Tipos de datos en NQTHM

Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Tipos de datos en NQTHM.

- **Tipos de datos primitivos:**
  - **Números naturales:** `NUMBERP`.
  - **Pares ordenados:** `LISTP`.
  - **Literales:** `LITATOM`.
  - **Números negativos:** `NEGATIVEP`.
- **Estos cuatro tipos se definen inicialmente al cargar NQTHM.**
- **El usuario puede definir nuevos tipos con la función** `ADD-SHELL`.
- **En cada momento, cualquier objeto que NQTHM maneje pertenece exactamente a uno de los tipos de datos definidos.**
- **A cada tipo de datos se le asocian las siguientes funciones:**
  - **Función constructora.**
  - **Función(es) accesor(as).**
  - **Función reconocedora.**
  - **Valores por defecto.**
  - **Objeto base (opcional).**

Tipos de datos definidos con ADD-SHELL

- **La orden ADD-SHELL:**

```
(ADD-SHELL constructor base reconocedor
           ((selector1 rt1 valor-defecto1))
            ....
            (selectorN rtN valor-defectoN))
```

- Cada rtI es una restricción de tipo, de la forma (ONE-OF rec1 ... recM) ó (NONE-OF rec1 ... recM) donde cada recI es un reconocedor de un T.D. (posiblemente el mismo que se está definiendo).

- Cada valor-defectoI es el símbolo del objeto base de un T.D. ya definido.

- El resto de símbolos, deben ser nuevos.

- Si no hay elemento base, usar NIL.

Los números naturales.

- **Definición:**

```
(add-shell add1 zero numberp
   ((sub1 (one-of numberp) zero)))

;;; Tipo de datos primitivo.

;;; ADD1   : función constructora.
;;; SUB1   : función accesora.
;;; (ZERO) : objeto base.
;;; NUMBERP: reconocedor del tipo de dato.
;;; (ONE-OF NUMBERP) : restricción de tipo.
```

- **Ejemplos:**

```
>(r-loop)
*(zero)
 0  ;;; abreviatura
*(add1 (add1 (add1 (zero))))
 3  ;;; abreviatura
*(sub1 (add1 (add1 (zero))))
 1
*(sub1 'hola)
 0
*(add1 'hola)
 1
*(numberp (add1 'hola))
 T
*(numberp 3)
 T
```

Los pares ordenados.

- **Definición:**

```
(add-shell cons nil listp
   ((car (none-of) zero) (cdr (none-of) zero)))
;;; Tipo de dato primitivo.

;;; CONS        : función constructora.
;;; CAR, CDR    : funciones accesoras.
;;; No existe objeto base.
;;; LISTP       : reconocedor del tipo de dato.
;;; (NONE-OF)   : restricción (sin restricción).
;;; ZERO        : valor por defecto.
```

- **Ejemplos:**

```
>(r-loop)
*(cons 3 4)
 '(3 . 4)
*(cons 3 nil)
 '(3)
*(car (cons 3 4))
 3
*(cdr (cons 3 4))
 4
*(listp (cons 3 4))
 T
*(listp 'hola)
 F
*(car 'hola)
 0
*(cdr 'hola)
 0
```

Observaciones (I).

- **Los valores por defecto permiten que las funciones constructoras y accesoras estén definidas para cualquier elemento.**
- **Respecto de los accesores:**

```
Cuando un accesor actúa sobre un elemento que no
es del tipo esperado, devuelve el correspondiente
valor por defecto:

Ejemplo:
========
El tipo esperado del argumento de la función CAR es
LISTP. Si CAR actúa sobre algo que no es LISTP,
devuelve el valor por defecto definido en el
ADD-SHELL (es decir, (ZERO) ó 0). P.ej. (CAR 3) = 0
```

Observaciones (II).

- **Respecto de los constructores:**

```
Cuando un constructor actúa sobre algún elemento
que no verifica la restricción de tipo, actúa como
si recibiera en su lugar el valor por defecto.

Ejemplo:
========
ADD1 debe recibir elementos que sean números (NUMBERP).
Si se aplica a un argumento que no es NUMBERP, este
argumento se transforma al valor por defecto
(0 ó (ZERO)). P.ej., (ADD1 'hola) es (ADD1 0), es decir, 1.
```

- **Otros tipos de datos ya definidos:**
  - `NEGATIVEP`: **los enteros negativos.**
  - `LITATOM`: **los símbolos.**
- **Atención:** `NIL` **es** `LITATOM`**.**
- **En NQTHM no se maneja el concepto de lista vacía.**

Un ejemplo definido por el usuario.

- **Tipo de datos PILAP (pila de números):**

```
(add-shell apila vacia pilap
   ((cima (one-of numberp) zero)
    (desapila (one-of pilap) vacia)))

;;; APILA           : función constructora.
;;; CIMA, DESAPILA  : funciones accesoras.
;;; (VACIA)         : objeto base.
;;; LISTP           : reconocedor del tipo de dato.
;;; (ONE-OF NUMBERP) : restricción para APILA, primer arg.
;;; (ONE-OF PILAP)   : restricción para APILA, segundo arg.
;;; PILAP           : función reconocedora.
```

- **Ejemplos:**

```
*(setq p (apila 3 (vacia)))
 (APILA 3 (VACIA))
*(setq p (apila 2 (apila 'hola p)))
 (APILA 2 (APILA 0 (APILA 3 (VACIA))))
*(cima p)
 2
*(desapila p)
 (APILA 0 (APILA 3 (VACIA)))
*(desapila 3)
 (VACIA)
*(cima (apila 'hola (desapila 'hola)))
 0
*(apila 2 (apila 3 (desapila (apila 4 (apila 5 (vacia))))))
 (APILA 2 (APILA 3 (APILA 5 (VACIA))))
```

Axiomas añadidos por ADD-SHELL.

- **Por cada ADD-SHELL, el sistema genera una serie de axiomas (reglas) que definen el comportamiento de accesores, constructores y reconocedores.**
- **Ejemplo: algunos axiomas relativos a la "shell" NUMBERP:**

```
1) (OR (EQUAL (NUMBERP X) T) (EQUAL (NUMBERP X) F))
2) (NUMBERP (ADD1 X))
3) (NUMBERP (ZEROP))
4) (NOT (EQUAL (ADD1 X) (ZERO)))
5) (IMPLIES (AND (NUMBERP X) (NOT (EQUAL X (ZERO))))
         (EQUAL (ADD1 (SUB1 X)) X))
6) (IMPLIES (NUMBERP X1)
            (EQUAL (SUB1 (ADD1 X1)) X1))
....
```

- **Además, extiende la definición de la función COUNT, sobre los datos del tipo definido.**

La función COUNT.

- **Definición de COUNT:**

```
Dado un objeto A, la función (COUNT A) devuelve el
número de constructores que constituyen A.
Ejemplo:
========
(COUNT 4) = (COUNT (ADD1 (ADD1 (ADD1 (ADD1 (ZERO)))))) = 4
(COUNT '(1 . 2)) =
    (COUNT (CONS (ADD1 (ZERO)) (ADD1 (ADD1 (ZERO))))) = 4
(COUNT (APILA 0 (APILA 1 (VACIA)))) = 3
```

- **COUNT es la única función que se define incrementalmente, con cada ADD-SHELL**
- **Importancia de COUNT: usada como medida para admitir funciones recursivas.**

Observaciones finales

- **Los constructores, selectores y reconocedores de un T.D. no necesitan ni pueden ser definidos con `DEFN`.**
  - **Los constructores "fabrican" un nuevo objeto a partir de sus n argumentos.**
  - **Los selectores, cuando se aplican a un elemento ya "fabricado", obtienen uno de los n argumentos con los que fue construido.**
  - **Los reconocedores sirven para "detectar" elementos de un tipo de dato determinado (base o construidos)**
- **Aunque no se definen con `DEFN`, el evento `ADD-SHELL` introduce en el sistema los axiomas necesarios que permiten razonar sobre objetos pertenecientes al tipo.**