Proceedings of the 34th technical symposium on computer science education (pp. 132-136). Reno, NV, United States: ACM Press.

Collins, A., \& Gentner, D. (1987). How people construct mental models. In D. E. Holland \& N. Quinn (Eds.), Cultural models in language and thought (pp. 243-265). Cambridge, UK: Cambridge University Press.

Davis, E. A., Linn, M. C., \& Clancy, M. J. (1995a). Learning to use parentheses and quotes in LISP. Computer Science Education, 6, 15-31.

Davis, E. A., Linn, M. C., \& Clancy, M. J. (1995b). Students' off-line and on-line expereinces. Journal of Educational Computing Research, 12(2), 109-134.

Davis, E. A., Linn, M. C., Mann, L. M., \& Clancy, M. J. (1993). Minding your ps and qs: Using parentheses and quotes in LISP. In C. R. Cook, J. C. Scholtz, \& J. C. Spohrer (Eds.), Empirical studies of programmers: Fifth workshop (pp. 62-85). Norwood, NJ: Ablex Publishing Corporation.

Felleisen, M., Findler, R. B., Flatt, M., \& Krishnamurthi, S. (2001). How to design programs: an introduction to programming and computing. Cambridge, MA: MIT Press.

Fischbein, E., \& Baltsan, M. (1998-1999). The mathematical concept of set and the 'collection' model. Educational Studies in Mathematics, 37(1), 1-22.

Fischbein, E., Deri, M., Nello, M. S., \& Marino, M. S. (1985). The role of implicit models in solving verbal problems in multiplication and division. Journal for Research in Mathematics Education, 16(1), 3-17.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. Cognitive Science, 7, 155-170.

Gentner, D. (1998). Analogy. In W. Bechtel \& G. Graham (Eds.), A companion to cognitive science (pp. 107-113). Oxford: Basil Blackwell.

Gentner, D. (1999). Analogy. In R. A. Wilson \& F. C. Keil (Eds.), The MIT encyclopedia of the cognitive sciences (pp. 17-20). Cambridge, MA: MIT Press.

Gentner, D., \& Gentner, D. R. (1983). Flowing waters or teeming crowds: Mental models of electricity. In D. Gentner \& A. L. Stevens (Eds.), Mental models (pp. 99-129). Hillsdale, NJ: Lawrence Erlbaum Associates.

Gentner, D., \& Markham, A. B. (1994). Structural alignment in comparison: No difference without similarity. Psychological Science, 5(3), 152-158.

Gentner, D., \& Markham, A. B. (1997). Structure mapping in analogy and similarity. American Psychologist, 52, 45-56.

Gentner, D., \& Stevens, A. L. (Eds.). (1983). Mental models. Hillsdale, NJ: Lawrence Erlbaum Associates.

Gentner, D., \& Toupin, C. (1986). Systematicity and surface similarity in the development of analogy. Cognitive Science, 10, 277-300.

Grillmeyer, O. (1997). Exploring computer science with scheme. New York, NY: SpringerVerlag.

Groves, F. H., \& Pugh, A. F. (2002). Cognitive illusions as hindrances to learning complex environmental issues. Journal of Science Education and Technology, 11(4), 381-390.

Harvey, B., \& Wright, M. (1994). Simply scheme: introducing computer science. Cambridge, MA: The MIT Press.

Hoadley, C. M., Linn, M. C., Mann, L. M., \& Clancy, M. J. (1996). When, why and how do novice programmers reuse code? In D. A. Boehm-Davis \& W. D. Gray (Eds.), Empirical studies of programmers, sixth workshop (pp. 109-130). Norwood, NJ: Ablex Publishing Corporation.

Holyoak, K. J., \& Hummel, J. E. (2001). Toward an understanding of analogy within a biological symbol system. In D. Gentner, K. J. Holyoak, \& B. K. Kokinov (Eds.), The analogical mind (pp. 161-195). Cambridge, MA: The MIT Press.

Holyoak, K. J., \& Thagard, P. (1997). The analogical mind. American Psychologist, 52(1), 35-44.

Höök, K., Taylor, J., \& Du Boulay, B. (1990). Redo "TRY ONCE AND PASS": The influence of complexity and graphical notation on novices' understanding of Prolog. Instructional Science, 19(4/5), 337-360.

Hummel, J. E., \& Holyoak, K. J. (1997). Distributed representations of structure: A theory of analogical access and mapping. Psychological Review, 104(3), 427-466.

Jarvis, S., \& Odlin, T. (2000). Morphological type, spatial reference, and language transfer. Studies in Second Language Acquisition, 22, 535-556.

Kim, E., \& Pak, S.-J. (2002). Students do not overcome conceptual difficulties after solving 1000 traditional problems. American Journal of Physics, 70(7), 759-765.

Kolikant, Y. B.-D. (2005). Students' alternative standards for correctness. In Icer '05: Proceedings of the 2005 international workshop on computing education research (pp. 37-43). New York, NY, USA: ACM Press.

Lee, O., \& Lehrer, R. (1988). Conjectures concerning the origins of misconceptions in Logo. Journal of Educational Computing Research, 4(1), 87-105.

Linchevski, L., \& Vinner, S. (1988). The naive concept of sets in elementary teachers. In Proceedings of the $12^{\text {th }}$ international conference, psychology of mathematics education (Vol. 11, pp. 471-478). Vezprem, Hungary.

Linn, M. C., \& Eylon, B.-S. (in press). Science education: Integrating views of learning and instruction. In P. A. Alexander \& P. H. Winne (Eds.), Handbook of educational psychology, 2nd edition. Lawrence Erlbaum Associates.

McCloskey, M. (1983). Naive theories of motion. In D. Gentner \& A. L. Stevens (Eds.), Mental models (pp. 299-324). Hillsdale, NJ: Lawrence Erlbaum Associates.

Medin, D. L., Goldstone, R. L., \& Gentner, D. (1993). Respects for similarity. Psychological Review, 100(2), 254-278.

Pea, R. (1986). Language-independent conceptual 'bugs' in novice programming. Journal of Educational Computing Research, 2(1), 25-36.

Pepperberg, I. M., \& Gordon, J. D. (2005). Number comprehension by a grey parrot (Psittacus erithacus), including a zero-like concept. Journal of Comparative Psychology, 119(2), 197-209.

Piaget, J. (1926/1972). The child's conception of the world. Towota, NJ: Littlefield Adams. Ryan, J. N. (1985a). The language gap: Common words with technical meanings. Journal of Chemical Education, 62(12), 1098-1099.

Ryan, J. N. (1985b). The secret language of science or, radicals in the classroom. The American Biology Teacher, 47(2), 91.

Scherz, Z., Goldberg, D., \& Fund, Z. (1990). Cognitive implications of learning PrologMistakes and misconceptions. Journal of Educational Computing Research, 6(1), 89-110.

Schmidt, H.-J. (1997). Students' misconceptions-Looking for a pattern. Science Education, $81(2), 123-135$.

Schoenfeld, A. H. (1998). Towards a theory of teaching-in-context. Issues in Education, 4(1), 1-94.

Scholtz, J., \& Wiedenbeck, S. (1993). An analysis of novice programmers learning a second language. In C. R. Cook, J. C. Scholtz, \& J. C. Spohrer (Eds.), Empirical studies of programmers: Fifth workshop (pp. 187-205). Norwood, NJ: Ablex Publishing Corporation.

Smith, J. P., III, diSessa, A. A., \& Roschelle, J. (1993). Misconceptions reconceived: A constructivist analysis of knowledge in transition. Journal of the Learning Sciences, 3(2), 115-163.

Spellman, B. A., \& Holyoak, K. J. (1996). Pragmatics in analogical mapping. Cognitive Psychology, 31, 307-346.

Spohrer, J. C., \& Soloway, E. (1986). Analyzing the high frequency bugs in novice programs. In E. Soloway \& S. Iyengar (Eds.), Empirical studies of programmers (pp. 230-251). Norwood, NJ: Ablex Publishing Corporation.

Stacey, K., \& MacGregor, M. (1993). Origins of students' errors in writing equations. In A. B. . T. Cooper (Ed.), New directions in algebra education. Brisbane: Queensland University of Technology.

Stevens, A., Collins, A., \& Goldin, S. E. (1979). Misconceptions in students' understanding. Journal of Man-Machine Studies, 11, 145-156.

Taylor, J. (1990). Analysing novices analysing Prolog: What stories do novices tell themselves about Prolog? Instructional Science, 19(4/5), 283-309.

Tohill, J. M., \& Holyoak, K. J. (2000). The impact of anxiety on analogical reasoning. Thinking and Reasoning, 6(1), 27-40.

Van Someren, M. W. (1990). Understanding students' errors with Prolog unification. Instructional Science, $19(4 / 5), 361-376$.

Waltz, J. A., Lau, A., Grewal, S. K., \& Holyoak, K. J. (2000). The role of working memory in analogical mapping. Memory and Cognition, 28(7), 1205-1212.

Wiedenbeck, S., \& Scholtz, J. (1996). Adaptation of programming plans in transfer between programming languages: A developmental study. In D. A. Boehm-Davis \& W. D. Gray (Eds.), Empirical studies of programmers, sixth workshop (pp. 233-253). Norwood, NJ: Ablex Publishing Corporation.

Williams, H. T. (1999). Semantics in teaching introductory physics. American Journal of Physics Teachers, 67(8), 670-680.

Williams, M. D., Hollan, J. D., \& Stevens, A. L. (1983). Human reasoning about a simple physical system. In D. Gentner \& A. L. Stevens (Eds.), Mental models (pp. 131-154). Hillsdale, NJ: Lawrence Erlbaum Associates.

Zazkis, R., \& Gunn, C. (1997). Sets, subsets, and the empty set: Students' constructions and mathematical conventions. Journal of Computers in Mathematics and Science Teaching, 16(1), 133-169.

## Appendix A

## Interview Questions

## A. 1 Spring A

## First Interview

1. What is (number? (word 12345 ))?
2. What does this procedure do?
```
(define (mystery x)
    (if (word? x)
        (+ x 0)
        x))
```

3. Describe the domain and range of this procedure:
```
(define (mystery2 x)
    (cond
        ((word? x) x)
        ((number? x) (/ x 0))
        ((boolean? x) #f)
        ((sentence? x) (first x))
        (else x)))
```

4. Let's say we already have vowel? written. Now I want to define consonant?. Will this work?
(define (consonant? L)
(not (vowel? L)))
5. If you type (first ('a b)), Scheme gives you just a. If you type (word (first
' ( a b) ) 'nd), Scheme gives you and. However, if we replace (first ' (a b)) with a, the answer we know Scheme will give, Scheme complains. This doesn't seem right. What do you think is going on here?
6. What is (bf $\quad\left(\begin{array}{ll}1 & 2)\end{array}\right)$ ?
7. What is (bl '(1 2))?
8. What is (bf ' (1 $2 \mathrm{l}_{3} 4$ 4) )?
9. What is (bl ' (1 $\left.2 \mathrm{l}_{3} 34\right)$ )?
10. What is (bf '(1))?
11. What is (bl '(1))?
12. What is (empty? '(""))
13. What is (first '(""))
14. Here is something that you might have been asked to do in homework or on a test:

Write a procedure called sum-of-square-roots, which takes two numbers, $x$ and $y$, finds the square root of each, and returns the sum of those square roots. $x$ and $y$ will not be negative numbers.

One student writes this as an answer on the test:
(define
(sum-of-square-roots $x$ y)
(if (or (< x 0) ( $<\mathrm{y} 0$ )) ; ; line 1
'() ; ;line 2
(+ (sqrt x) (sqrt y)))) ; ;line 3

Why did this person include lines 1 and 2 ?

## Second Interview

## Main Sheet

- What is (bf ' $\left(\begin{array}{ll}1 & 2) \text { )? }\end{array}\right.$
- What is (bf ' (1 $2 \boldsymbol{2} 34)$ )?
- What is (bf '(1))?
- What is (empty? '(""))?
- What is (first '(""))?
- The procedure negate-all takes a sentence of numbers and swaps their signs. In other words, it returns a sentence with all of those numbers multiplied by -1 . It doesn't have to work with an empty sentence. Here's a version somebody wrote. Does it work? How do you know? Test it for at least the following sentences: (1), (-1 $2-3$ ), and (10 $203040-50$ ).

```
(define (negate-all sent)
```

    (if (empty? (bf sent))
            (* -1 (first sent))
            (se (* -1 (first sent))
                (negate-all (bf sent)))))
    - This summer we had students write a program called number-spell, which takes a number and returns a sentence with that number written out in words. For example, (number-spell 11000 should give (eleven thousand). Don't worry, you won't have to write this!

Students made a lot of mistakes on this. Here are four examples:

- One student's program spelled 11000 as (eleven thousand zero zero zero)
- One student's program spelled 11000 as (eleven thousand zero)
- One student's program spelled 11000 as (eleven thousand "")
- One student's program spelled 11000 as (ten one thousand)

Which of these sound like the most serious error to you? Which sound like the least serious? Why?

- You and a friend are working on a homework problem together. Let's say this homework is a group project, so you aren't cheating. Here is the problem: Write a procedure called divide-by-largest, which takes a sentence of numbers and divides every number in it by the largest number. For example, (divide-by-largest ' (1 243432 1)) should give you ( 0.250 .50 .7510 .750 .50 .25 ) .

Your friend writes some of the code, but you write the rest. Here is your code:

```
(define (divide-by-largest sent)
    (divide-sentence-by sent (sent-max sent)))
;;divide-sentence-by should take a number and divide ;;all of the numbers
in that sentence by that num (define (divide-sentence-by sent num)
    (if (empty? sent)
        ,()
        (se (/ (first sent) num)
                    (divide-sentence-by (bf sent) num))))
```

You try (divide-by-largest '( $\left.\begin{array}{llllll}1 & 2 & 3 & 4 & 3 & 1\end{array}\right)$ ) and get an error!

## Second Sheet

[This sheet was handed out when students got to Problem 7. It was folded so that students could see either the code or the test cases, but not both.]

Here is your friend's code. Can you find the error?

```
    ;;sent-max should take a sentence and return the smallest
number in it (define (sent-max sent)
    (cond ((= (count sent) 1)
            sent)
        ((> (first sent) (first (bf sent)))
        (sent-max (se (first sent) (bf (bf sent)))))
        (else (sent-max (bf sent)))))
```

Your friend sends you some test cases that he or she says will prove that sent-max works. Can you find the error?

```
    >
(sent-max '(55)) (55) > (sent-max '(1 2 3 4 3 2 1)) (4) > (sent-max '(-3
-2 -1)) (-1)
```


## Third Interview

## Sentence Questions, Main Sheet

1. The procedure negate-all takes a sentence of numbers and swaps their signs. In other words, it returns a sentence with all of those numbers multiplied by -1 . It doesn't have to work with an empty sentence. Here's a version somebody wrote. Does it work? How do you know? Test it for at least the following sentences: (1), (-1 $24-3$ ), and (10 $203040-50$ ).
```
    (define (negate-all sent)
```

        (if (empty? (bf sent))
            (* - 1 (first sent))
            (se (*-1 (first sent))
                    (negate-all (bf sent)))))
    2. What is (bf $\quad\left(\begin{array}{ll}1 & 2)\end{array}\right)$ ?
3. What is (bf '(1) $\left.\begin{array}{llll}1 & 3 & 4\end{array}\right)$ ?
4. What is (bf '(1))?
5. What is (empty? '(""))?
6. What is (first '(""))?
7. You and a friend are working on a homework problem together. Let's say this homework is a group project, so you aren't cheating. Here is the problem: Write a procedure
called divide-by-largest, which takes a sentence of numbers and divides every number in it by the largest number. For example, (divide-by-largest ' (1 243432 1)) should give you ( $0.250 .50 .7510 .75 \quad 0.5 \quad 0.25)$.

Your friend writes some of the code, but you write the rest. Here is your code:

```
(define (divide-by-largest sent)
    (divide-sentence-by sent (sent-max sent)))
;;divide-sentence-by should take a number and divide ;;all of the numbers
in that sentence by that num (define (divide-sentence-by sent num)
    (if (empty? sent)
        ,()
        (se (/ (first sent) num)
            (divide-sentence-by (bf sent) num))))
```

You try (divide-by-largest '( $\left.\begin{array}{lllllll}1 & 2 & 3 & 4 & 3 & 2 & 1\end{array}\right)$ ) and get an error!

## Sentence Questions, Second Sheet

This sheet was the same as that handed out in the second interview.

## List Questions, Main Sheet

1. The procedure negate-all takes a list of numbers and swaps their signs. In other words, it returns a list with all of those numbers multiplied by -1 . It doesn't have to work with an empty list. Here's a version somebody wrote. Does it work? How do you know? Test it for at least the following lists: (1), (-1 2 -3), and (10 2030 $40-50$ ).
```
(define (negate-all lst)
    (if (null? (cdr lst))
            (* -1 (car lst))
            (cons (* -1 (car lst))
                (negate-all (cdr lst)))))
```

2. What is (cdr ' (1 2))?
3. What is (cdr '(1 2 2 3 4) )?
4. What is (cdr '(1))?
5. What is (null? '(()))?
6. What is (car '(()))?
7. You and a friend are working on a homework problem together. Let's say this homework is a group project, so you aren't cheating. Here is the problem: Write a procedure called divide-by-largest, which takes a list of numbers and divides every number in it by the largest number. For example, (divide-by-largest ' ( $\begin{array}{llllll}1 & 2 & 3 & 4 & 3 & 2\end{array}$ 1) ) should give you (0.25 $0.50 .7510 .750 .5 \quad 0.25$ ).

Your friend writes some of the code, but you write the rest. Here is your code:

```
(define (divide-by-largest lst)
    (divide-sentence-by lst (list-max lst)))
;;divide-list-by should take a number and divide ;;all of the numbers in
that list by that num (define (divide-sentence-by lst num)
    (if (null? lst)
        ,()
        (cons (/ (car lst) num)
                    (divide-list-by (cdr lst) num))))
```

You try (divide-by-largest '(123 $\left.23 \begin{array}{lll}1 & 2 & 1\end{array}\right)$ ) and get an error!

## List Questions, Second Sheet

Here is your friend's code. Can you find the error?

```
;;list-max should take a list and return the smallest
number in it (define (list-max lst)
    (cond ((= (length lst) 1)
            lst)
        ((> (car lst) (car (cdr lst)))
            (list-max (cons (car lst) (cdr (cdr lst)))))
        (else (list-max (cdr lst)))))
```

Your friend sends you some test cases that he or she says will prove that list-max works. Can you find the error?

```
>
(list-max '(55)) (55) > (list-max '(1 2 3 4 3 2 1)) (4) > (list-max '(-3
-2 -1)) (-1)
```


## A. 2 Summer A

## First Interview

1. What is (bf $\quad\left(\begin{array}{ll}1 & 2)\end{array}\right)$ ?
2. What is (bl ' $(12)$ )?
3. What is (bf '(1 2434$)$ )?
4. What is (bl '(1 2234 ))?
5. What is (bf '(1))?
6. What is (bl '(1))?
7. What is (empty? '(""))
8. What is (first '(""))

## Second Interview

This was the same as the second interview in Spring A.

## Third Interview

The sheets sheets were identical to those in the third interview in Spring A.

## A. 3 Fall A

## First Interview

1. What is (bf $\quad\left(\begin{array}{ll}1 & 2)\end{array}\right)$ ?
2. What is (bl '(1 2))?
3. What is (bf ' $\left.\begin{array}{llll}1 & 2 & 3 & 4\end{array}\right)$ ?
4. What is (bl '(1 $\left.2 \begin{array}{lll}1 & 3\end{array}\right)$ )?
5. What is (bf '(1))?
6. What is (bl '(1))?
7. What is (empty? '(""))
8. What is (first '(""))
9. What do you think are the essential parts of an English sentence?
10. What do you think are the essential parts of a Scheme sentence?

## Second Interview

## Main Sheet

1. What is (bf $\quad\left(\begin{array}{ll}1 & 2)\end{array}\right)$ ?
2. What is (bf '(1 2434 ) ?
3. What is (bf '(1))?
4. What is (empty? '(""))?
5. What is (first $\quad(" \mathrm{"})$ )?
6. The procedure negate-all takes a sentence of numbers and swaps their signs. In other words, it returns a sentence with all of those numbers multiplied by -1 . It doesn't have to work with an empty sentence. Here's a version somebody wrote. Does it work? How do you know? Test it for at least the following sentences: (1), (-1 $24-3$ ), and (10 $203040-50$ ).
```
    (define (negate-all sent)
```

        (if (empty? (bf sent))
            (* -1 (first sent))
            (se (* -1 (first sent))
                    (negate-all (bf sent)))))
    7. You and a friend are working on a homework problem together. Let's say this homework is a group project, so you aren't cheating. Here is the problem: Write a procedure called divide-by-largest, which takes a sentence of numbers and divides every number in it by the largest number. For example, (divide-by-largest ' (1 243432 1)) should give you ( 0.250 .50 .7510 .750 .50 .25 ).

Your friend writes some of the code, but you write the rest. Here is your code:

```
(define (divide-by-largest sent)
    (divide-sentence-by sent (sent-max sent)))
;;divide-sentence-by should take a number and divide ;;all of the numbers
in that sentence by that num (define (divide-sentence-by sent num)
    (if (empty? sent)
        ,()
        (se (/ (first sent) num)
                    (divide-sentence-by (bf sent) num))))
```

You try (divide-by-largest '( $\left.\begin{array}{lllllll}1 & 2 & 3 & 4 & 3 & 2 & 1\end{array}\right)$ ) and get an error!
8. What do you think are the essential parts of an English sentence?
9. What do you think are the essential parts of a Scheme sentence?

## Second Sheet

Here is your friend's code. Can you find the error?

```
    ;;sent-max should take a sentence and return the largest
number in it (define (sent-max sent)
    (cond ((= (count sent) 1)
            sent)
        ((> (first sent) (first (bf sent)))
            (sent-max (se (first sent) (bf (bf sent)))))
        (else (sent-max (bf sent)))))
```

Your friend sends you some test cases that he or she says will prove that sent-max works. Can you find the error?
>
(sent-max $\left.{ }^{\prime}(55)\right)(55)>\left(s e n t-\max ^{\prime}(1234321)\right)(4)>\left(\right.$ sent-max ${ }^{\prime}(-3$ -2 -1)) (-1)

## Third Interview

## Main Sheet

1. What is (bf $\quad\left(\begin{array}{ll}1 & 2)\end{array}\right)$ ?
2. What is (bf '(1 2434 ) ?
3. What is (bf '(1))?
4. What is (empty? '(""))?
5. What is (first '(""))?
6. The procedure negate-all takes a sentence of numbers and swaps their signs. In other words, it returns a sentence with all of those numbers multiplied by -1 . It doesn't have to work with an empty sentence. Here's a version somebody wrote. Does it work? How do you know? Test it for at least the following sentences: (1), (-1 $24-3$ ), and (10 $203040-50$ ).
```
(define (negate-all sent)
    (if (empty? (bf sent))
        (* -1 (first sent))
        (se (* -1 (first sent))
                (negate-all (bf sent)))))
```

7. What is (every bf '(1 22333 4444))?
8. Explain what happens when Scheme evaluates (every square '())
9. You and a friend are working on a homework problem together. Let's say this homework is a group project, so you aren't cheating. Here is the problem: Write a procedure called divide-by-largest, which takes a sentence of numbers and divides every number in it by the largest number. For example, (divide-by-largest ' $\begin{array}{llllll}1 & 2 & 3 & 4 & 3 & 2\end{array}$ 1)) should give you ( 0.250 .50 .7510 .750 .50 .25 ).

Your friend writes some of the code, but you write the rest. Here is your code:

```
(define (divide-by-largest sent)
    (divide-sentence-by sent (sent-max sent)))
;;divide-sentence-by should take a number and divide ;;all of the numbers
in that sentence by that num (define (divide-sentence-by sent num)
    (if (empty? sent)
        ()
        (se (/ (first sent) num)
                    (divide-sentence-by (bf sent) num))))
```

You try (divide-by-largest '(1 2434321$)$ ) and get an error!
10. What do you think are the essential parts of an English sentence?
11. What do you think are the essential parts of a Scheme sentence?

## Second Sheet

This was the same as that given out in the second Fall A interview.

## Fourth Interview

## Sentence Questions, Main Sheet

This was the same as that given out in the third Fall A interview.

## Sentence Questions, Second Sheet

This was the same as that given out in the second Fall A interview.

## List Questions, Main Sheet

## Main Sheet

1. What is $(c d r \quad$ ' $(12))$ ?
2. What is (cdr ' $\left.\begin{array}{llll}1 & 2 & 3 & 4\end{array}\right)$ ?
3. What is (cdr '(1))?
4. What is (null? '(()))?
5. What is (car '(()))?
6. The procedure negate-all takes a list of numbers and swaps their signs. In other words, it returns a list with all of those numbers multiplied by -1 . It doesn't have to work with a null list. Here's a version somebody wrote. Does it work? How do you know? Test it for at least the following lists: (1), ( $\left.\begin{array}{lll}-1 & 2 & -3\end{array}\right)$, and (10 203040 -50).
(define (negate-all lst)
(if (null? (cdr lst))
(* -1 (car lst))
(se (* -1 (car lst))
(negate-all (cdr lst)))))
7. What is (map bf '(1 22333 4444))?
8. Explain what happens when Scheme evaluates (map square '())
9. You and a friend are working on a homework problem together. Let's say this homework is a group project, so you aren't cheating. Here is the problem: Write a procedure called divide-by-largest, which takes a list of numbers and divides every number in it by the largest number. For example, (divide-by-largest '(1 $\begin{aligned} & 1 \\ & 3\end{aligned} 4$ should give you (0.25 0.50 .7510 .750 .50 .25 ).

Your friend writes some of the code, but you write the rest. Here is your code:

```
(define (divide-by-largest lst)
    (divide-sentence-by lst (list-max lst)))
;;divide-sentence-by should take a number and divide ;;all of the numbers
in that list by that num (define (divide-sentence-by lst num)
    (if (null? lst)
            ,()
            (cons (/ (car lst) num)
                    (divide-list-by (cdr lst) num))))
```

You try (divide-by-largest '( $\left.\begin{array}{lllllll}1 & 2 & 3 & 4 & 3 & 2 & 1\end{array}\right)$ ) and get an error!
10. What do you think are the essential parts of an English sentence?
11. What do you think are the essential parts of a Scheme list?

## List Questions, Second Sheet

Here is your friend's code. Can you find the error?

```
;;list-max should take a list and return the largest
number in it (define (list-max lst)
    (cond ((= (length lst) 1)
            lst)
        ((> (car lst) (car (cdr lst)))
            (list-max (cons (car lst) (cdr (cdr lst)))))
        (else (list-max (cdr lst)))))
```

Your friend sends you some test cases that he or she says will prove that list-max works. Can you find the error?

```
    >
(list-max '(55)) (55) > (list-max '(1 2 3 4 3 2 1)) (4) > (list-max '(-3
-2 -1)) (-1)
```


## A. 4 Spring B

All handouts were the same as those in Fall A.

## Appendix B

## Changes in the Curriculum

These are all of the activities inspired by this research that were added to the CS 3 curriculum. Some of the reasons behind them and observations on their effectiveness are included here for convenience, but most of the details can be found in earlier chapters.

## B. 1 Additions to Summer A

## An Analogy for Words

This page (shown in Figure B.1) was added to the start of the second day's lab materials. Students saw a very basic introduction to words and sentences on the first day, and were assumed to have done the readings that introduce words and sentences. The idea was to offer students an analogy for words that explains first, butfirst, and empty words.

This lab reading was assigned to Summer A, Fall A, and Spring B students. In Summer A, we accompanied it with a brief $\mathrm{Pez}^{\circledR}$ demonstration and free candy. Few students gave any indication that they remembered this analogy during interviews; one who did said that it helped him with butfirst but not with butlast, since a $\mathrm{Pez}^{\circledR}$ dispenser can only dispense candy from one end.

A sentence is a collection of words. A word is a collection of letters. Amazingly enough, your TA or instructor will now explain how words and sentences are like Pez candy dispensers. Here's a basic summary of the argument:

- The sentence or word itself is the dispenser
- Individual words in the sentence or letters in the word are like the candies.
- Individual candies are in a specific order within the dispenser, just like individual words or letters are in a specific order within the sentence or word.
- With a flip of your finger, you can separate the first candy (first) from the dispenser and all of the rest of the candies (butfirst). You can use those two procedures to separate the first word or letter from the rest of the sentence or word.
- As long as it is your Pez dispenser, it's OK to take the last candy out. It's also OK to take the first or butfirst of a one-word sentence or a one-letter word.
- People collect empty Pez dispensers. I promise I'm not making this up. If you don't believe me, check out the Burlingame Pez Museum. It's equally OK in Scheme to have an empty sentence (it looks like ()) or an empty word (it looks like " ").
- A Pez dispenser is only empty when it doesn't have any candy at all in it. You can't just say it's empty if the last thing in there is a candy you don't like. Likewise, you can't say a sentence is empty just because you don't like what it contains. If " " is still a word, (" ") is not an empty sentence.

Don't go too crazy with the analogy, though. You can pop the top on a Pez dispenser even after it's empty, although you won't get any candy. If you try to take apart an empty word or sentence, Scheme throws a fit.

Figure B.1: An analogy for words.

## Comparing English and Scheme

This activity (Figure B.2) was given as a review just before the first midterm. A similar activity was designed to be given out as homework once students had leaned how to combine conditionals (like if) and predicates (like empty?) with words and sentences. This would have put it two labs after students had their first serious introduction to words and sentences. Unfortunately, that activity was lost in the computer system we used in lab, so this one was added at the end of the section on basic Scheme.

Both Scheme and English deal with numbers, words, and sentences. Sometimes Scheme and English agree, but sometimes they do not. These differences can cause all kinds of trouble on exams.

List as many ways that Scheme and English words, sentences, and numbers are alike as you can think of. Then list as many ways that Scheme and English words, sentences, and numbers are different as you can think of. Then comment on what other people have said. Here are some things to consider:

1. If somebody says "No" in English, it doesn't really matter if it is the word "no" or the sentence "No." You get the idea. How about Scheme? Is (no) the same as no?
2. How are parentheses used in English? How about Scheme?
3. English doesn't even have empty sentences or words. What do you think () or " " are in Scheme?
4. Numbers and words are pretty different in English. For example, it's OK if you name a baby after a famous singer. It's not OK if you name a baby after a famous number ("This is 2.718281828 . Isn't she cute?"). What about in Scheme

Figure B.2: Comparing English and Scheme, Summer A.

This was intended to make students think about the analogies they might use. Ideally, students would recognize that there were limits to how far they could carry analogies and would thus be less likely to run wild with them, as they had in previous semesters. Despite the fact that this activity was given out rather late, it may have helped make students aware of the limitations of analogies to English.

The activity was given only to Summer A students.

## B. 2 Additions to Fall A

## An Analogy for Words

The $\mathrm{Pez}{ }^{\circledR}$ activity was again used this semester, although the TAs did not talk very much about it or give out free candy.

Both English and Scheme have things called words and sentences. These are similar, but not identical. List at least two ways in which English words or sentences are like Scheme words or sentences and at least two ways in which they are not like Scheme words or sentences. Also, make at least one intelligent comment on a classmate's list.

Just so you know, things like "Yeah!" do not count as intelligent responses, no matter how much thought you put into them.

Figure B.3: Comparing English and Scheme, Fall A and Spring B.

## Comparing English and Scheme

This homework activity (Figure B.3) was intended to be given out the day students were first formally introduced to words and sentences. However, the system again lost it, and it was not given out until three labs (one and one-half weeks) later. The goal was to make students think about the limitations of the analogies they might make between English and Scheme.

Although given out late, this activity appeared to have some merit. As in the summer semester, students were less likely to use English to justify their mistakes were more able to recognize their errors when they made them. This activity was given to Fall A and Spring B students.

## B. 3 Additions to Spring B

## An Analogy for Words

The $\mathrm{Pez}{ }^{\circledR}$ activity was used again this semester, although the TAs did not talk very much about it or give out free candy.

Give good definitions for Scheme words and sentences. Make sure you mention how they are or are not like English words and sentences. Give this some thought. Once you submit it, you won't be able to go back and change what you wrote.

Figure B.4: Defining Terms.

## Comparing English and Scheme

This activity was used again this semester, but it was actually given on the day students were formally introduced to words and sentences.

## Defining Terms

This activity (Figure B.4) was also given on the first day students were formally introduced to words and sentences. The goal was to get students to think even harder about words and sentences in English and in Scheme. It may have some merit, as students performed much better in the interviews this semester. However, the "Comparing English and Scheme" homework was given on the same day, as were the WebScheme exercises described on the current page.

## Parentheses Are Important

This activity (Figure B.5) was not specifically inspired by this research, but it fit in quite well. WebScheme is a technology that allows interactive web pages to interpret Scheme. In this case, students were shown Scheme expressions and asked to predict the results. This was the first time we had been able to ask such a question and have the computer check the responses. This appeared to be very helpful. Many students typed in sentences without parentheses and were surprised to see error messages pop up on their screens. While some of these students ignored the error message and moved on to other activities without ever getting the right answer, many asked TAs or lab assistants. In previous semesters, these

Fill in the blanks below to get the right answer. To see if you are right, press the arrow. If you see a green check, you got it.

You will need to wait until you see the word "SchemeHandler" in the upper left corner of the page before you start.

| Scheme Expression | Value |  | Correct? |
| :---: | :---: | :---: | :---: |
| (sentence 'I '(me mine)) |  |  | (8) |
| (sentence '( ) '(is empty)) |  | $1$ | (?) |
| (word 'ab 'cd) |  | $\Gamma$ | ? |
| (sentence 'ab 'cd) |  |  | (8) |
| (sentence 'a (word 'k 'c)) |  |  | (8) |

Figure B.5: WebScheme activity about empty words and sentences.
students would have assumed their parentheses-free answers were correct and would not have thought about it again until forced to by an error in their code.

Students saw this question in lab the first day they were formally introduced to words and sentences. It was first given in Spring B. Because it was introduced along with one new homework and the "Comparing English and Scheme" activity was moved to be on the same day, it is difficult to say if this activity is at all useful.

## Empty

Since Summer 2002, students have been asked to explain (butfirst 'x) and (butfirst ' (x)). When CS 3 started using WebScheme in Spring B, this exercise was changed. Students were now given a WebScheme activity which asked them to fill in two blanks with the results of (butfirst 'x) and (butfirst '(x)). When students left the blanks empty, WebScheme marked them wrong with bright red Xs. Again, many students decided to ask someone for an explanation rather than assuming their answers were correct. Students in Spring B did measurably better on questions involving (""), and a combination of this and the $\mathrm{Pez}^{\circledR}$ activity may have been responsible.

## Appendix C

## A Quick Introduction to Scheme

This appendix provides a very basic introduction to Scheme as a programming language. It is aimed at a reader with minimal programming experience and no need to write programs in Scheme in the future. As such, this appendix will cut some corners and possibly explain things in a way that would not please a textbook writer or programming teacher.

## C. 1 Scheme Basics

Scheme is a conversational language, meaning that while it is possible to use it to write large programs, it is quite convenient to type in little bits of code and get immediate results. To make Scheme do something interesting, you would use a procedure. Procedures are chunks of code that tell Scheme how to do some specific thing. Common procedures include $+,-, *, /$, as well as others that are not so mathematical. Scheme usually needs more than just a procedure to do something interesting. Most procedures need some sort of data to do their jobs. For example, + needs numbers to add together. These additional pieces of information are called arguments.

Once you have a procedure and enough arguments, you can tell Scheme to do something. This is accomplished by typing an open parenthesis, then the procedure, then the arguments,
and finally a closing parenthesis. For example, to tell Scheme to add 3, 5, and 7, you would type ( +357 ). This combination of a procedure and its arguments, surrounded by parentheses, is called an expression. What if you want to do something more complicated, like $3+(5 * 7)$ ? Well, you can tell Scheme to multiply 5 and 7 by typing (* 57 ), and you can tell Scheme to add the 3 by saying (+ 3 (* 5 7)). You can combine as many expressions as you want.

There are several types of data in Scheme. In CS 3, we usually deal with words, sentences, and lists. Words are collections of letters, digits, and other characters. For example, the, yes!, c++, and 888 are all words. Unfortunately, some characters have special meanings in Scheme. To get around this limitation and use characters like ', ,, and . in a word, you need to put quotation marks around it. Thus, to type it's into Scheme, you would actually need to type "it's". Words can contain any number of letters, including zero letters. A word with zero letters is called an empty word, and it is written as "". The quotation marks aren't actually part of the empty word. They are used as delimiters, to show you that there really is a word there.

Scheme data (words and sentences) can look a lot like Scheme code (procedures, expressions, etc.). It is possible to have the word first, but there is also a procedure called first. How can we tell these apart? The solution is to use a' (a quote) in front of anything that Scheme should take exactly as it is typed rather than interpreting it as Scheme code. Thus, when you want to tell Scheme to do something to the word first, you would type 'first. You don't need to quote numbers, since you can't confuse a number with a procedure.

Sentences are collections of words. Scheme sees a sentence as one object, no matter how many words it may contain. To help distinguish a sentence from a bunch of words, sentences are written with parentheses around them: (this is a sentence with 7 words!). Sentences can contain any number of words, even one or zero. An empty sentence is written as a pair of parentheses: ().

Sentences and expressions look a lot alike. You could, if you wanted to, have the sentence (+ 3 4), but this looks just like the expression that tells Scheme to add 3 and 4. How do you tell the two apart? Again, you put a' in front of the sentence. To tell Scheme that you want a sentence that looks like math rather than telling Scheme to do math, you would type ' ( +3 4). You don't need to put the quote in front of words in a sentence, assuming you've quoted the sentence. One quote works for everything in the sentence.

Lists are a lot like sentences, except that they can also contain other lists. Two examples are (this is a list with seven words!) and (this is a list (with another list inside!)). Lists which contain no elements are called null or empty lists and are written as (). As with sentences, if you want to type a list into Scheme, you need to add a '.

Words and sentences were invented just for Simply Scheme, while lists are part of standard Scheme. Sentences are based on lists, but they are easier to use and less overwhelming.

## C. 2 Working with Words and Sentences

There are many things you can do with words and sentences. You can take them apart using the procedures first and butfirst. First takes the first letter out of a word or the first word out of a sentence: (first 'example) is e, while (first '(example sentence)) is example. Butfirst, which can be abbreviated as bf, gets rid of the first letter of a word or the first word of a sentence: (bf 'example) is xample, while (butfirst '(example sentence)) is (sentence). Taking the butfirst of a one-letter word gives you an empty word, while taking the butfirst of a one-word sentence gives you an empty sentence. Taking the first or butfirst of an empty word or sentence causes Scheme to print out an error message, since neither the empty word nor the empty sentence contains anything for you to take out or throw away.

It is important to point out again that a sentence can contain a single word, so taking the butfirst of (example sentence) gives you the sentence (sentence), not the word sentence.

First and butfirst take sentences apart starting from the left. Two similar procedures, last and butlast (abbreviated bl), take sentences apart starting from the right. For example, (last 'sentence) is e, while (butlast 'sentence) is sentenc.

You can also build words and sentences. The procedure word combines the letters of several words into one larger word: (word 'exa 'mple) produces example. Because word puts all of the letters into a new word, the empty word might seem to disappear: (word 'wor "" 'd) gives you word, not wor""d. The empty word has no letters (the quotation marks are not actually a part of the empty word), so it has no letters to contribute to the new word.

The procedure sentence combines several words or sentences into one larger sentence: (sentence 'example 'sentence) produces (example sentence). While sentence can be given words or sentences, it takes all of the words and puts them into a new sentence. This means that while you may see the words from one sentence inside another, you will never see one whole sentence, complete with parentheses, inside another sentence: (sentence '(this is one sentence) '(this is another)) gives you (this is one sentence this is another). Keep in mind that an empty word is still a word, and it acts like all other words when put into a sentence: (sentence "" (is an empty word)) gives you ("" is an empty word).

When programming, it is important to know when you have an empty word or sentence. Scheme has a procedure called empty? that will tell you whether a word or sentence is empty: (empty? 3) will say no, while (empty? "") will say yes. It is important to remember that empty? only says yes when it is given a word or sentence with absolutely nothing inside.

Doing something to every word in a sentence can be very useful. Scheme allows you to do this with the procedure every. Writing (every butfirst ' (example sentence)) tells Scheme to take the butfirst of every word in the sentence, giving you (xample entence).

## C. 3 Working with Lists

Lists are very similar to sentences. You can take them apart using the procedures car and cdr. Car is the list equivalent of first, while cdr is the list equivalent of butfirst. There are no list versions of last or butlast.

Lists can be put together using several different procedures. The only one of these procedures that is important to the thesis is cons, which puts one thing into a list: (cons 'example '(list)) gives you (example list). Cons does not work very well (at least by CS 3 standards) if you do not give it a list as a second argument: don't try (cons 'example 'list). It is also important to know that when lists are combined, it is possible to put one whole list inside another: (cons ' (this is one list) ' (this is another)) gives you ((this is one list) this is another).

The procedure null? tells you whether a list you have is empty (null). A list is empty only when it contains absolutely nothing.

The list version of every is called map.

## C. 4 More Advanced Scheme

The previous sections should explain all of the code used in this thesis, except for questions O 5 and O6. Both of these require an understanding of recursion, which is far beyond the scope of a quick introduction to Scheme. Simply Scheme provides a good introduction to recursion.


[^0]:    ${ }^{1}$ Scheme lists can look a lot like Scheme programs, and symbols can look a lot like Scheme procedure names. See Appendix C for an explanation of this and a quick introduction to Scheme in general.

[^1]:    ${ }^{2}$ This sentence demonstrates some of the confusion that students learning Scheme might encounter. A cleaner version would be "The two sentences (xar) and (zevox) are put together into (xar zevox) in the same way that the word xar and sentence (zevox), the sentence (xar) and word zevox, and the two words xar and zevox are."

[^2]:    ${ }^{3}$ The empty word itself has no letters. The quotation marks are there because neither the Scheme interpreter nor human programmers could make sense of the empty word if it were to be written literally, using zero characters. For a quick introduction to words, sentences, and other relevant details of Scheme, see Appendix C (starting on page 148).

[^3]:    ${ }^{1}$ In an episode of the BBC comedy Father Ted, one character asks where his record collection has gone. Another character hands a single record to him, saying "You need more than one record to have a collection. What you have is a record."

[^4]:    Table 3.5: Relevant or significant changes to the curriculum of each semester.

[^5]:    1 "Good" in this case does not mean "deep" or "technically correct." Parentheses are only part of the textual representation of sentences. A graphical Scheme might use color or some shapes other than parentheses, while the internal Scheme representations of sentences have nothing to do with parentheses, colors, or anything even remotely similar. However, perhaps twenty of all interviewed students have even the faintest idea of anything except textual Scheme; the remaining students should be evaluated in terms of what they have been taught. In the textual representation of Scheme, the key difference between a sentence and a bunch of words sitting together is the presence of parentheses. A "good" understanding, when students know nothing about the internal workings of Scheme, should include the ability to differentiate between sentences and words as well as the ability to write each in such a way that a fellow programmer or computer can identify them. This having been said, the only important elements of a sentence are the words it contains.

[^6]:    ${ }^{1}$ Gender data was not collected during the first two rounds of interviews in Spring A.

[^7]:    ${ }^{2}$ While these paragraphs deal exclusively with the sentence version of the problem, the same observations hold true for the list version.

[^8]:    ${ }^{3}$ Victor Borge's "Phonetic Punctuation" routine, in which he proposes that each punctuation mark have its own sound so that it can be read aloud with other text, draws half of its humor from the absurdity of spoken punctuation. The other half, of course, comes from Borge's none-too-sophisticated choice of sounds.

[^9]:    ${ }^{4}$ The statements "He would come in last in a one-man race" and "He would come in first in a one-man race" have the same logical meaning, but they have very different connotations.

[^10]:    ${ }^{5}$ Thinking about sentences and lists in different ways is not a bad thing. Sentences and lists are different. Sentences are flat (no sentences inside sentences), while lists are deep (lists can be inside lists inside lists. . .). Sentences are symmetric (words can be removed from the left with first and butfirst just as easily as they can on the right with last and butlast), while lists are not (car and cdr work on the left side of a list, but no standard Scheme procedures do the same on the right). However, treating sentences and words differently in the interviews is bad because none of the interview questions dealt with these issues. The reasoning needed to solve a given list question was the same as that needed to solve the equivalent sentence question.

[^11]:    ${ }^{1}$ Most students know on some level that programs should behave consistently, produce reasonable answers, and not give error messages unless they have to, but they do not always know when to apply these ideas. In a later semester, I wrote (first '("")), (empty? '("")), (bf '(1)), (bl '(1)), (bf , (1 2)), (bl
     think of likely results of evaluating each. Students were unsure which results were correct until I proposed using these rules of thumb.

