COP 4020 — Programming Languages 1

Makeup Test on Declarative Programming Techniques

Special Directions for this Test

This test has 8 questions and pages numbered 1 through 9.
This test is open book and notes.
If you need more space, use the back of a page. Note when you do that on the front.
Before you begin, please take a moment to look over the entire test so that you can budget your time.
Clarity is important; if your programs are sloppy and hard to read, you may lose some points. Correct syntax also makes a difference for programming questions.

When you write Oz code on this test, you may use anything in the declarative model (as in chapters 2–3 of our textbook). So you must not use imperative features (such as cells and assignment).
You are encouraged to define functions or procedures not specifically asked for if they are useful to your programming; however, if they are not in the Oz base environment, then you must write them into your test. (This means you can use things in the Oz base environment such as Map, FoldR, Filter, Append, etc.)

For Grading

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>
1. (5 points) In the following statement, written in the kernel language of the declarative model, circle each free variable identifier occurrence (and only the free variable identifier occurrences).

\[
\text{ProdFuns = proc } \{ \text{Ls Acc Return} \} \\
\quad \text{case Ls of} \\
\quad \quad \text{Hd | Tail then local Temp in} \\
\quad \quad \quad \text{Temp = proc } \{ \text{X R} \} \{ \text{Prod Hd X R} \} \text{ end} \\
\quad \quad \text{local Temp2 in} \\
\quad \quad \quad \text{Temp2 = Temp | Acc} \\
\quad \quad \quad \{ \text{ProdFuns Tail Temp2 Return} \} \\
\quad \quad \text{end} \\
\quad \text{end} \\
\text{else } \{ \text{Reverse Acc Return} \} \\
\text{end} \\
\text{end}
\]

2. (10 points) The following is another copy of the same kernel language code as above. On the following copy, circle each bound variable identifier occurrence (and only the bound variable identifier occurrences).

\[
\text{ProdFuns = proc } \{ \text{Ls Acc Return} \} \\
\quad \text{case Ls of} \\
\quad \quad \text{Hd | Tail then local Temp in} \\
\quad \quad \quad \text{Temp = proc } \{ \text{X R} \} \{ \text{Prod Hd X R} \} \text{ end} \\
\quad \quad \text{local Temp2 in} \\
\quad \quad \quad \text{Temp2 = Temp | Acc} \\
\quad \quad \quad \{ \text{ProdFuns Tail Temp2 Return} \} \\
\quad \quad \text{end} \\
\quad \text{end} \\
\text{else } \{ \text{Reverse Acc Return} \} \\
\text{end} \\
\text{end}
\]
3. (5 points) Desugar the following Oz code into kernel syntax by expanding all syntactic sugars. (Assume that
\texttt{Times} is a function that is declared elsewhere.)

\begin{verbatim}
fun {By2 I}
  {Times I 2}
end
\end{verbatim}
4. (10 points) Write a function

    EvenNumbers: <fun $ <List Integer>>: <List Integer>

    that takes a list of integers LoN and returns a list of the even integers in LoN, in their original order. The following are examples, that use the Test procedure from the homework.

    (Test (EvenNumbers nil) '== nil)
    (Test (EvenNumbers [2]) '== [2])
    (Test (EvenNumbers [3]) '== nil)
    (Test (EvenNumbers [1 2 3 4 5 6 7 6 2 3 1]) '== [2 4 6 6 2])
    (Test (EvenNumbers [2 3 4 5 6 7 6 2 3 1]) '== [2 4 6 6 2])
    (Test (EvenNumbers [3 4 5 6 7 6 2 3 1]) '== [4 6 6 2])
5. (10 points) Write a function

\[ \text{SumSquares: } \langle \text{fun} \rangle \{ \langle List \ Number \rangle \} : \langle \text{Number} \rangle \]

that takes a list of numbers, LoN, and returns the sum of their squares.

Your solution must have iterative behavior, and must be written using tail recursion. Don’t use any higher-order functions in your solution. (You are supposed to know what these directions mean.)

The following are examples, that use the Test procedure from the homework.

\{Test \{SumSquares nil\} ‘==’ 0\}
\{Test \{SumSquares [5]\} ‘==’ 25\}
\{Test \{SumSquares [5 10]\} ‘==’ 125\}
\{Test \{SumSquares [1 2 3 4]\} ‘==’ 30\}
\{Test \{SumSquares [7 5 10 12 ~7 ~5 ~10 ~12]\} ‘==’ 636\}
\{Test \{SumSquares [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]\} ‘==’ 0\}

6. (10 points) Write a definition of SumSquares that solves the problem given above, but using FoldL. Do this by filling in the blanks in the following code outline. (Note that these blanks may be larger in size than you need.)

\[ \text{fun } \{ \langle \text{SumSquares} \ LoN\rangle \}
\{ \langle \text{FoldL} \rangle \}
\{ \langle \text{finish code here} \rangle \}
\} \]
end
7. (25 points) This problem is about “sales data records.” Sales data records are defined by the following grammar.

\[
\text{(SalesData)} ::=
\begin{array}{l}
\text{store(address: (String) amounts: (List (Int)))} \\
\text{| group(name: (String) members: (List (SalesData)))}
\end{array}
\]

Write a function

\[
\text{NewAddress: \{fun \ ($ <\text{SalesData}> <\text{String}> <\text{String}>): <\text{SalesData}>}\]
\]

that takes a sales data record \( SD \), two strings \( \text{New} \) and \( \text{Old} \), and returns a sales data record that is just like \( SD \) except that all store records in \( SD \) whose address field’s value is (\( \equiv \) to) \( \text{Old} \) in \( SD \) are changed to \( \text{New} \) in the result.

The following are examples using the Test function from the homework.

\[
\begin{aligned}
\text{Test } \{\text{NewAddress group(name: "StartUP!" members: nil)}
&\quad \text{"Downtown" "50 Washington Ave."}\}
&\quad \text{'}==' \{\text{group(name: "StartUP!" members: nil)}
\}
\text{Test } \{\text{NewAddress store(address: "The Mall" amounts: [10 32 55])}
&\quad \text{"110 Main St." "The Mall"}
&\quad \text{'}==' \{\text{store(address: "110 Main St." amounts: [10 32 55])}\}
\text{Test } \{\text{NewAddress group(name: "Target" members: nil)}
&\quad \text{"NewAddress" "OldAddress")}
&\quad \text{'}==' \{\text{group(name: "Target") members: nil)}
&\quad \text{\{store(address: "The Mall" amounts: [10 32 55])\})
\text{Test } \{\text{NewAddress group(name: "Target") members: nil)}
&\quad \text{\{store(address: "The Mall" amounts: [10 32 55])
&\quad \text{\{store(address: "Downtown" amounts: [4 0 2 0])\})
&\quad \text{\"253 Sears Tower" "The Mall")}
&\quad \text{'}==' \{\text{group(name: "Target") members: nil)}
&\quad \text{\{store(address: "253 Sears Tower" amounts: [10 32 55])
&\quad \text{\{store(address: "Downtown" amounts: [4 0 2 0])\})
\text{Test } \{\text{NewAddress group(name: "ACME") members: nil)}
&\quad \text{\{group(name: "Robucks") members: nil)}
&\quad \text{\{store(address: "The Mall" amounts: [99])
&\quad \text{\{store(address: "Maple St." amounts: [32])\})
&\quad \text{\{group(name: "Target") members: nil)}
&\quad \text{\{store(address: "The Mall" amounts: [10 55])
&\quad \text{\{store(address: "Downtown" amounts: [4])\})
&\quad \text{\"High St." "The Mall")}
&\quad \text{'}==' \{\text{group(name: "ACME") members: nil)}
&\quad \text{\{group(name: "Robucks") members: nil)}
&\quad \text{\{store(address: "High St." amounts: [99])
&\quad \text{\{store(address: "Maple St." amounts: [32])\})
&\quad \text{\{group(name: "Target") members: nil)}
&\quad \text{\{store(address: "High St." amounts: [10 55])
&\quad \text{\{store(address: "Downtown" amounts: [4])\})
\end{aligned}
\]

There is space for your answer on the next page.
Put your answer to the NewAddress problem here.
8. (25 points) This problem is about “expressions” (that encode the abstract syntax of a subset of Oz that corresponds to the one-argument “\(\lambda\)-calculus”).

\[
\text{(Expression)} ::= \\
\text{varIdExp}(\text{Atom}) \\
| \text{funExp}(\text{Atom} \ \text{Expression}) \\
| \text{applyExp}(\text{Expression} \ \text{Expression})
\]

Write a function

\[
\text{FreeVarIds}: \langle \text{fun} \ \langle \langle \text{Expression} \rangle \rangle : \langle \text{List} \ \langle \text{Atom} \rangle \rangle \rangle
\]

that takes an expression \(\text{Exp}\) and returns a list of atoms from each free variable identifier occurrence in \(\text{Exp}\). (The returned list can have duplicate elements.) An atom \(X\) should be in the result just when \(\text{Exp}\) contains a record of the form \(\text{varIdExp}(X)\), and that record is not nested within a record of form \(\text{funExp}(X \ E)\) (since that declares \(X\) as a formal parameter).

The following are examples using the \text{Test} function from the homework. Note that your answer should produce a list with the same elements, but the elements need not be in the same order as shown for the test outputs.

\[
\begin{align*}
\{\text{Test} \ \{\text{FreeVarIds} \ \text{varIdExp}(z)\} \ '==' \ [z]\} \\
\{\text{Test} \ \{\text{FreeVarIds} \ \text{varIdExp}(q)\} \ '==' \ [q]\} \\
\{\text{Test} \ \{\text{FreeVarIds} \ \text{applyExp}(\text{varIdExp}(q) \ \text{varIdExp}(z))\} \ '==' \ [q \ z]\} \\
\{\text{Test} \ \{\text{FreeVarIds} \ \text{applyExp}(\text{varIdExp}(x) \ \text{varIdExp}(x))\} \ '==' \ [x \ x]\} \\
\{\text{Test} \ \{\text{FreeVarIds} \ \text{funExp}(\text{varIdExp}(x))\} \ '==' \ \text{nil}\} \\
\{\text{Test} \ \{\text{FreeVarIds} \ \text{funExp}(\text{varIdExp}(q) \ \text{varIdExp}(z))\}) \ '==' \ [q]\} \\
\{\text{Test} \ \{\text{FreeVarIds} \ \text{funExp}(\text{varIdExp}(q) \ \text{varIdExp}(z))\}) \ '==' \ [z]\} \\
\{\text{Test} \ \{\text{FreeVarIds} \ \text{funExp}(\text{varIdExp}(q) \ \text{varIdExp}(z)))\}) \ '==' \ [q \ z]\} \\
\{\text{Test} \ \{\text{FreeVarIds} \ \text{funExp}(\text{varIdExp}(q) \ \text{varIdExp}(z)))\}) \ '==' \ [z]\} \\
\{\text{Test} \ \{\text{FreeVarIds} \ \text{funExp}(\text{varIdExp}(q) \ \text{varIdExp}(z)))\}) \ '==' \ [x \ x]\} \\
\{\text{Test} \ \{\text{FreeVarIds} \ \text{funExp}(\text{varIdExp}(q) \ \text{varIdExp}(z)))\}) \ '==' \ [z]\} \\
\{\text{Test} \ \{\text{FreeVarIds} \ \text{funExp}(\text{varIdExp}(q) \ \text{varIdExp}(z)))\}) \ '==' \ [x \ z]\} \\
\end{align*}
\]

There is space for your answer on the next page.
Put your answer to the FreeVarIds problem here.