Programming Language Concepts, cs2104
Tutorial 10. Answers

Exercise 1. (`WaitOr` and `WaitSome`) One problem that occurs quite often in practice is to wait until at least one out of two variables becomes bound. For this purpose, Oz provides the procedure `{WaitOr X Y}`. It suspends until `X` or `Y` becomes bound. Write an Oz procedure able to simulate `{WaitOr …}`.
For instance, Mozart provides the procedure `{Record.waitOr R ?LI}` which blocks until at least one field of *R* is determined. It returns the feature *LI* of a determined field and it raises an exception if *R* is not a proper record, that is, if *R* is a literal. For example,

`{Browse {Record.waitOr a(_ b:1)}}` displays `b`
`{Browse {Record.waitOr a(2 b:_)}}` displays `1`
`{Browse {Record.waitOr a(_ b:_)}}` blocks.

Moreover, write a procedure `{WaitSome Xs}` that suspends the executing thread until at least one variable from the list `Xs` becomes bound.

Solution.
The idea is to create a thread for `X` and `Y` that suspends until one of them is bound. If it becomes bound, the thread binds a variable shared among these two threads (say `B`). Execution then continues as soon as `B` is bound.

```
declare
proc {WaitOr X Y}
B in
   thread
      {Wait X}
      B=true
   end
   thread
      {Wait Y}
      B=true
   end
   {Wait B}
end
```

The idea is to create a thread for each element of list `Xs` that suspends until the element is bound. If it becomes bound, the thread binds a variable shared among all threads (here `Y`). Execution then continues as soon as `Y` is bound:

```
declare
proc {WaitSome Xs}
  Y
in
  {ForAll Xs proc {$ X} thread {Wait X} Y=true end end}
  {Wait Y}
end
```

Exercise 2. (cells – reference and value) Explain what and why the following Oz program will display:

```
declare
X = {NewCell 0}
{Assign X 5}
Y = X
{Assign Y 10}
{Browse {Access X} == 10}
{Browse X == Y}
Z = {NewCell 10}
{Browse Z == Y}
{Browse @X == @Z}
```

Solution. It will display `true`, `true`, `false`, `true` since `X` and `Y` refer to the same cell, while `Z` has a different address (but the same integer stored inside).

Exercise 3. (arrays) Write an Oz function which takes `N` as the input and returns the array <1!, 2!, 3!, …, `N`!>, where `N`! means 'factorial of `N`' (that is, `N`!=1*2*…*`N`).

Solution.

```
declare
fun {MakeFactorialArray N}
 A = {NewArray 1 N 1}
in
 for I in 2..N do
   A.I := A.(I-1)*I
 end
 A
end
proc {DisplayArray A N}
   for I in 1..N do
      {Browse A.I}
   end
end
{DisplayArray {MakeFactorialArray 5} 5}
```

Another way to display an array is to translate it into a record, then use the records' display. Here it is this solution:

`{Browse {Array.toRecord a {MakeFactorialArray 5}}}`

will display `a(1 2 6 24 120)`.

Exercise 4. (call by value and call by reference) Explain what and why the following Oz program will display:

```
declare
proc {F A}
   A:=@A+1
   A:=@A*@A
end
proc {G A}
   E={NewCell A}
in
   E:=@E+1
   E:=@E*@E
end
local
   C={NewCell 0}
   D={NewCell 1}
in
   C:=5
   D:=6
   {Browse @C#@D}
   {F C}
   {G @D}
   {Browse @C#@D}
end
```

Solution. It will display `5#6`, `36#6` since `C` is passed by variable (reference) and `D` is passed by value.

Exercise 5. Consider the following Oz procedures that can be used to capture relationships between people:

```
proc {Male X}
    choice X=richard | X= john | … end
end
proc {Female X}
    choice X=susan | X=amy | … end
end
proc {Parent X Y}  // X is the parent of Y
    choice X=susan Y=john | X=richard Y=john | … end
end
```

Based on the above relations, we can define a new procedure which determines if X is a son of Y, as follows:

```
proc {Son X Y}  // X is the son of Y
    {Parent Y X} {Male X}
end
```

In a similar fashion, write new non-deterministic procedures for the following relationships.

```
proc {Mother X Y}  // X is the mother of Y

proc {GrandPa X Y}  // X is the grandfather of Y

proc {Brother X Y}  // X is a brother of Y

proc {Uncle X Y}  // X is a uncle of Y

proc {Descendant X Y}  // X is descendant of Y
```

Solution.

```
proc {Mother X Y}
  {Parent X Y} {Female X}
end

proc {GrandPa X Y}
  Z in
  {Parent X Z} {Parent Z Y} {Male X}
end

proc {Brother X Y}
  Z in
  {Parent Z X}
  {Parent Z Y}
  {Male X}
  if X==Y then fail end
end

proc {Uncle X Y}
  P in
  {Brother X P}
  {Parent P Y}
end

proc {Descendant X Y}
  choice
    {Parent Y X}
  []
    Z in
    {Parent Y Z}
    {Descendant X Z}
  end
end
```