Ejemplos de ejecución para "subst"

1. > (subst (num 1729) 'x (num 2))
   - (num 1729) → expr
   - 'x → sub-id
   - (num 2) → val

   Siguiendo el cazamiento de patrones expr caza con "num (n)", por lo que simplemente devolvemos la expresión.

   [num (n) expr]

   ∴ Regresamos expr = (num 1729)

   ↳ 1729

2. > (subst (add (num 17) (num 29)) 'x (num 2))
   - (add (num 17) (num 29)) → expr
   - 'x → sub-id
   - (num 2) → val

   Siguiendo el cazamiento de patrones, expr caza con add (l r), por lo que procesamos recursivamente cada uno de sus lados.

   [add (l r) (add (subst l sub-id val)
   (subst r sub-id val))]

   (add (num 17) (num 29)) ← expr

   ↳ (add (subst (num 17) 'x (num 2))
   (subst (num 29) 'x (num 2)))

   Cada lado de la suma anterior caza con el patrón num (n) por lo que sólo devolveremos la expresión.

   ∴ El resultado final es: (add (num 17) (num 29))

   ↳ 17 + 29

3. > (subst (sub (num 25) (num 20)) 'x (num 2))
expr sub-id val

Siguiendo el cazamiento de patrones, expr caza con sub (l r), por lo que procesamos recursivamente cada uno de sus lados.

[sub (l r) (sub (subst l sub-id val)
(subst r sub-id val))]

(sub (num 25) (num 20)) ← expr

(sub (subst (num 25) 'x (num 2))
(subst (num 20) 'x (num 2))

Cada lado de la resta anterior caza con el patrón num (n) por lo que sólo devolvemos la expresión

∴ El resultado final es: (sub (num 25) (num 20))

4. > (subst (id 'x) 'x (num 20))
expr sub-id val

Siguiendo el cazamiento de patrones, expr caza con id (v), por lo que evaluamos el condicional.

[id (v) (if (Symbol=? sub-id v)
val
expr)]

(id 'x) ← expr

→ (if (symbol=? 'x)
(num 20)
(id 'x))

La condición del if es verdadera, por lo que devolvemos la expresión asociada al then

∴ El resultado final es: (num 20)

{+ x 8}, queremos sustituir x por 2

5. > (subst (add (id 'x) (num 8)) 'x (num 2))

expr — sub-id — val

Siguiendo el cazamiento de patrones, expr caza con add (l r), por lo que, como vimos en el ejemplo 2 procesamos recursivamente cada uno de sus lados.

(add (subst (id 'x) 'x (num 2))
(subst (num 8) 'x (num 2)))

Para el lado izquierdo:
(subst (id 'x) 'x (num 2))

expr — sub-id — val

Como vimos en el ejemplo 4, tenemos que evaluar el condicional.

(if (symbol=? 'x 'x) = (num 2)
(num 2)
(id 'x))

Para el lado derecho

(subst (num 8) 'x (num 2))

expr sub-id val

Como vimos en el ejemplo 1, simplemente devolvemos la expresión.

(num 8)

∴ El resultado final es: (add (num 2) (num 8))

Queremos cambiar y por 17.

6. >(subst (with 'x (add (id 'y) (id 'y)) (add (id 'x) (id 'y))) 'y (num 17))

expr sub-id val

Siguiendo el cazamiento de patrones, expr caza con with (bound-id named-expr bound-body), por lo que evaluamos el condicional

```
[with (bound-id named-expr bound-body)
      (if (symbol=? bound-id sub-id)
          (with bound-id
                (subst named-expr sub-id val)
                bound-body)
          (with bound-id
                (subst named-expr sub-id val)
                (subst bound-body sub-id val)))]
(if (symbol=? 'x 'y)
    (with 'x
          (subst (add (id 'y) (id 'y)) 'y (num 17))
          (add (id 'x) (id 'y)))
    (with 'x
          (subst (add (id 'y) (id 'y)) 'y (num 17))
          (subst (add (id 'x) (id 'y)) 'y (num 17)))))
```

(a condición del if es falsa, por lo que devolvemos la expresión asociada al else

```
(with 'x
      (subst (add (id 'y) (id 'y)) 'y (num 17))
      (subst (add (id 'x) (id 'y)) 'y (num 17)))
```

Procesando las llamadas pendientes:

```
(subst (add (id 'y) (id 'y)) 'y (num 17))
= (add (num 17) (num 17))
(subst (add (id 'x) (id 'y)) 'y (num 17))
= (add (id 'x) (num 17))
```

∴ El resultado final es:

```
(with 'x (add (num 17) (num 17)) (add (id 'x) (num 17)))
```