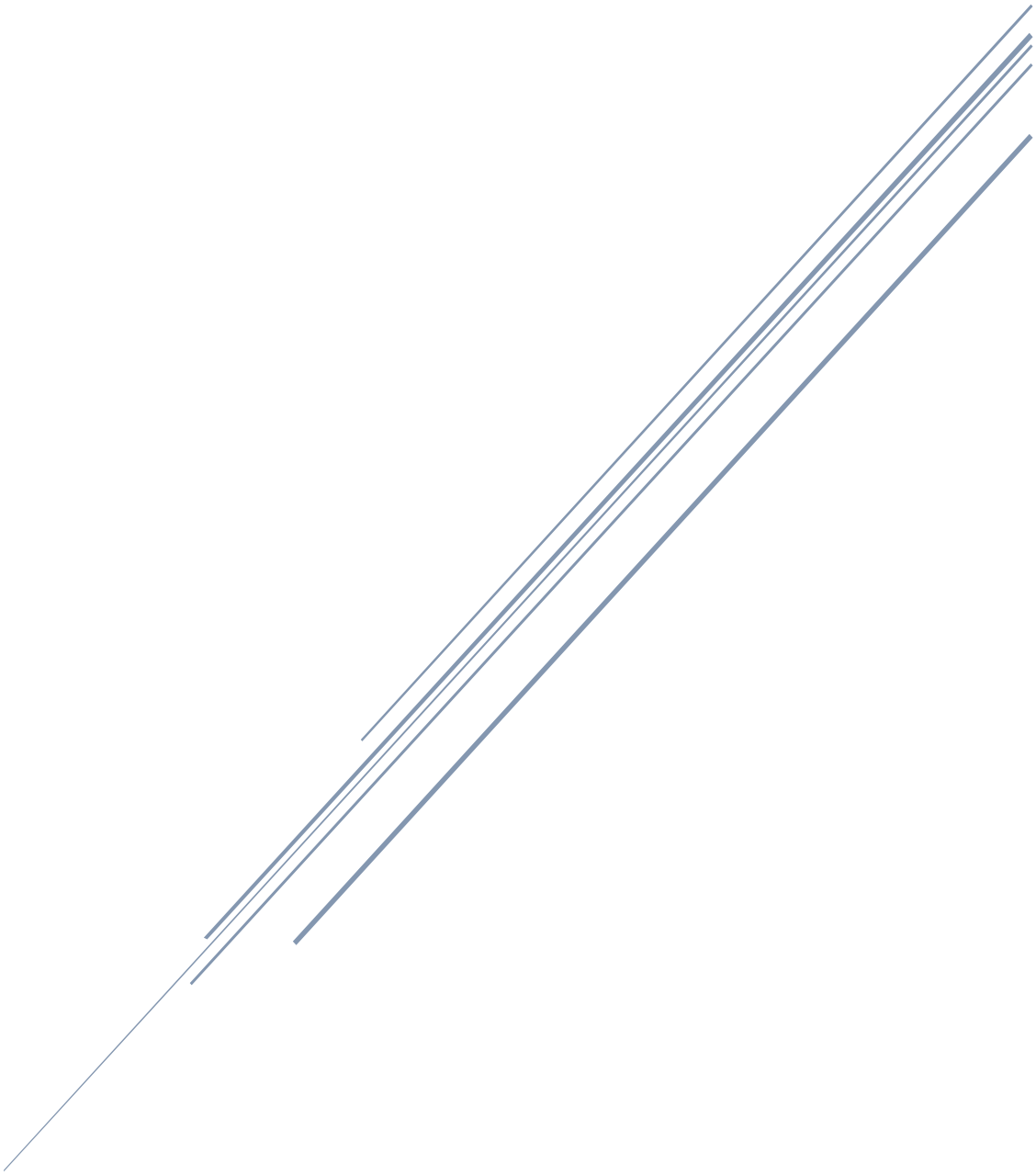
PRACTICA 1

LPM - Laboratorio de programación multiparadigma



MITSS
Sergi Sanz Carreres

Índice:

1.- ¿Qué se obtiene tras ejecutar el siguiente comando y que ecuación habría que añadir?

Maude> red in BATTERY-HASKELL : consume(- ^ - ^ nil) .

Al ejecutar el comando obtenemos la siguiente salida:

```
Maude> red in BATTERY-HASKELL : consume(- ^ - ^ nil) .
reduce in BATTERY-HASKELL : consume(- ^ - ^ nil) .
rewrites: 2 in 0ms cpu (0ms real) (~ rewrites/second)
result Battery: - ^ - ^ consume(nil)
```

Donde podemos observar que en el programa original devuelve *consume(- ^ - ^ nil)* de forma que como no se puede consumir una batería vacía devuelve la misma batería.

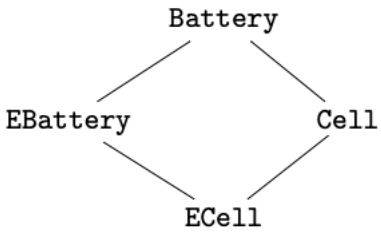
Por tanto, se ha añadido la siguiente ecuación al programa para resolver este problema:

eq consume(nil) = nil .

De manera que al ejecutar el comando anterior obtenemos el siguiente resultado:

```
Maude> red in BATTERY-HASKELL : consume(- ^ - ^ nil) .
reduce in BATTERY-HASKELL : consume(- ^ - ^ nil) .
rewrites: 3 in 0ms cpu (0ms real) (~ rewrites/second)
result Battery: - ^ - ^ nil
```

2.- Gráficamente, se puede representar el orden parcial de la relación de subtipado como un grafo:



Muestra un ejemplo de un término que sea de tipo EBattery pero no ECell y un término que sea de tipo Battery pero no EBattery ni Cell.

Un ejemplo que se podría aplicar tanto para un termino que sea de tipo EBattery pero no ECell sería nil. Al igual que para el ejemplo de un término que sea de tipo Battery pero no EBaterry ni Cell sería cualquiera batería distinta de nil (_ ^ _).

3.- ¿Para qué nos puede servir diferenciar dentro del tipo Cell el subtipo ECell?

El principal motivo es poder representar una celda que contiene batería de una que está vacía, de esta forma utilizando esta implementación evitaremos que en la expresión resultante aparezca nil.

4.- Para una batería de 2 celdas, muestra el conjunto de valores que pueden tomar una variable de tipo EBattery y una variable de tipo Battery.

Para una batería de 2 celdas el conjunto de valores que puede adoptar una variable de tipo EBattery es: _ ^_

Mientras que una batería de tipo Battery puede tener un conjunto de valores distintos que son las siguientes: o ^ o, o ^ + , o ^ - , + ^ o , - ^ o , + ^ + , + ^ - , - ^ +

5.- ¿Qué tipo se obtiene tras ejecutar el siguiente comando y qué significa?

Maude> red in BATTERY-MAUDE : consume(- ^ -) .

El resultado obtenido al ejecutar el comando anterior es el siguiente:

```
Maude> red in BATTERY-MAUDE : consume(- ^ -) .
reduce in BATTERY-MAUDE : consume(- ^ -) .
rewrites: 1 in 0ms cpu (0ms real) (~ rewrites/second)
result EBattery: - ^ -
```

Devuelve un resultado de tipo EBattery ya que al hacer el consume comprueba que todas las celdas estén todas vacías, y al ser así devuelve el resultado de EBattery

6.- Usando tus propias palabras, ¿qué se obtiene cuando se ejecuta el siguiente comando y por qué?

Maude> red in BATTERY-LEFT-RIGHT : consume-left-right(- ^ o ^ o) .

Al ejecutar el comando anterior obtenemos el siguiente resultado:

```
Maude> red in BATTERY-LEFT-RIGHT : consume-left-right(- ^ o ^ o) .
reduce in BATTERY-LEFT-RIGHT : consume-left-right(- ^ o ^ o) .
rewrites: 0 in 0ms cpu (0ms real) (~ rewrites/second)
result Battery: consume-left-right(- ^ o ^ o)
```

El resultado no variará ya que con el comando red únicamente se evalúan las ecuaciones, no se evalúan las reglas del módulo.

7.- ¿Y con el siguiente comando?

Maude> rew in BATTERY-LEFT-RIGHT : consume-left-right(- ^ o ^ o) .

Al ejecutar el comando anterior obtenemos el siguiente resultado:

```
Maude> rew in BATTERY-LEFT-RIGHT : consume-left-right(- ^ o ^ o) .
rewrite in BATTERY-LEFT-RIGHT : consume-left-right(- ^ o ^ o) .
rewrites: 1 in 0ms cpu (0ms real) (~ rewrites/second)
result Battery: - ^ + ^ o
```

En este caso al utilizar el comando rew si que se tienen en cuenta las reglas declaradas lo que supondrá que se produce un paso de reescritura

8.- ¿Y con el siguiente comando?

Maude> search in BATTERY-LEFT-RIGHT : consume-left-right(- ^ o ^ o) =>! Bt:Battery .

Al ejecutar el comando anterior obtenemos el siguiente resultado:

```
Maude> search in BATTERY-LEFT-RIGHT : consume-left-right(- ^ o ^ o) =>! Bt:Battery .
search in BATTERY-LEFT-RIGHT : consume-left-right(- ^ o ^ o) =>! Bt .

Solution 1 (state 1)
states: 3  rewrites: 2 in 0ms cpu (0ms real) (~ rewrites/second)
Bt --> - ^ + ^ o

Solution 2 (state 2)
states: 3  rewrites: 2 in 0ms cpu (0ms real) (~ rewrites/second)
Bt --> - ^ o ^ +

No more solutions.
states: 3  rewrites: 2 in 0ms cpu (0ms real) (~ rewrites/second)
```

El comando search intenta hacer maching con las reglas definidas, por tanto, se obtienen todos los posibles resultados que se pueden obtener a partir de la expresión *consume(- ^ o ^ o)*.

9.- ¿Y con el siguiente comando?

Maude> search in BATTERY-LEFT-RIGHT : consume-left-right(- ^ o ^ o) => Bt:Battery .*

Al ejecutar el comando anterior obtenemos el siguiente resultado:

```
Maude> search in BATTERY-LEFT-RIGHT : consume-left-right(- ^ o ^ o) =>* Bt:Battery .
search in BATTERY-LEFT-RIGHT : consume-left-right(- ^ o ^ o) =>* Bt .

Solution 1 (state 0)
states: 1  rewrites: 0 in 0ms cpu (0ms real) (~ rewrites/second)
Bt --> consume-left-right(- ^ o ^ o)

Solution 2 (state 1)
states: 2  rewrites: 1 in 0ms cpu (0ms real) (~ rewrites/second)
Bt --> - ^ + ^ o

Solution 3 (state 2)
states: 3  rewrites: 2 in 0ms cpu (0ms real) (~ rewrites/second)
Bt --> - ^ o ^ +

No more solutions.
states: 3  rewrites: 2 in 0ms cpu (0ms real) (~ rewrites/second)
```

Obtenemos un resultado más ya que en este caso, =>* busca todos los estados que hacen matching con la expresion de la derecha