Lazy Evaluation

R K Joshi

Something strange in the following implementation?

```
def append(l1,l2):
  x = isnil(l2)(0,1)
  if (x==0): return (l1)
  h = head (l2)
  l = push_back(l1,h)
  return (append(l,tail(l2)))

def _append(l1,l2):
  return( isnil(l2) (l1,
            append( push_back(l1,head(l2)), tail(l2))))
```

Something wrong with this implementation?

```
x = 3
y = 0
v = iszero(y) ('undef',x/y)

print v
```

Use of Lazy Evaluation

If we don't need a particular argument, why evaluate it?

If we had this lazy evaluation, we can specify functions neatly without having to use an if-then-else statement and local variables

The following style can be eliminated!

```
if (y==0): return ('undef')
else: return (x/y)
```

Lazy Evaluation in Programming Languages

It's also called as 'call by need' parameter passing method

as against call by value

Not all languages support lazy evaluation

It's a major feature of Functional Programming Languages, which makes it possible to specify functions in the declarative style as opposed to imperative control constructs based style

Using Lambda Expressions to Obtain the Effect of Lazy Evaluation

```
def close(x):
        return (lambda f: f(x))
val = lambda x: x
true = lambda x,y:x
false = lambda x,y:y
def iszero(x) :
        if(x==0): return(true)
        else: return (false)
```

```
def nildiv(x,y):
        return (lambda f:
                close("undefined value")(f))

def actualdiv(x,y):
        return (lambda f: close(x/y)(f))

def div(x,y):
        return (iszero(y)
                (nildiv(x,y),actualdiv(x,y)))
```

```
def close(x):
    return (lambda f: x)
val = lambda x: x
true = lambda x,y:x
false = lambda x,y:y
def iszero(x) :
    if(x==0): return(true)
    else: return (false)
```

```
def nildiv(x,y):
    return (lambda f:
        close("undefined value")(f))
def actualdiv(x,y):
    return (lambda f: close(x/y)(f))

def div(x,y):
    return (iszero(y)
        (nildiv(x,y),actualdiv(x,y)))
```

```
c = div (1,0)
print (c(val))

c = div (4,2)
print (c(val))
```