Programación funcional recursiva con $\mathcal{SIPRES}$

El lenguaje de programación recursiva $\mathcal{SIPRES}$

Authors: Edwin Camilo Cubides Garzón, Ph.D.(c)
eccubidesg@unal.edu.co

Jonatan Gómez Perdomo, Ph.D.
jgomezpe@unal.edu.co

Grupo de investigación en vida artificial – Research Group on Artificial Life – (Alife)
Departamento de Ingeniería de Sistemas e Industrial
Facultad de Ingeniería
Universidad Nacional de Colombia

2do semestre de 2019

Outline

Outline

Constantes

`0`: representa el valor 0, el cardinal del conjunto vacío $|\varnothing| = 0$.

`true`: representa el valor lógico verdadero.

`false`: representa el valor lógico falso.

`[]`: representa la lista vacía, una lista sin elementos.

`undef`: representa el símbolo indefinido, se utiliza para definir que una expresión no puede computarse.

Funciones primitivas

`s(•)`: permite calcular la función sucesor en los números naturales ($s : \mathbb{N} \mapsto \mathbb{N}^+ : n \to n + 1$).

`equal(•,•)`: define el único predicado permitido en $\mathcal{SIPRES}$, y que permite definir que dos expresión son equivalentes.

`[•|•]`: es el constructor básico de las listas (la estructura de datos básica en $\mathcal{SIPRES}$), el primer parámetro es un valor (cabeza) y el segundo es una lista (cola) ésta puede ser vacía o no.

Números naturales (notación de Peano)

Números: para el cero (0) se utiliza la constante "0". Para un $n \in \mathbb{N}$ se construye componiendo la función primitiva "s" n veces y evaluando la más interna en 0, es decir,

$$n := \underbrace{\texttt{s(s(}\cdots\texttt{(s(}0\texttt{))}\cdots\texttt{))}}_{n \text{ veces}}$$

Ejemplo

- $0 := 0$
- $1 := \texttt{s(0)}$
- $2 := \texttt{s(s(0))}$
- $3 := \texttt{s(s(s(0)))}$
- $4 := \texttt{s(s(s(s(0))))}$
- $5 := \texttt{s(s(s(s(s(0)))))}$
- $10 := \texttt{s(s(s(s(s(s(s(s(s(s(0)))))))))}$

Variables

Variables: son secuencias de una o más letras mayúsculas del alfabeto inglés

$$\{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}, \texttt{E}, \texttt{F}, \texttt{G}, \texttt{H}, \texttt{I}, \texttt{J}, \texttt{K}, \texttt{L}, \texttt{M}, \texttt{N}, \texttt{O}, \texttt{P}, \texttt{Q}, \texttt{R}, \texttt{S}, \texttt{T}, \texttt{U}, \texttt{V}, \texttt{W}, \texttt{X}, \texttt{Y}, \texttt{Z}\}^{+}$$

Ejemplo

- `A`
- `B`
- `N`
- `M`
- `X`
- `Y`
- `Z`
- `NUM`
- `VAR`
- `AUX`
- `HELLO`

Funciones

Funciones: son expresiones de la forma

$$f(x_1, x_2, \ldots, x_n)$$

donde el nombre de la función "f" es una secuencia de una o más letras minúsculas del alfabeto inglés, los dígitos, los símbolos _, $. No puede iniciar con un dígito, ni puede ser alguna de las palabras reservadas: "`true`", "`false`", "`undef`", "`s`", "`equal`".

Ejemplo

- `f`
- `g`
- `$h$`
- `and`
- `or`
- `sum`
- `fact`
- `length`
- `quick_sort`
- `mod2`
- `_q0`

Funciones (conti.)

Para los parámetros $x_1, x_2, \ldots, x_n$ se tiene que $n \geq 1$ y cada x_i es o una constante o un número o una variable o una lista o una función.

Ejemplo

- `f(X)`
- `g(false)`
- `h(f(A),g(B))`
- `and(X,X)`
- `or(true,X)`
- `sum(M,s(N))`
- `fact(s(NUM))`
- `length([s(0)|[0|[0|[]]]])`
- `quick_sort([s(0)|[0|[]]])`
- `append([0|[s(0)|[]]],[s(s(0))|[s(s(s(0)))|[]]])`

Ecuaciones

Ecuaciones: son expresiones de la forma

$$\texttt{equal}(l,r)$$

donde "l" es una función y "r" es o una constante o un número o una variable o una lista o una función. A "l" se le llama la parte izquierda de la ecuación y a "r" se le llama la parte derecha de la ecuación.

Ejemplo

- `equal(and(A,A),A)`
- `equal(or(true,X),true)`
- `equal(div(NUM,0),undef)`
- `equal(sum(M,s(N)),sum(s(M),N))`
- `equal(fact(s(NUM)),prod(s(NUM),fact(NUM)))`
- `equal(length([s(0)|[0|[0|[]]]]),s(s(s(0))))`

Programas

Programas: son secuencias de una o más ecuaciones separadas por los símbolos ";" o "↵". El símbolo ↵ se utiliza para representar el carácter no imprimible de salto de línea que se obtiene al oprimir la tecla *enter*.

Para cada ecuación se debe cumplir que:

- las variables de la parte derecha de la ecuación deben estar en la parte izquierda.

Ejemplo

```
equal(succ(N),s(N))
```

Ejemplo

```
equal(pred(0),undef); equal(pred(s(N)),N)
```

Programas (conti.)

Ejemplo

```
equal(and(true,true),true)↵
equal(and(true,false),false)↵
equal(and(false,true),false)↵
equal(and(false,false),false)
```

Ejemplo

```
equal(even(0),true);↵
equal(even(s(0)),false);↵
equal(even(s(s(NUM))),even(NUM));
```

Objetivos

Objetivos: son secuencias de una o más expresiones que no sean ecuaciones ni que contengan variables.

Ejemplo

- `0`
- `s(0)`
- `not(true)`
- `xor(true,false)`
- `sum(s(0),s(s(s(0))))`
- `fact(s(s(s(0))))`
- `true`
- `false`
- `undef`
- `[]`
- `length([0|[s(0)|[]]])`
- `first([true|[false|[]]])`
- `app([],[s(0),[s(0)|[]]])`
- `quick_sort([s(0)|[0|[]]])`

Ejercicios sobre programas

Ejercicio

Ejercicio 1 Desde el siguiente enlace, descargar el IDE interprete del lenguaje $\mathcal{SIPRES}$

http://www.alife.unal.edu.co/~eccubidesg/computacion-evolutiva/SIPRES.jar

escribir el programa

```
equal(succ(N),s(N))
```

y evaluar en los valores $0, 1, 2, 5$ y 10, como se muestra en la figura en la siguiente diapositiva. ¿Cuál es la función matemática que se esta programando? exprese la función en notación dominio, rango é imagen.

Ejercicios sobre programas (conti.)

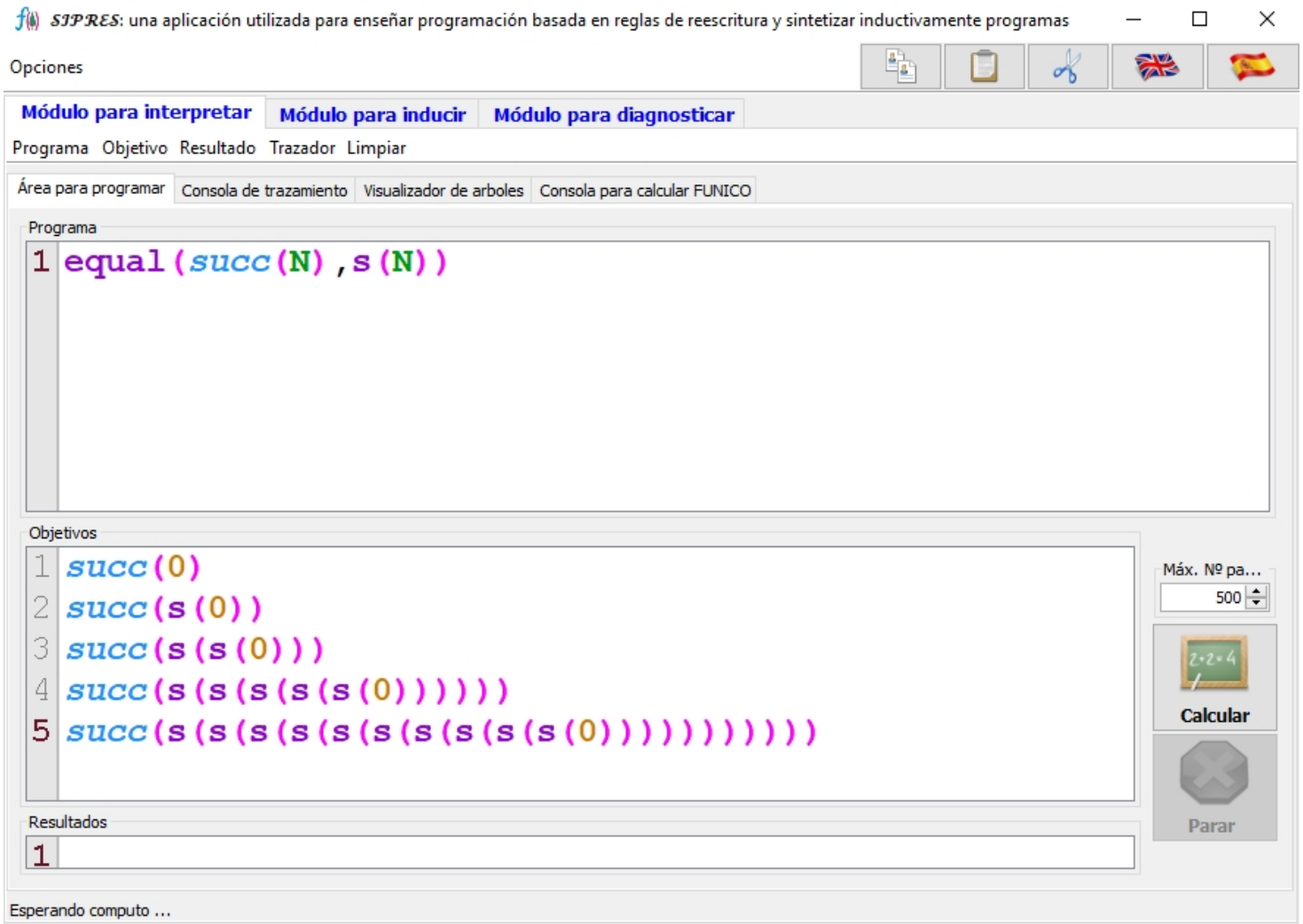


Figure: Uso del sistema $\mathcal{SIPRES}$ con el programa succ(•).

Ejercicios sobre programas

Ejercicio

Ejercicio 2 Descargar el IDE del interprete del lenguaje $\mathcal{SIPRES}$, escribir el programa

```
equal(f(0),0);equal(f(s(0)),s(0))↵
equal(f(s(s(0))),s(s(0)))↵
equal(f(s(s(s(NUM)))),f(NUM))
```

y evaluar en los valores $0, 1, 2, 3, 4, 5, 6$ y 7. ¿Cuál es la función matemática que se está programando?.

Outline

Azúcar sintáctico para números naturales

Para facilitar la programación, en la escritura de ecuaciones u objetivos se puede utilizar la notación tradicional de secuencias de uno o más dígitos; como en el lenguaje $\mathcal{SIPRES}$ los números naturales se representan internamente con la notación de Peano, entonces durante la interpretación se hace la traducción a notación de Peano.

Ejemplo

- $0 := \texttt{0} \equiv \texttt{0}$
- $1 := \texttt{s(0)} \equiv \texttt{1}$
- $2 := \texttt{s(s(0))} \equiv \texttt{2}$
- $3 := \texttt{s(s(s(0)))} \equiv \texttt{3}$
- $4 := \texttt{s(s(s(s(0))))} \equiv \texttt{4}$
- $5 := \texttt{s(s(s(s(s(0)))))} \equiv \texttt{5}$
- $10 := \texttt{s(s(s(s(s(s(s(s(s(s(0)))))))))} \equiv \texttt{10}$

Azúcar sintáctico para listas

Como las listas son estructuras que se utilizan con mucha frecuencia; entonces, para facilitar la representación, se tiene que la lista vacía se representa con "[]", y una lista con $n \geq 1$ elementos $x_1, x_2, \ldots, x_n$, se representa en $\mathcal{SIPRES}$ como

$$[x_1 \,|\, [x_2 \,|\, \cdots \,|\, [x_n \,|\, []] \cdots]]$$

la cual se puede escribir equivalentemente como $[x_1, x_2, \ldots, x_n]$

Ejemplo

- `[]` $\equiv$ `[]`
- `[0|[]]` $\equiv$ `[0]`
- `[0|[1|[2|[]]]]` $\equiv$ `[0,1,2]`
- `[true|[false|[]]]` $\equiv$ `[true,false]`
- `[[1|[0|[]]]|[[1|[0|[]]]|[]]]` $\equiv$ `[[1,0],[0,1]]`

Azúcar sintáctico para ecuaciones

Como las ecuaciones en $\mathcal{SIPRES}$ se expresan en la forma

$$\texttt{equal}(l,r)$$

estas se pueden escribir equivalentemente en la forma

$$l \texttt{ = } r$$

Ejemplo

- `equal(and(A,A),A)` $\equiv$ `and(A,A) = A`
- `equal(or(true,X),true)` $\equiv$ `or(true,X) = true`
- `equal(div(NUM,0),undef)` $\equiv$ `div(NUM,0) = undef`
- `equal(sum(M,s(N)),sum(s(M),N))` $\equiv$ `sum(M,s(N)) = sum(s(M),N)`
- `equal(fact(s(N)),prod(s(N),fact(N)))` $\equiv$ `fact(s(N)) = prod(s(N),fact(N))`

Ejercicios sobre programas

Ejercicio

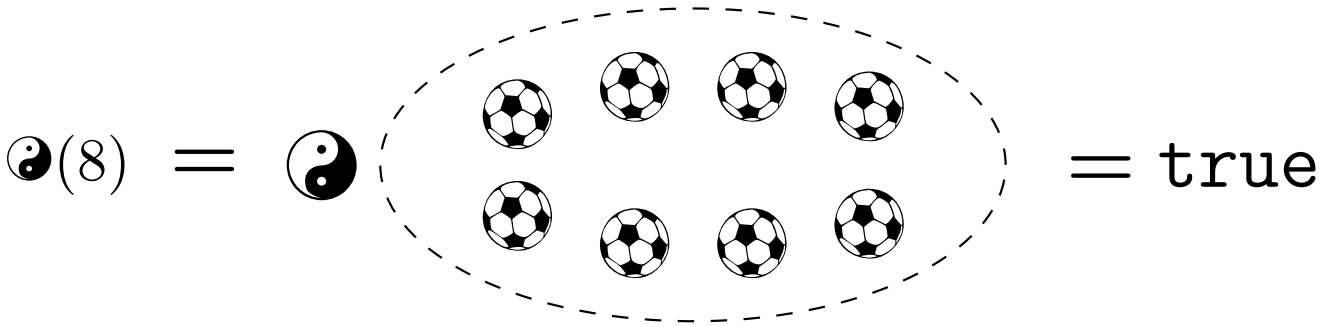
Ejercicio Desarrolle de nuevo los ejercicios 1 y 2, pero ahora utilizando el azúcar sintáctico para números naturales y para ecuaciones ¿Varían las respuestas que se obtuvieron al resolver los ejercicios 1 y 2, previamente?.

Outline

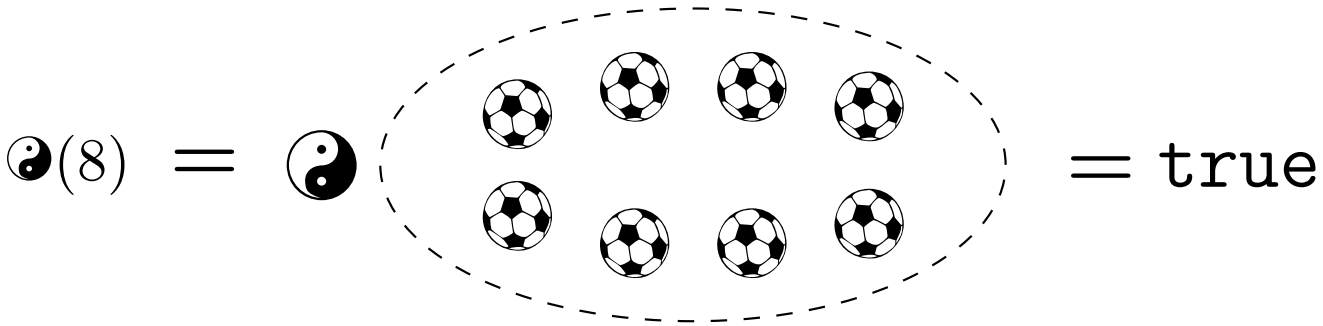
Ejemplo: programa recursivo yinyang

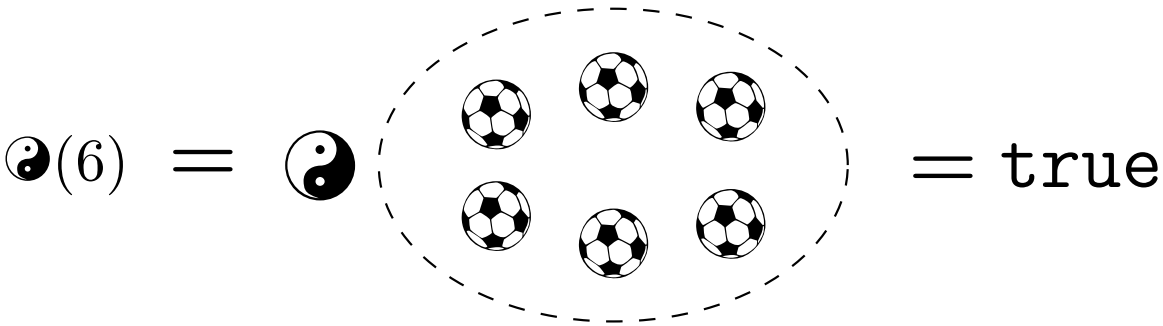
```
yinyang(8) = true
yinyang(6) = true
yinyang(4) = true
yinyang(2) = true
yinyang(0) = true

yinyang(7) = false
yinyang(5) = false
yinyang(3) = false
yinyang(1) = false
```

Ejemplo: programa recursivo yinyang



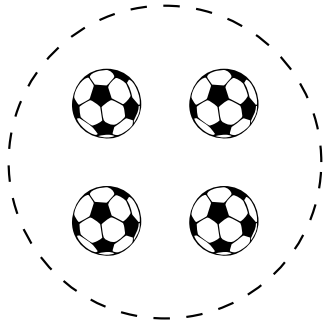
Ejemplo: programa recursivo yinyang





Ejemplo: programa recursivo yinyang

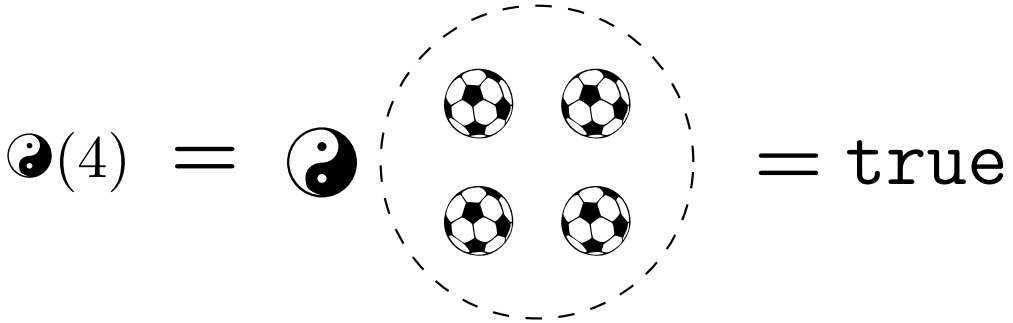
$$☯(4) = ☯ \quad = \texttt{true}$$

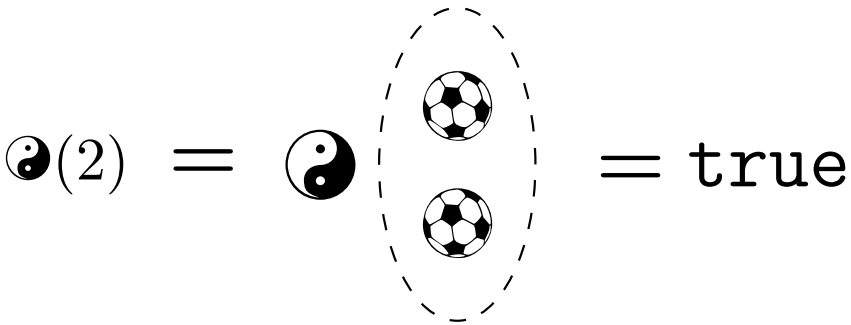


$$☯(2) = ☯ \quad = \texttt{true}$$

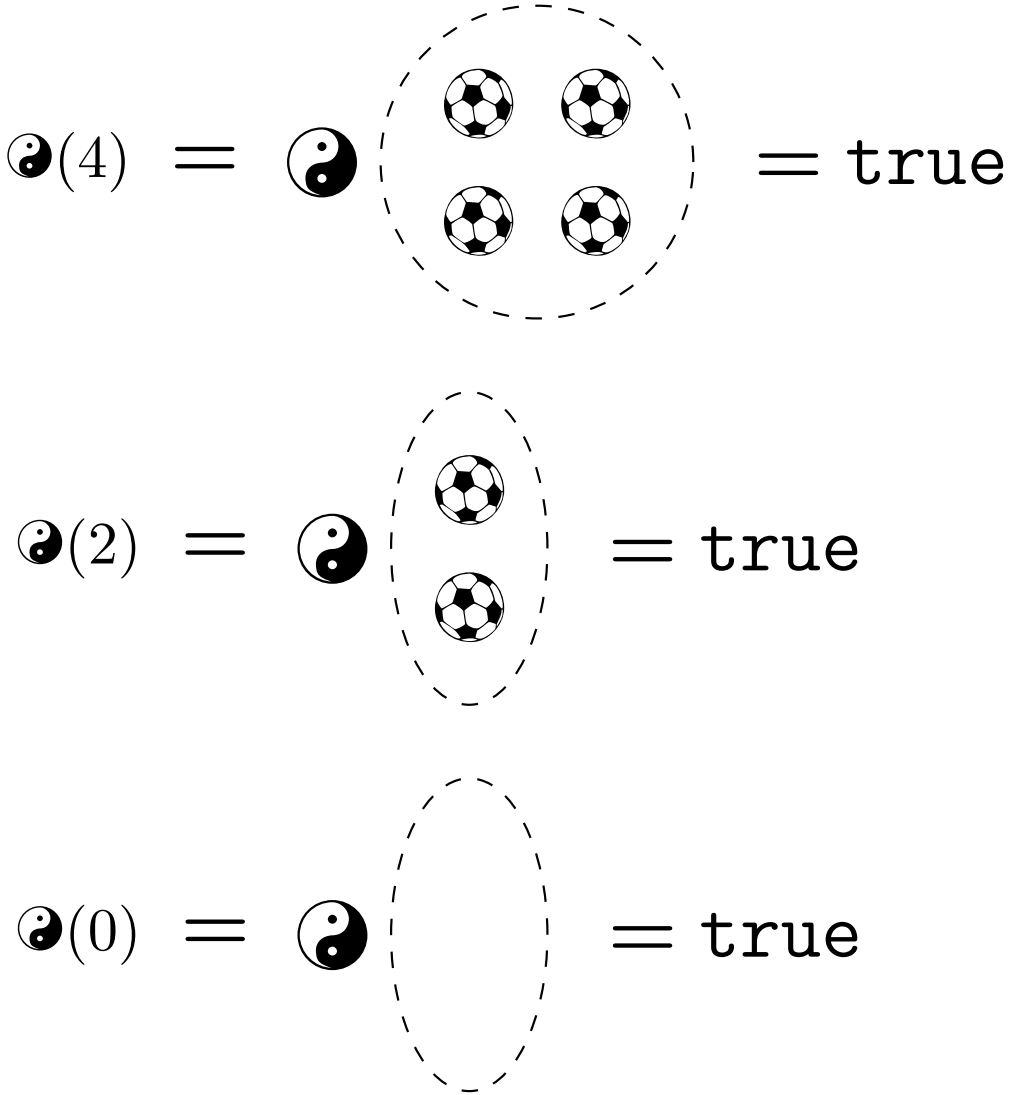
$$☯(0) = ☯ \quad = \texttt{true}$$

Ejemplo: programa recursivo yinyang

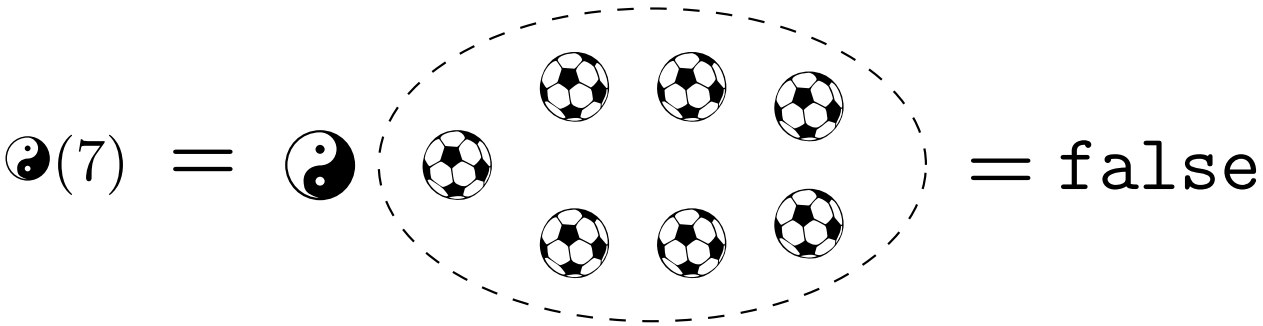




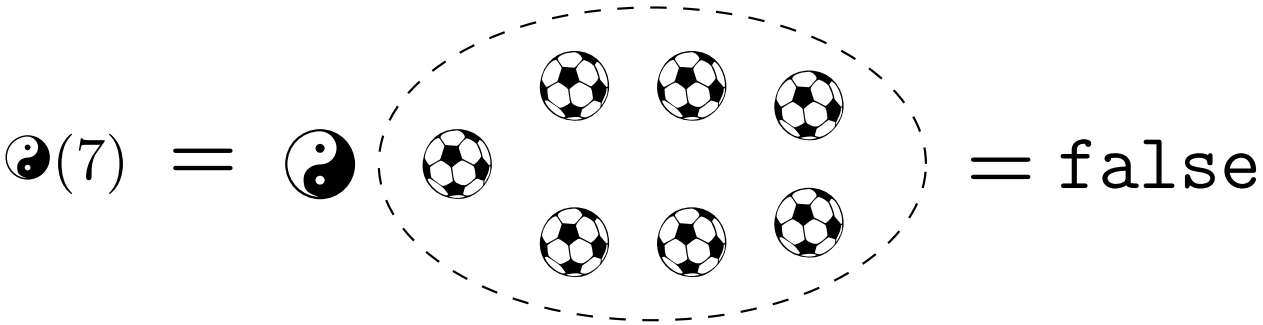
Ejemplo: programa recursivo yinyang

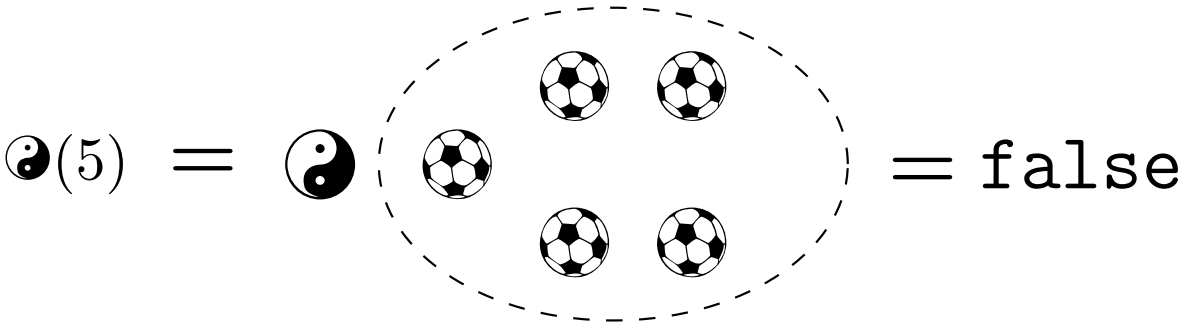


Ejemplo: programa recursivo yinyang

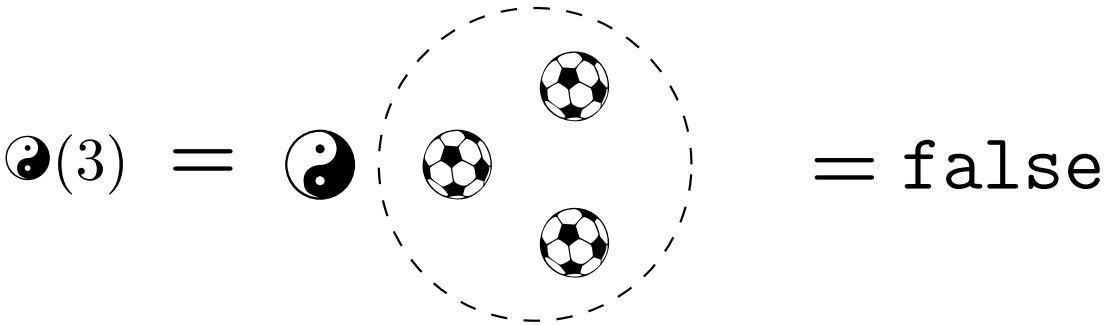


Ejemplo: programa recursivo yinyang



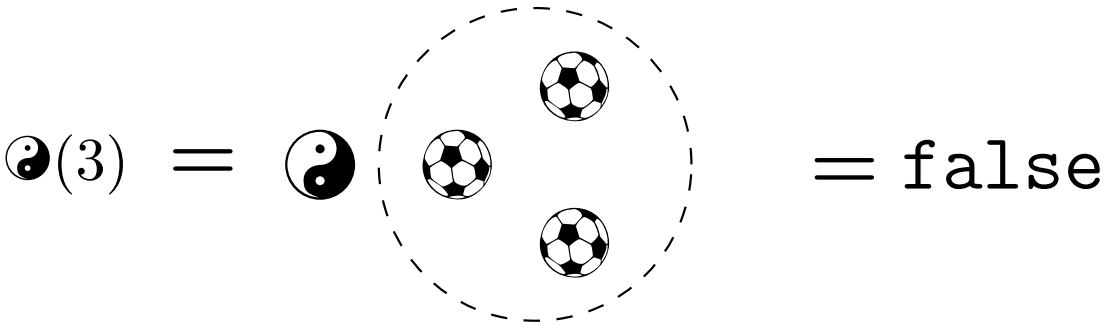


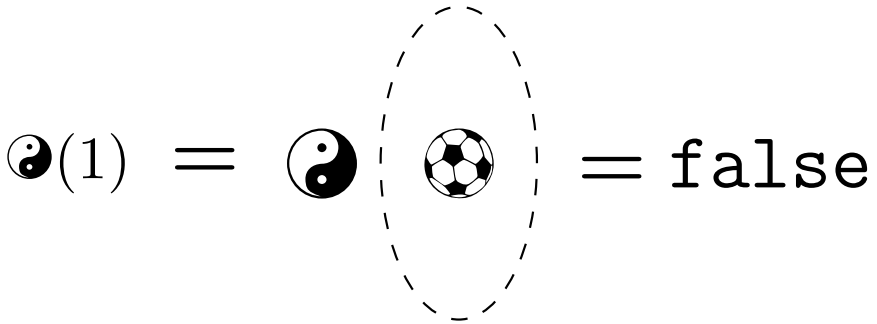
Ejemplo: programa recursivo yinyang




```
yinyang(0) = true
yinyang(1) = false
yinyang(s(s(N))) = yinyang(N)
```

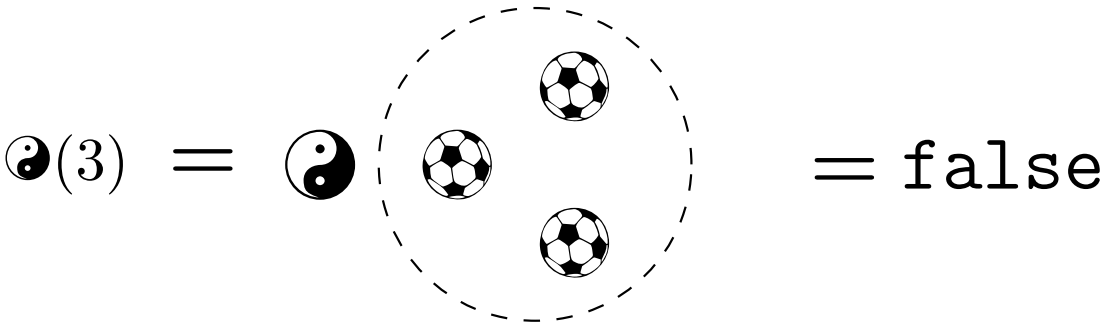
Ejemplo: programa recursivo yinyang

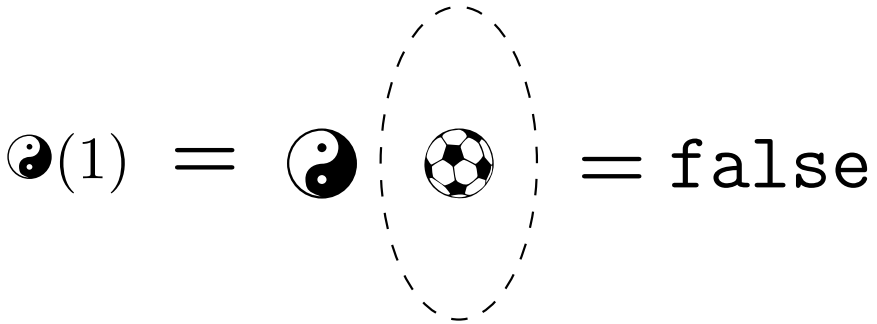





```
yinyang(0) = true
yinyang(1) = false
yinyang(s(s(N))) = yinyang(N)
```

Ejemplo: programa recursivo yinyang






```
yinyang(0) = true
yinyang(1) = false
yinyang(s(s(N))) = yinyang(N)
```

Ejemplo: programa recursivo maltese_cross

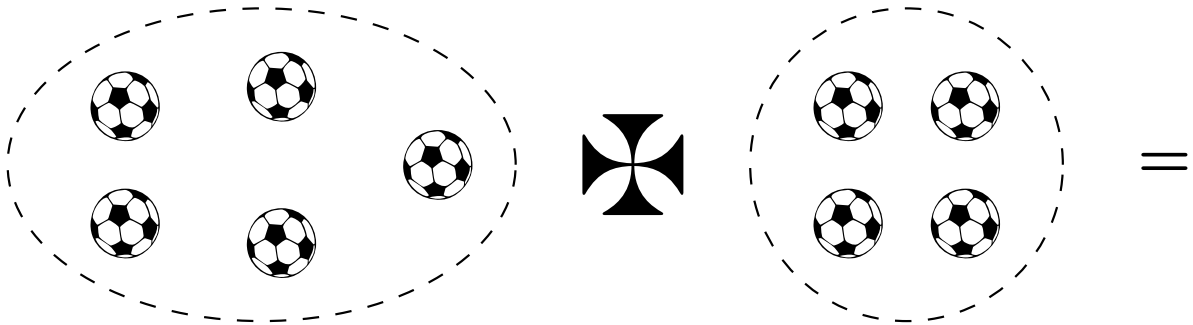
```
maltese_cross(0,0) = 0
maltese_cross(0,1) = 1
maltese_cross(1,0) = 1
maltese_cross(1,1) = 2
maltese_cross(2,2) = 4
maltese_cross(2,3) = 5
maltese_cross(4,1) = 5
```

```
maltese_cross(5,4) = 9
maltese_cross(6,3) = 9
maltese_cross(7,2) = 9
maltese_cross(8,1) = 9
maltese_cross(9,0) = 9
```

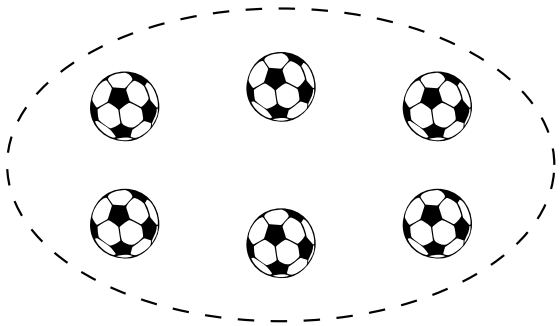
Ejemplo: programa recursivo maltese_cross

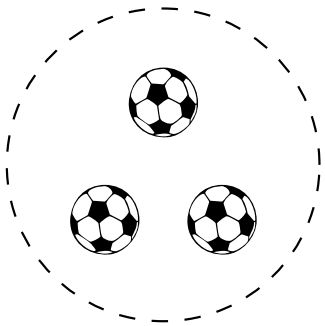
```
maltese_cross(0,0) = 0
maltese_cross(0,1) = 1
maltese_cross(1,0) = 1
maltese_cross(1,1) = 2
maltese_cross(2,2) = 4
maltese_cross(2,3) = 5
maltese_cross(4,1) = 5
```

```
maltese_cross(5,4) = 9
maltese_cross(6,3) = 9
maltese_cross(7,2) = 9
maltese_cross(8,1) = 9
maltese_cross(9,0) = 9
```

$5 \maltese 4 =$  $=$

Ejemplo: programa recursivo maltese_cross

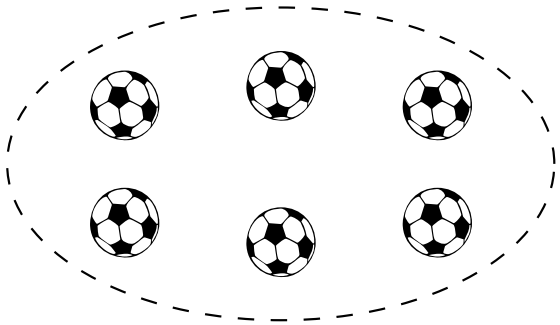
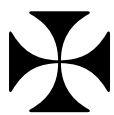
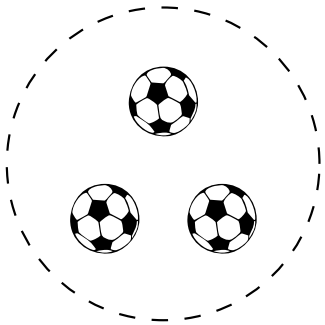




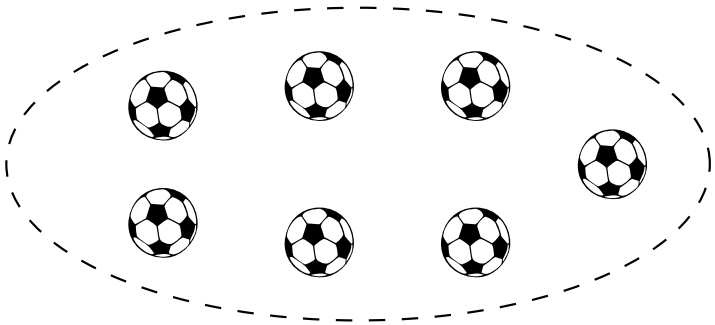
$$= (5 \cup 1) \maltese 3 =$$

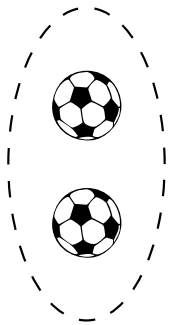
$$= ((5 \cup 1) \cup 1) \maltese 2 =$$

Ejemplo: programa recursivo maltese_cross

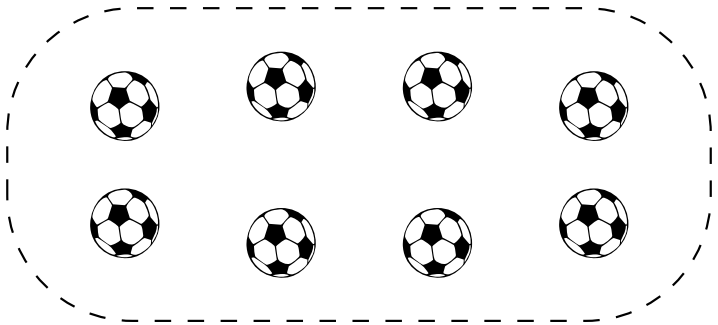
$$= (5 \cup 1) \maltese 3 =$$

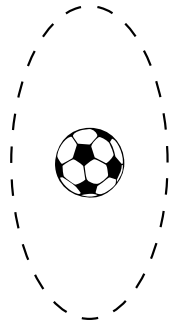




$$= ((5 \cup 1) \cup 1) \maltese 2 =$$

Ejemplo: programa recursivo maltese_cross

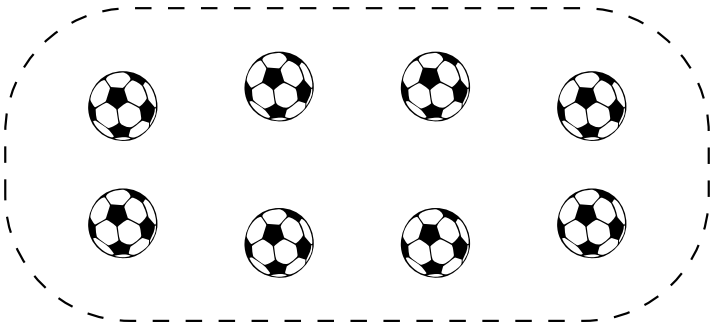
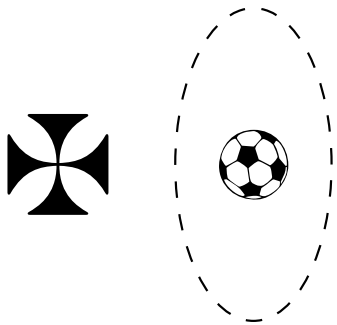




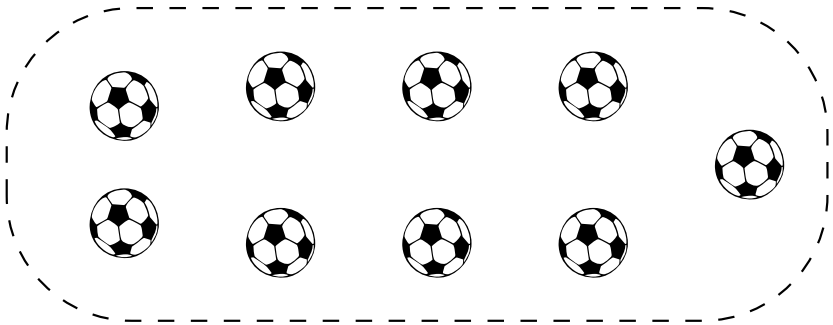
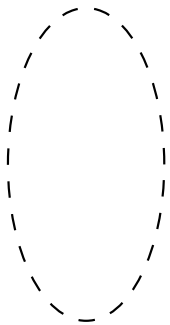
$$= (((5 \cup 1) \cup 1) \cup 1) \maltese 1 =$$

$$= ((((5 \cup 1) \cup 1) \cup 1) \cup 1) \cup 0$$

Ejemplo: programa recursivo maltese_cross

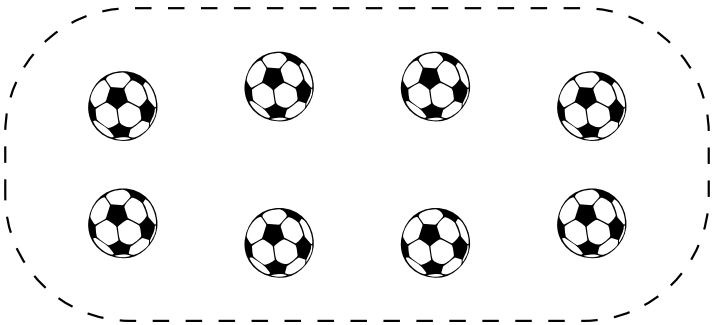
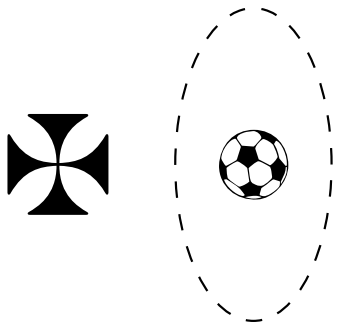
$$= (((5 \cup 1) \cup 1) \cup 1) \maltese 1 =$$

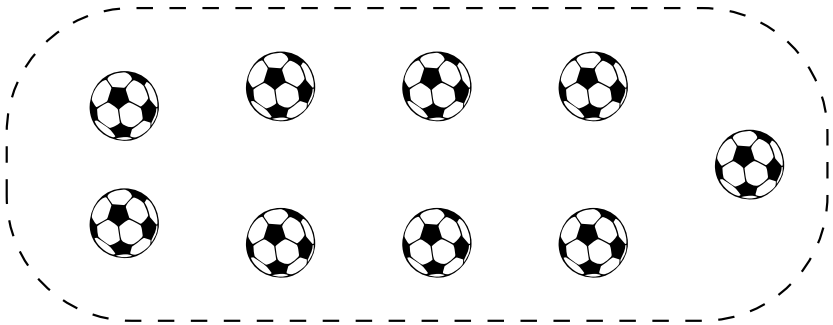
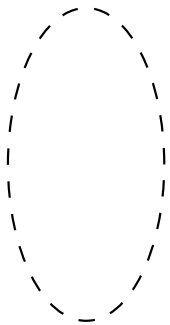
$$=$$

$$((((5 \cup 1) \cup 1) \cup 1) \cup 1) \cup 0$$

Ejemplo: programa recursivo maltese_cross

$= (((5 \cup 1) \cup 1) \cup 1) \maltese 1 =$

$=$

$((((5 \cup 1) \cup 1) \cup 1) \cup 1) \cup 0$

Ejemplo: programa recursivo maltese_cross

$$\begin{aligned}((((5 \cup 1) \cup 1) \cup 1) \cup 1) \cup 0 &= (((6 \cup 1) \cup 1) \cup 1) \cup 0 \\ &= ((7 \cup 1) \cup 1) \cup 0 \\ &= (8 \cup 1) \cup 0 \\ &= 9 \cup 0 \\ &= 9\end{aligned}$$

```
maltese_cross(N,0) = N
maltese_cross(N,s(M)) = maltese_cross(s(N),M)
```

Ejemplo: programa recursivo maltese_cross

$$\begin{aligned}((((5 \cup 1) \cup 1) \cup 1) \cup 1) \cup 0 &= (((6 \cup 1) \cup 1) \cup 1) \cup 0 \\ &= ((7 \cup 1) \cup 1) \cup 0 \\ &= (8 \cup 1) \cup 0 \\ &= 9 \cup 0 \\ &= 9\end{aligned}$$

```
maltese_cross(N,0) = N
maltese_cross(N,s(M)) = maltese_cross(s(N),M)
```