Práctica 1

Jose Collado (jocolsan@inf.upv.es)

Andres Paul (anpaumuo@inf.upv.es)

Preguntas

Batería de celdas mejorada usando subtipos y axiomas en Maude

1. ¿Qué se obtiene tras ejecutar el siguiente comando y cómo se podría arreglar?
   `Maude> red in BATTERY-HASKELL : consume(- ^5 - ^ nil) .`

   Obtenemos esto:
   `result Battery: - ^ - ^ consume(nil)`

   El *consume(nil)* del final no debería aparecer. Para obtener una buena ejecución he añadido la siguente ecuación:
   `eq consume(nil) = nil .`

   Al volver a ejecutar el comando obtengo la reducción correcta:
   `result Battery: - ^ - ^ nil`

Batería de celdas mejorada usando subtipos y axiomas en Maude

2. 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.

   EBattery: - ^ -

   Battery: + ^ -

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

   Para diferenciar celdas llenas y celdas vacias. Además tambien se usa para la relación de subtipado y un conjunto de *ECell* genera una *EBattery.*

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.

   EBattery: - ^ -

   Battery: o ^ o + ^ + - ^ - o ^ + o ^ - + ^ o + ^ - - ^ o - ^ +

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

   `result EBattery: - ^ -`
   Significa que como es una batería formada por *ECells* obtenemos una expresión de tipo *EBattery.*

Batería de celdas mejorada usando indeterminismo en Maude

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) .`

   No obtenemos nada porque estamos usando el comando *reduce* y este solo sirve para ecuaciones. En este modulo solo tenemos reglas.

7. ¿Y con el siguiente comando?
   `rew in BATTERY-LEFT-RIGHT : consume-left-right(- ^ o ^ o) .`

   Con ese comando obtenemos una de las soluciones correctas (recordemos que esta versión puede consumir por la derecha o por la izquierda):
   `result Battery: - ^ + ^ o`

8. ¿Y con el siguiente comando?
   `Maude> search in BATTERY-LEFT-RIGHT : consume-left-right(- ^ o ^ o) =>! Bt:Battery .`

   Con este comando *search* obtenemos todas las soluciones posibles hasta que no sea posible dar más pasos de ejecución. Por lo tanto encontramos las 2 soluciones posibles que hay al dar 1 paso de ejecución, que es el maximo para llegar a ese estado.

9. ¿Y con el siguiente comando?
   `search in BATTERY-LEFT-RIGHT : consume-left-right(- ^ o ^ o) =>* Bt:Battery .`

   Con este comando obtenemos todas las soluciones posibles utilizando tantos pasos de ejecución como sea necesario. Por lo tanto encontramos una solución mas que con el comando anterior que corresponde a la solucion de no dar ningun paso de ejecución (`Bt --> consume-left-right(- ^ o ^ o)`)

Ejercicio libre sobre la batería de celdas

```
mod BATTERY-ANY-CELL is
protecting BATTERY-MAUDE .
op consume-any-cell : Battery -> Battery .

var FirstBt : Battery .
var SecondBt : Battery .
var EBt : EBattery .


rl consume-any-cell(FirstBt ^ o ^ SecondBt) => consume-any-cell(FirstBt ^ + ^
SecondBt) .
rl consume-any-cell(FirstBt ^ + ^ SecondBt) => consume-any-cell(FirstBt ^ - ^
SecondBt) .
eq consume-any-cell(EBt) = EBt .
endm
```