```
import unittest

def make_monoid(S, i, op1):
  S.identity=i
  S.__add__ = op1
  return S

def make_tests(monoid, name):
  class check_monoids(unittest.TestCase):
    def test_closure(self):
      print("### Checking Closure on {0}: ".format(name))
      checks = [(x+y).inSet()
         for x in monoid.signature()
            for y in monoid.signature()]
      self.assertTrue(all(checks))
    def test_associativity(self):
      print("### Checking Associativity on {0}: ".format(name))
      checks = [((x+y)+z)==(x+(y+z))
         for x in monoid.signature()
            for y in  monoid.signature()
               for z in monoid.signature()]
      self.assertTrue(all(checks))
    def test_identity(self):
      print("### Checking Identity {1} on {0}: ". \
                        format(name, monoid.identity))
      checks = [monoid.identity in monoid.signature()]
      for x in monoid.signature():
        checks += [(monoid.identity+x) == x]
      self.assertTrue(all(checks))
  return check_monoids

class Z7:
  def __init__(self, val): self.value = val
  def __eq__(self, other): return self.value == other.value
  def __repr__(self): return str(self.value)
  def signature(): return [Z7(x) for x in range(7)]
  def inSet(self): return (self in Z7.signature())

def firstn(g, n):
   for i in range(n):
      yield next(g)

def gabstar():
  yield ABstar("")
  strings = ["a", "b"]
  while True:
    for elem in strings: yield ABstar(elem)
    tmp = []
    for elem in strings:
      tmp += [elem+"a"]+[elem+"b"]
    strings=tmp

def ngen():
  val = 0
  yield N(val)
  while True:
    val += 1
    yield N(val)

class ABstar:
  def __init__(self, str): self.value = str
  def __eq__(self, other): return self.value == other.value
  def __repr__(self): return self.value
  def signature(n=50): return firstn(gabstar(),n)
  def inSet(self): return (self in gabstar())

class N:
  def __init__(self, val): self.value = val
  def __eq__(self, other): return self.value == other.value
  def __repr__(self): return str(self.value)
  def signature(n=50): return [N(x) for x in range(n+1)]
  def inSet(self): return (self in ngen())

monoid_z7 = make_monoid(Z7, Z7(1),
    lambda x,y: Z7(0 if (y.value == 0) else (x.value//y.value)%7))
test_z7 = make_tests(monoid_z7, "(Z7, *)")

monoid_abstar = make_monoid(ABstar, ABstar(""),
    lambda x,y: ABstar((x.value)+(y.value)))
test_abstar = make_tests(monoid_abstar, "(AB*, *)")

monoid_n = make_monoid(N, N(0), lambda x,y: N(x.value+y.value))
test_n = make_tests(monoid_n, "(N, +)")

all_tests = [test_n, test_abstar, test_z7]
suite = unittest.TestSuite()

if __name__ == "__main__":
  for tc in all_tests:
    suite.addTests(unittest.TestLoader().loadTestsFromTestCase(tc))
  unittest.TextTestRunner(verbosity=2).run(suite)
```