COMP 410
Fall 2019

Nondeterministic Execution with Monadic Iterators

Assume you have an `Iterator` implementation with the following available operations (using Scala syntax):

```
// creates iterator holding a
def unit[A](a: A): Iterator[A]

// creates empty iterator
def mzero[A](): Iterator[A]

// effectively OR
def mplus[A](left: Iterator[A], right: Iterator[A]): Iterator[A]

// effectively AND
def bind[A, B](it: Iterator[A],
               f: A => Iterator[B]): Iterator[B]
```

Consider each of the following Prolog procedures. Write an equivalent version which acts as a generator of values, using the monadic iterator operations above.

1.)

```
foo(1).
foo(2).
foo(3).

def foo(): Iterator[Int] =
```

2.)

```
exp(_, num(0)).
exp(Bound, plus(E1, E2)) :-
  Bound > 0,
  NewBound is Bound - 1,
  exp(NewBound, E1),
  exp(NewBound, E2).
```

```
sealed trait Exp
// Num(0) is equivalent to num(0) in Prolog
case class Num(i: Int) extends Exp
// Plus(foo, bar) is equivalent to plus(foo, bar) in Prolog
case class Plus(e1: Exp, e2: Exp) extends Exp

def exp(bound: Int): Iterator[Exp] =
```