Generic Prescription for Exam 2 Makeup

1 Overview and Conditions

There will be a makeup exam for exam 2. We will record the maximum of your original exam 2 grade and the grade on the makeup. The makeup exam is optional—you don't have to take it if you are happy with your exam 2 grade. The makeup exam will happen at 9:15am in our usual classroom (103 HEC) on November 9.

However, this will not be a simple retaking of the exam. The main objective is for you to learn the material (and for us to make sure that we have communicated it to you). We expect you to do some work before taking the new exam. You can either do this completely on your own, or see the professor or the TA for help on it.

We suggest that you come see one of us in person if your score was especially low on the exam, say below 40. Also, please see us if you are confused about anything at all that was on the exam, especially what the probems mean.

What we ask before the exam is that you do whatever of the following you find is helpful for your learning of the material. You can simply bring it to the exam as evidence that you have done something. However, it will be better if you show it to one of us earlier if you want some feedback. We will schedule a review session at 9:30am in 202 HEC on November 7 at which you can do some of this. However, you must do something if you wish to take the makeup exam, any must bring whenever you have done with you to the makeup exam as evidence (e.g., printouts or handwritten solutions). You will not be allowed to take the makeup exam without having some evidence of work that you have done.

Finally note that the makeup exam will be a new exam over the same material. In particular, all the problems will be different, and you will see a new grammar for the last problem.

2 Match Your Trouble to Makup Work

2.1 Oz Syntax Trouble

If you had trouble with the syntax of Oz (grammar errors), try the following:

1. Review the reference material for Oz in appendixes B and C of our textbook.
2. Write out a list of the kind of mistakes you make in syntax.
3. Copy, by hand, up to 5 procedures from chapter 3 of the textbook. Check these over for your mistakes by comparing it with the book.
4. Type in all the code you wrote on the exam, and get it to run.
5. Write down up to 3 different examples that use (correctly) the parts of the Oz syntax you made the most mistakes with. Don't just make a difference in names.
6. For all code you write in homeworks and when reviewing for this makeup, write the code out by hand first, carefully distinguishing upper and lower case characters. Then type in the problem, and run it using Oz, noting any syntax errors you make.

2.2 Free and Bound Variable Identifier Occurrences

1. Review section 2.4.3 in the textbook about "Free and bound identifier occurrences." Note especially the box that distinguishes "Bound identifier occurrences and bound variables."

2. Make up another problem or two similar to problems 1 or 2 on the exam and solve them. You can check that identifier occurrences are free if you feed your statement to Oz and it complains that those variable identifiers are "not introduced." When you declare the free variable identifier occurrences, Oz will no longer complain that they are not declared. For example, changing

```
fun {Foo X Y}
   {Plus {Foo {Add 1 X} {Sub3 Y}} 7}
end
```

to

```
declare Foo Plus Add Sub3 in
fun {Foo X Y}
   {Plus {Foo {Add 1 X} {Sub3 Y}} 7}
end
```

makes Oz no longer complain about the free variable identifiers `Foo`, `Plus`, `Add`, and `Sub3`. You can tell that variable identifier occurrences are bound if changing the name to a fresh name results in a complaint from Oz. For example, in the above, changing the occurrence of `X` on the third line to `Z` makes Oz complain about `Z`.

3. Carefully read our solution to the first two problems in homework 3a (the review for exam 2) and relate these to the exam problems in exams 1 and 2.

4. Carefully read test cases for the first two problems in homework 3a (the review for exam 2). Do you understand why all of those test cases are correct?

2.3 Desugaring to Kernel Syntax

1. Read sections 2.3.1 and 2.6 of the textbook about desugaring from the Oz language to the kernel syntax.

2. Section 2.3.1 of the textbook, especially tables 2.1 and 2.2 describe the declarative kernel language's syntax. Find everything that is your answer to the desugaring problems in exams 1 and 2 that are not in that grammar. How are those (sugared) statements and expressions desugared into the kernel syntax?

3. Make up another problem or two similar to problem 3 on exam 2 and solve them. You can check that your solution is in the declarative kernel grammar by referring to the grammar in the textbook's section 2.3.1. You can also check your work by using the "Core Syntax" menu item from the Oz Programming Interface, which approximates kernel syntax, although it still uses infix operators and also does not desugar function and procedure names used in declarations.

4. Carefully read our solution to the third problem in homework 3a (the review for exam 2) and relate it to the exam desugaring problems in exams 1 and 2.

5. Carefully read test cases for the first third problem in homework 3a (the review for exam 2). Do you understand why all of those test cases are correct?

2.4 Recursion on Flat Lists

If you had trouble with problem 4 on the exam.

1. Type in your solution to this problem without any corrections we have made to it. Feed it to Oz. Fix any syntax errors and then see what happens on each of the test cases? Do you understand what happens and what the output is in each case? Now fix it as we indicated. Why does that fix work?

2. Solve the problem 4 ways: by writing out the recursion from scratch, by using `Map`, by using `FoldR`, and by using a **`for`** loop with `collect:`. Get each solution to run using Oz.

3. State a general rule about when you can use `Map` to solve problems involving lists. Why can't you easily use `Map` when your rule is violated?

4. Which of the two arguments to `Map` is the list? Is that a general rule such functions in Oz?

5. Design one or two other problems that involve lists and that can be solved using `Map`. Solve them using `Map` and test your solutions on the computer.

2.5 Tail Recursion on Lists

If you had trouble with problem 5 on the exam.

1. Review sections 3.2, 3.4.2.3–5, and 3.4.3 in the textbook, paying particular attention to the terms "iterative behavior" and "tail recursion."

2. Type in your solution to this problem without any corrections we have made to it. Feed it to Oz. Fix any syntax errors and then see what happens on each of the test cases? Do you understand what happens and what the output is in each case? Now fix it as we indicated. Why does that fix work?

3. There are two ways to solve this problem: (a) Use the list argument itself is used as an accumulator, by using a one-element list as a base case and whenever the list has at least 2 elements, taking the `Min` of the first two elements and recursing on the list formed from the minimum of those elements. (b) Use a helping procedure with an explicit accumulator, `MinSoFar`. Program the other solution, that is the one that is different than the one you did on the exam. Which solution is easier to conceptualize? Which technique is guaranteed to be a way to write tail recursions?

4. Design a problem that can be solved using tail recursion but for which you can't use the given arguments as accumulators. Solve this problem and then test it on the computer.

2.6 FoldR, FoldL, and Other Library Functions

Note: we gave a problem about `FoldR` on the exam. You are also responsible for understanding `FoldL`, `Filter`, `Map`, and other library functions we studied.

1. Review section 3.6.1 in the textbook, paying particular attention to the definitions and uses of `FoldR` and `FoldL`.

2. What are the types of the arguments to `FoldR`, `FoldL`, `Map`, and `Filter`?

3. What is the relationship between the arguments to `FoldR` and the parts of a typical function that does recursion over flat lists? Can all functions that follow the grammar for flat lists be implemented using `FoldR`?

4. Solve 2 or 3 problems in the section on recursion over flat lists in the "Following the Grammar" handout using `FoldR` and again using `FoldL`. Test your solutions on the computer.

5. Design a problem on lists and solve it using `FoldR` and then again using `FoldL`. Solve this problem both ways and then test each on the computer.

6. Can all problems that you can easily solve using tail recursion be solved using `FoldL`?

2.7 Recursion over Window Layouts and Other Grammars

Note: we gave a problem about the Window Layout grammar on the exam. However, you are also responsible for understanding the other grammars in the "Following the Grammar" handout, and indeed for learning the general concept of "Following the Grammar" to the extent that you can apply the concept to an arbitrary grammar.

1. Review the "Following the Garmmar" handout, which is available from the course resources page. Please read the entire thing and do the exercises (noting the answers following some of them in tiny upside-down script). See also section 3.4.2.6 in the textbook.

2. Write up to 4 examples of Oz expressions that are in the grammar for ⟨WindowLayout⟩.

3. Why is `nil` not of type ⟨WindowLayout⟩? Why is a list of ⟨WindowLayout⟩s not in the grammar for ⟨WindowLayout⟩?

4. In what sense, if any, does your solution to the last 2 problems on the exam not "follow the grammar"?

5. Type in your solution to these last 2 problems without any corrections we have made to it. Feed them to Oz. Fix any syntax errors and then see what happens on each of the test cases. Do you understand what happens and what the output is in each case? Now fix it as we indicated. Why does that fix work?

6. Practice doing problems using the window layout grammar and the statement and expression grammar until you can do them quickly. Write them out by hand and check them on the computer.

7. Make up your own problems with a new grammar, or explore some of those you haven't worked with yet in the "Following the Grammar" handout. Write out solutions by hand and check them on the computer.