Universidad Nacional Autónoma de México
Facultad de Ciencias
Lenguajes de Programación

Práctica 2

Karla Ramírez Pulido karla@ciencias.unam.mx
J. Ricardo Rodríguez Abreu ricardo_rodab@ciencias.unam.mx
Manuel Soto Romero manu@ciencias.unam.mx

Fecha de inicio: 23 de febrero de 2018
Fecha de término: 1 de marzo de 2018
Semestre 2018-2

Objetivo

Conocer y utilizar los componentes de la variante `plai` para la creación y uso de Tipos de Datos Abstractos (TDA) mediante las primitivas `define-type` para crearlos y `type-case` o `match` para usarlos mediante la técnica de *apareamiento de patrones*[1].

Antecedentes

En las sesiones previas de laboratorio se revisó cómo crear y usar Tipos de Datos Abstractos y se elaboraron actividades disponibles en el repositorio del curso. En caso de no haber elaborado dichas actividades durante las sesiones de laboratorio, se recomienda revisarlas junto con el Anexo 2 de las notas del curso.

Repositorio

El material necesario para completar esta práctica se encuentra en el repositorio de *GitHub Classroom* del curso: `https://classroom.github.com/g/oxGkdx4j`.

Desarrollo de la práctica

En equipos de **tres integrantes** completar las funciones faltantes del archivo `practica2.rkt` hasta que pasen todas las pruebas unitarias incluidas en el archivo `pruebas_practica2.rkt`[2] .

[1] Pattern Matching.
[2] Para tener derecho a calificación, el archivo debe ejecutarse sin errores.

Ejercicio 2.1 (5 pts.)

1. Completar la definición del tipo de dato abstracto `Figura` para construir figuras geométricas. El tipo de dato abstracto debe contener:

   - Un constructor `(triangulo a b c)` donde `a`, `b` y `c` son número reales y representan los lados del triángulo.
   - Un constructor `(cuadrado a)` donde `a` es un número real y representa el lado del cuadrado.
   - Un constructor `(rectangulo a b)` donde `a` y `b` son números reales y representan la altura y base del rectángulo.
   - Un constructor `(rombo a D d)` donde `a`, `D` y `d` son números reales y representan el lado, diagonal mayor y diagonal menor del rombo respectivamente.
   - Un constructor `(paralelogramo a b h)` donde `a`, `b` y `h` son números reales y representan los lados y altura del paralelogramo respectivamente.
   - Un constructor `(circulo D)` donde D es un número real y representa el diámetro del círculo.
   - Un constructor `(elipse a b)` donde `a` y `b` son números reales y representan el semieje mayor y el semieje menor de la elipse respectivamente.

2. Una vez definido el tipo de dato, se deben debe completar el cuerpo de las siguientes funciones:

   a) La función `(perimetro fig)` que dada una figura regrese el perímetro de ésta.

   ```
   ;; perimetro: Figura -> number
   (define (perimetro fig) ...)
   ```

   ```
   > (define figura (triangulo 17 29 20)
   > (perimetro figura)
   66
   ```

   b) La función `(area fig)` que dada una figura calcule el área de ésta.

   ```
   ;; area: Figura -> number
   (define (area fig) ...)
   ```

   ```
   > (define figura (triangulo 17 29 20)
   > (area figura)
   165.698
   ```

Ejercicio 2.2 (5 pts.) Dada la siguiente definición de nodos simples:

```
(define-type Nodo
    [vacio]
    [nodo (elemento any?) (siguiente Nodo?)])
```

1. Completar la definición de los siguientes tipos de datos abstractos para trabajar con las estructuras de datos `Pila` y `Cola`:

   El tipo `Pila` debe contener:

   - Un constructor `(pila n)` para representar una `Pila`, tal que `n` es una sucesión de nodos.
   - Un constructor `(mete-pila e p)` el cual representa la operación que agrega un elemento `e` en la `Pila` `p`. El elemento se agrega al inicio de la sucesión de nodos.
   - Un constructor `(saca-pila p)` para representar la operación de eliminar un elemento de la `Pila` `p`. El elemento a eliminar es el primero en la sucesión de nodos.
   - Un constructor `(mira-pila p)` el cual representa la operación que muestra el elemento encima de la `Pila` `p`. El elemento a mostrar es el primero en la sucesión de nodos.

   El tipo `Cola` debe contener:

   - Un constructor `(cola n)` para representar una `Cola`, tal que `n` es una sucesión de nodos.
   - Un constructor `(mete-cola e c)` para representar la operación de agregar un elemento `e` en la `Cola` `c`. El elemento se agrega al inicio de la sucesión de nodos.
   - Un constructor `(saca-cola c)` el cual representa la operación de eliminación de un elemento de la `Cola` `c`. El elemento a eliminar es el último en la sucesión de nodos.
   - Un constructor `(mira-cola c)` para representar la operación que muestra el primer elemento de la `Cola` `c`. El elemento a mostrar es el último en la sucesión de nodos.

2. Una vez definidos los tipos de datos, se debe de completar el cuerpo de la función `(calc-pila p)` que evalúa expresiones del tipo `Pila` y una función `(calc-cola c)` que evalúa expresiones del tipo `Cola`.

```
;; calc-pila: Pila -> any
(define (calc-pila p) ...)
```

```
> (define p (pila (nodo 1 (nodo 2 (nodo 3 (vacio))))))
> (calc-pila p)
(nodo 1 (nodo 2 (nodo 3 (vacio))))
> (calc-pila (mete-pila 4 p))
(nodo 4 (nodo 1 (nodo 2 (nodo 3 (vacio)))))
> (calc-pila (saca-pila p))
(nodo 2 (nodo 3 (vacio)))
> (calc-pila (mira-pila p))
1
```

```
;; calc-cola: Cola -> any
(define (calc-cola c) ...)
```

```
> (define c (cola (nodo 1 (nodo 2 (nodo 3 (vacio))))))
> (calc-cola c)
(nodo 1 (nodo 2 (nodo 3 (vacio))))
> (calc-cola (mete-cola 4 c))
(nodo 4 (nodo 1 (nodo 2 (nodo 3 (vacio)))))
> (calc-cola (saca-cola c))
(nodo 1 (nodo 2 (vacio)))
> (calc-cola (mira-cola c))
3
```

Referencias

Algunas referencias de consulta:

[1] Karla Ramírez, Manuel Soto, *Notas de laboratorio del curso de Lenguajes de Programación*, Semestre 2018-2, Facultad de Ciencias, UNAM. Disponibles en: [http://lenguajesfc.com/notas.html].

[2] Rodrigo Ruiz Murgía, *Manual de prácticas para la asignatura de Lenguajes de Programación*, Reporte de actividad docente, Facultad de Ciencias, 2016.

[3] Eric Tánter, *PREPLAI: Scheme y Programación Funcional*, Primera edición, 2014. Disponible en: [http://users.dcc.uchile.cl/~etanter/preplai/] (Consultado el 4 de agosto de 2017).

[4] Matthias Felleisen, Robert Findler, Matthew Flatt, Shriram Krishnamurthi, *How to Design Programs*, Segunda Edición, The Mit Press, 2017. Disponible en: [http://www.ccs.neu.edu/home/matthias/HtDP2e/] (Consultado el 4 de agosto de 2017).

[5] Matthias Felleisen, David Van Horn, Conrad Barski, *Realm Of Racket*, Primera edición, No Starch Press, 2013.