Fall, 2007 Name: _____

COP 4020 — Programming Languages 1

Test on Message Passing, the Relational Model, and Programming Models vs. Problems

Special Directions for this Test

This test has 6 questions and pages numbered 1 through 7.

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 message passing or relational model (as in chapters 5 and 9 of our textbook). The problem will say which model is appropriate. However, you must not use imperative features (such as cells and assignment) or the library functions `IsDet` and `IsFree`.

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 functions in the Oz base environment such as `Map`, `FoldR`, `Filter`, `Append`, `Max`, etc.)

For Grading

Problem	Points	Score
1	10	
2	10	
3	40	
4	10	
5	20	
6	10	

1. (10 points) Circle just the statement or statements below that are correct, and then briefly justify your answer.

   (a) The message passing model can simulate cells like those found in the explicit state model.

   (b) The message passing model is referentially transparent, because the message passing model does not include any way to update state.

   (c) The message passing model does not need to be defined using a mutable store, because none of its primitives update any state.

   (d) The `Send` primitive in the message passing model needs to be defined using mutable state, because it updates the store variable that tracks the end of the stream.

2. (10 points) Circle just the statement or statements below that are correct, and then briefly justify your answer.

   (a) The relational model is most useful when you know an algorithm for solving your problem and you have a lot of time to implement it carefully.

   (b) The relational model is most useful when you do not know an algorithm for solving your problem or you do not have much time to implement such an algorithm carefully.

   (c) The relational model is most useful when the problem's search space is very large or infinite.

   (d) The relational model is most useful when the problem's search space is finite and not too large.

3. (40 points) For each of the following programming problems, name the best programming model for solving the problem, and briefly explain why that model is best for the problem. When in doubt, favor the least expressive model (i.e., the one with the fewest features) that can solve the problem. (Choose from among the programming models we studied this semester.)

   (a) A server that allows several different users to share book recommendations. Each recommendation is a pair consisting of a book name and a string describing why the book is good. Users may post a new recommendation or retrieve the current set of recommendations. Users may be posting and asking for recommendations concurrently, but the server need only handle one request at a time.

   (b) A library that approximates mathematical functions to varying degrees of accuracy. For example, want to provide all the trigonometric functions (such as sine and cosine), with an accuracy that users can select. It's important that you develop these functions in a way that minimizes your overall programming time for the collection of functions. You are told that they can all be approximated by "Tailor series" formulas.

   (c) A function that solves a Sudoku puzzle. A Sudoku puzzle is a 9×9 grid with some numbers filled in. A solution fills in the other numbers so that each column, each row, and each of the nine 3×3 boxes in the grid contain the digits from 1 to 9, only one time each (that is, exclusively). Since you're writing this program for a friend and in your spare time, it's important that you not spend too much time writing the code. Also, you don't care about how fast the program runs.

   (d) A function that manipulates abstract syntax trees (recursive data) for query optimization. This function has to take a query's abstract syntax tree as an argument, and produces one that has the same functionality, but which has, for example, certain subtrees that represent expensive queries converted to other subtrees that are less expensive.

4. (10 points) Using Oz's message passing model, write a function `NewGauge` that takes no arguments and returns a port object that stores an integer value in its state and that can respond to the messages `up`, `down`, and `fetch(Var)`. Initially the value stored in the port object is 0. The message `up` increments this value and `down` decrements it. The `fetch(Var)` message binds `Var` to the current value. The following are some examples, written using the `Test` function as in the homework.

```
declare
MyGauge = {NewGauge}
{Test local R in {Send MyGauge fetch(R)} R end '==' 0}
{Send MyGauge up}
{Test {Send MyGauge fetch($)} '==' 1}
{Send MyGauge up}
{Test {Send MyGauge fetch($)} '==' 2}
{Send MyGauge down}
{Test {Send MyGauge fetch($)} '==' 1}
{Send MyGauge up}
{Send MyGauge up}
{Test {Send MyGauge fetch($)} '==' 3}

MyGauge2 = {NewGauge}
{Test {Send MyGauge2 fetch($)} '==' 0}
{Send MyGauge2 down}
{Test {Send MyGauge2 fetch($)} '==' ~1}
{Send MyGauge2 down}
{Test {Send MyGauge2 fetch($)} '==' ~2}
for I in 1..12 do {Send MyGauge2 up} end
{Test {Send MyGauge2 fetch($)} '==' 10}
```

You should use `NewPortObject`, from the textbook and homework, in your solution.

Please write your answer below.

```
\insert 'NewPortObject.oz'  % assume this is already written
declare
```

5. (20 points) Using Oz's message passing model, write a function `NewAuctioneer` that takes no arguments and returns a port object that acts as an auctioneer (an agent that is selling an item in an auction).

   This port object understands several messages. The `bid(Amt DidIWin)` message places a bid on the item, where `Amt` is a non-negative integer (the number of dollars bid) and `DidIWin` is an unbound store variable.

   The `stop` message ends the auction. When the `stop` message is received, the `DidIWin` variable in the first `bid` message received whose `Amt` was the largest is bound to `true`, and all other `DidIWin` variables in other `bid` messages are bound to `false`. (Note that there may be no bids. In case of a tie, the first `bid` message with the highest amount wins. If any `bid` message is received after the `stop` message, it does not win.)

   The following are some examples, written using the `Test` function as in the homework.

```
declare
MyAer = {NewAuctioneer}
local Status1 Status2 Status3 Status4 in
   {Send MyAer bid(17 Status1)}
   {Send MyAer bid(8 Status2)}
   {Send MyAer bid(27 Status3)}
   {Send MyAer bid(24 Status4)}
   {Send MyAer stop}
   {Test Status1 '==' false}
   {Test Status2 '==' false}
   {Test Status3 '==' true}
   {Test Status4 '==' false}
end

MyAer2 = {NewAuctioneer}
local Status1 Status2 Status3 Status4
   Status5 Status6 Status7
in
   {Send MyAer2 bid(100 Status1)}
   {Send MyAer2 bid(15 Status2)}
   {Send MyAer2 bid(99 Status3)}
   {Send MyAer2 bid(99 Status4)}
   {Send MyAer2 bid(100 Status5)}
   {Send MyAer2 bid(100 Status6)}
   {Send MyAer2 bid(100 Status7)}
   {Send MyAer2 stop}
   % first of the highest bids wins
   {Test Status1 '==' true}
   {Test Status2 '==' false}
   {Test Status3 '==' false}
   {Test Status4 '==' false}
   {Test Status5 '==' false}
   {Test Status6 '==' false}
   {Test Status7 '==' false}
end

% Stopping with no bids
MyAer3 = {NewAuctioneer}
{Send MyAer3 stop}
% Bidding after an auction is over
{Test {Send MyAer3 bid(50 $)} '==' false}
```

   You should use `NewPortObject`, from the textbook and homework, in your solution.

   Hint: you may want to use helping functions or procedures.

   Please write your answer in the space provided on the next page.

Please write your answer below.

```
\insert 'NewPortObject.oz'  % assume this is already written
declare
```

6. (10 points) Using Oz's relational model, write a procedure `PositionOf` such that `{PositionOf Ls E Index}` succeeds when `Ls` is a list in which the element `E` occurs at the `Index` position (counting from 1) and fails otherwise. The following are examples, using the standard `SolveAll` and `SolveFirst` functions described in class.

```
{Test {SolveAll proc {$ Ans} {PositionOf nil 0 Ans} end} '==' nil}
{Test {SolveAll proc {$ Ans} {PositionOf 0|nil 0 Ans} end} '==' [1]}
{Test {SolveFirst proc {$ Ans} {PositionOf [4 0 2 0] 4 Ans} end} '==' 1}
{Test {SolveAll proc {$ Ans} {PositionOf [4 0 2 0] 0 Ans} end} '==' [2 4]}
{Test {SolveFirst proc {$ Ans} {PositionOf [4 0 2 0] 2 Ans} end} '==' 3}
{Test {SolveAll proc {$ Ans} {PositionOf [4 0 2 0] 5 Ans} end} '==' nil}
{Test {SolveAll proc {$ Ans} {PositionOf [5 0 2 1 4 0 2 0 88] 0 Ans} end}
 '==' [2 6 8]}
{Test {SolveFirst proc {$ X} {PositionOf [99 22 X 55] 7 3} end} '==' 7}
{Test {SolveFirst proc {$ Ind} {PositionOf [99 22 8 55] Ind 3} end} '==' 8}
```