ESTRUCTURAS DE DATOS Y ALGORITMOS

CURSO 2009 – PRÁCTICO 3

SOLUCIÓN

Ejercicio 2 :: Funcion: *IsIncluded*

1. Introducción

La solución del ejercicio se realizará en dos etapas. Primero se analizará el problema presentando finalmente una solución el alto nivel, utilizando para ello la notación funcional vista en el práctico anterior. Luego, a partir de esta definición funcional de la solución, se llegará a la implementación de la misma en C/C++.

2. Análisis del problema (Alto Nivel)

La función **IsIncluded** determina si, dadas dos listas **l** y **p**, la lista **p** está incluida en la lista **l**. Analicemos entonces todos los casos posibles:

1. Sean **l** y **p** listas vacías entonces claramente **p** *está* incluida en **l**.
2. Sea **l** vacía y **p** no vacía entonces claramente **p** *no está* incluida en **l**.
3. Sea **l** no vacía y **p** vacía entonces **p** *está* incluida en **l** ya que la lista vacía forma parte de toda lista.
4. Sean ambas listas no vacías. En este caso pueden darse dos situaciones. O bien la lista **p** está contenida al principio de la lista **l** (o sea **p** es prefijo de **l**) con lo cual queda determinado de inmediato que **p** *está* incluida en **l**, o bien se tendrá que seguir recorriendo la lista **l** para determinar si **p** está o no incluida en ella.
   De aquí se desprende que vamos a necesitar una función auxiliar que nos permita determinar si, dadas dos listas **l** y **p**, **p** es prefijo de **l**.

```
IsIncluded: LNat x LNat -> BOOLEAN

IsIncluded ([], []) = true
IsIncluded (x.xs, []) = true
IsIncluded ([], y.ys) = false
IsIncluded (x.xs, y.ys)=IsPrefix(x.xs,y.ys) OR IsIncluded (xs, y.ys)
```

Nótese que no existe ninguna diferencia entre los dos primeros casos considerados por lo cual, se podrían expresar como un único caso: `IsIncluded ( xs, []) = true`.

```
IsPrefix : LNat x LNat -> BOOLEAN

IsPrefix ([], []) = true
IsPrefix (x.xs, []) = true
IsPrefix ([], y.ys) = false
IsPrefix (x.xs, y.ys)=  if (x == y)
                              IsPrefix (xs, ys)
                        else
                              false
```

Conviene recordar que si consideramos una evaluación en circuito corto (lazy o perezosa) el último caso es equivalente a:

```
IsPrefix (x.xs, y.ys) = (x == y) and IsPrefix (xs, ys).
```

Nótese que nuevamente no existe ninguna diferencia entre los dos primeros casos considerados por lo cual, se podrían expresar como un único caso: `IsPrefix (xs, []) = true`.

La función **IsPrefix** determina si, dadas dos listas **l** y **p**, **p** es prefijo de **l**, esto quiere decir si **p** está contenida al principio de **l**:

Ej:

l = [1 2 3 4 5] , **p** = [1 2 3] ⇒ **p** es prefijo de **l**
l = [1 2 3 4 5] , **p** = [2 3 4] ⇒ **p** no es prefijo de **l**

3. La implementación

Para la implementación de la función en C/C++, se supone que existe un juego de cinco funciones básicas que podremos utilizar. Estas cinco funciones básicas se corresponden con los elementos básicos de la notación funcional que hemos utilizado hasta el momento.

Veamos dichas funciones:

```
LNat* Null();
// Crea la lista vacía. Esto es []

LNat* Cons(int x, LNat*& l);
// Inserta un elemento al principio de la lista.
// Esto es, dado el elemento x y la lista xs, retorna x.xs

bool IsEmpty(LNat* l);
// Verifica si la lista está vacía.
// Esto es, verifica si la lista es igual a []

int Head(LNat* l);
// Retorna, si la lista no es vacía, el primer elemento de la lista.
// Esto es, dada la lista x.xs retorna x

LNat* Tail(LNat* l);
// Retorna, si la lista no es vacía, la lista sin su primer elemento.
// Esto es, dada la lista x.xs retorna xs
```

Entonces utilizando este juego de funciones básicas pasamos de la notación funcional a la implementación en C/C++ obteniendo las siguientes soluciones:

```
bool IsIncluded(LNat *l, LNat *p) {

      if (IsEmpty(p)) {
            return true;
      } else {
            if (IsEmpty(l)) {
                  return false;
            } else {
                  return (IsPrefix(l, p) || IsIncluded(Tail(l), p));
            }
      }
}
```

```
bool IsPrefix(LNat *l, LNat *prefix) {

      if (IsEmpty(prefix)) {
            return true;
      } else {
            if (IsEmpty(l)) {
                  return false;
            } else {
                  if (Head(prefix) == Head(l)) {
                        return IsPrefix(Tail(l), Tail(prefix));
                  } else {
                        return false;
                  }
            }
      }
}
```

Sacando partido del circuito corto la función **IsPrefix** puede ser escrita de la siguiente forma:

```
bool IsPrefix(LNat *l, LNat *prefix) {

      if (IsEmpty(prefix)) {
            return true;
      } else {
            if (IsEmpty(l)) {
                  return false;
            } else {
                  return ((Head(prefix) == Head(l)) &&
                          (IsPrefix(Tail(l),Tail(prefix))));
            }
      }
}
```

Ejercicio 3 :: Funcion: *Reverse*

1. Introducción

La solución del ejercicio se realizará en dos etapas. Primero se analizará el problema presentando finalmente una solución en alto nivel, utilizando para ello la notación funcional vista en el práctico anterior. Luego, a partir de esta definición funcional de la solución, se llegará a la implantación de la misma en C/C++.

2. Análisis del problema (Alto Nivel)

La función **Reverse** , dada una lista general **lg**, la invierte sin invertir sus sublistas.
Analicemos entonces todos los casos posibles.

1.- Sea **lg** vacía entonces claramente el resultado de invertirla es también una lista vacía.
2.- Sea **lg** no vacía entonces de a uno en uno se debe ir tomando el primer elemento de la lista general **lg** y desplazarlo hasta el final de la misma. Para ello utilizaremos la función auxiliar **SnocLG**, que dada una lista **p**, la coloca al final de una lista general **lg.**

```
Reverse: LGNat -> LGNat

Reverse ([]) = []
Reverse (l.lg) = SnocLG (l, Reverse (lg))


SnocLG: LNat x LGNat -> LGNat

SnocLG (p, []) = [p] = p.[]
SnocLG (p, l.lg) = l.SnocLG(p,lg)
```

3. La implementación

Para la implementación de la función en C/C++, se supone que existe un juego de cinco funciones básicas que podremos utilizar. Estas cinco funciones básicas se corresponden con los elementos básicos de la notación funcional que hemos utilizado hasta el momento.
Veamos dichas funciones:

```
LGNat* NullLG();
// Crea la lista general vacía. Esto es []

LGNat* ConsLG(LNat* l, LGNat*& lg);
// Inserta un elemento al principio de la lista general.
// Esto es, dado el elemento l y la lista general lg, retorna l.lg

bool IsEmptyLG(LGNat* lg);
// Verifica si la lista general está vacía.
// Esto es, verifica si la lista general es igual a [].

LNat* HeadLG(LGNat* l);
// Retorna, si la lista general no es vacía, el primer elemento de la
// lista general.
// Esto es, dada la lista general l.lg retorna l.

LGNat *TailLG(LGNat* l);
// Retorna, si la lista general no es vacía, la lista general sin su
// primer elemento.
// Esto es, dada la lista general l.lg retorna lg.
```

Entonces utilizando este juego de funciones básicas pasamos de la notación funcional a la implementación en C/C++ obteniendo las siguientes soluciones:

```
LGNat* Reverse(LGNat* lg) {

      if (IsEmptyLG(lg)) {
            return NullLG();
      } else {
            return SnocLG(HeadLG(lg), Reverse(TailLG(lg)));
      }
}

LGNat* SnocLG(LNat* l, LGNat* lg) {

      if (IsEmptyLG(lg)) {
            return ConsLG(l, lg);
      } else {
            return SnocLG(HeadLG(lg), SnocLG(l, TailLG(lg)));
      }

}
```