Aritmética Entera

Codificar enteros tomando como base una codificación de los naturales es relativamente sencillo. Para ello, dado un par (m, n), con $m, n \in \mathbb{N}$, consideraremos que representa al entero $m - n$[a].
Al igual que ocurre con otras codificaciones de los enteros, los pares no representan a los enteros de manera única, p.e. (n, m) codifica el mismo entero que $(n+k, m+k)$.

---

[a] En la asignatura de Estructuras de Datos se empleó una codificación similar, en vez de escribir un entero como un par (n, m) se le representaba mediante una serie de llamadas a *suc* y *pred* (n a *suc* y m a *pred*)

Aritmética entera (y 2)

En el fichero *enteros.rkt* del servidor ftp de la asignatura se puede ver una codificación de los enteros usando λ-Cálculo e implementada empleando `Scheme`. Dicha codificación incluye las siguientes operaciones:

- Suma y resta
- Multiplicación
- División euclídea
- Cálculo del máximo común divisor
- Relaciones de igualdad y de orden
- Reducción a representante canónico[a]

[a] Tomaremos $(n, 0)$ ó $(0, n)$ como representantes canónicos.

Objetivos de la práctica

(1) El conjunto de los números módulo p, $\mathbb{Z}_p$, es el conjunto cociente de $\mathbb{Z}$ mediante la relación de equivalencia

$$n\ R m \text{ si y sólo si } n - m \text{ es múltiplo de p}$$

Teniendo en cuenta lo anterior y la codificación, relaciones de orden y operaciones de enteros definidas en el fichero *enteros* mencionado, se pide codificar en $\lambda-$cálculo $\mathbb{Z}_p$. Dicha codificación debe incluir las siguientes operaciones:

(a) Reducción a representante canónico.
(b) Aritmética: suma, producto.
(c) Decisión sobre la inversibilidad y cálculo del inverso en el caso de que exista.

Objetivos de la práctica (y 2)

(2) Codificada la aritmética modular, se pide codificar las matrices $2x2$ en $\mathbb{Z}_p$ con p primo. Esta codificación debe incluir las siguientes operaciones:

(a) Suma y producto.
(b) Determinante.
(c) Decisión sobre inversibilidad y cálculo de inversa y del rango.
(d) Cálculo de potencias (naturales) de matrices. Este cálculo se tiene que hacer usando el algoritmo binario para el cálculo de potencias, también conocido como exponenciación binaria.

Algo de `Scheme`

Las necesidades de `Scheme` para esta práctica quedan cubiertas, salvo en el caso de la recursión, con la siguiente observación: el término del λ-Cálculo $\lambda x.M$ se codifica en `Scheme` mediante `(lambda (x) M)`. En el caso de que necesitáramos dar un nombre a un término para su posterior reutilización, la forma de hacerlo sería la siguiente `(define termino (lambda (x) M))`.

Por ejemplo, siguiendo lo visto en clase uno puede definir:

```
(define true (lambda (x y) x))
(define false (lambda (x y) y))
(define if (lambda (p x y) (p x y)))
. . .
```

Algo de Scheme. Recursión en λ-cálculo

Dos observaciones sobre cómo simular la recursividad en el λ-cálculo con Scheme:

- El combinador de punto fijo **Y** ha de definirse aplicando una η-expansión (líneas 23 a 29 del fichero *enteros.rkt*).
- Aún definiendo así **Y**, la recursividad no funciona tal cual se ha visto en clase. Ejemplos de cómo se puede simular la recursión se encuentran en el fichero *enteros.rkt*. Por ejemplo, en la definición del resto de la división euclídea (líneas 135-153)[a].

NOTA FINAL: Los únicos λ-términos no currificados que se admitirán en la práctica son los booleanos que aparecen al principio del fichero *enteros.rkt*.

---

[a] Esta simulación de la recursividad se ha extraído de

`http://www.shlomifish.org/lecture/Lambda-Calculus/slides/lc_church_div.scm.html`