INTERCALADO

- Técnica para:
  - Simular paralelismo
  - Lanzar procesos de ejecución independiente
- Su interés:
  - No reside en cuestiones de eficiencia
  - Sí reside en la posibilidad de resolver problemas que no admiten un tratamiento secuencial
- Condiciones de aplicación:
  - Necesidad de ejecutar varios procesos
  - La solución está relacionada con alguno/s de ellos, pero no sabemos de antemano cuál/es
  - Los procesos no relacionados con la solución pueden no terminar
- En qué consiste:
  - Distribuir el tiempo de ejecución entre los distintos procesos (en forma de pasos)
  - Ningún proceso puede acaparar el tiempo
  - Mientras no se llegue a la solución todos los procesos tienen oportunidad de terminar

$$\psi(x) = \begin{cases} \mathbf{27} & \exists \mathbf{y}(\mathbf{x} \leq \mathbf{y} \leq \mathbf{2} * \mathbf{x} \wedge \varphi_{\mathbf{x}}(\mathbf{y}) = \mathbf{3} * \mathbf{y}) \\ \perp & \mathbf{c.c.} \end{cases}$$

```
PASOS := 1;
SALIR := false;
while not SALIR loop
      for DATO in X1..2*X1 loop
            SALIR := SALIR ∨ E(X1, DATO, PASOS,
3*DATO);
      end loop;
      PASOS:= succ(PASOS);
end loop;
X0 := 27;
```

LEY DE LA FUNCIÓN INVERSA (II)

La inversa de una función inyectiva

while-computable

(AUNQUE NO SEA TOTAL)

es también while-computable

- Si es while-computable la función inyectiva:

$$\psi : \Sigma^* \longrightarrow \Sigma^*$$

entonces también es while-computable la función INVERSA:

$$\psi^{-1} : \Sigma^* \longrightarrow \Sigma^*$$

- Demostración:

Por ser ψ computable existe un índice e $\in \sum^*$ tal que $\varphi_e \cong \psi$

```
X0 := ε; PASOS := 1; SALIR := false;
while not SALIR  loop
      for  DATO  in  0..PASOS  loop
           if  E(e, DATO, PASOS, X1)  then
                X0 := DATO;
                SALIR := true;
           end if;
      PASOS:= succ(PASOS);
end loop;
```

Si usamos las funciones de codificación:

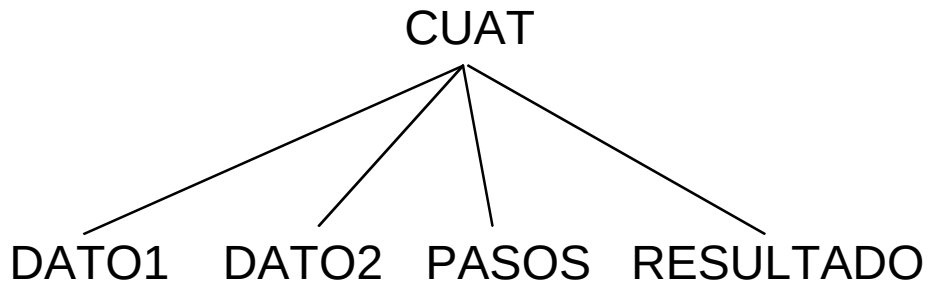
```
PAR := ε;
while not E(e, decod_2_1(PAR), decod_2_2(PAR), X1)
      loop    PAR := sig(PAR);
end loop;
X0:= decod_2_1(PAR);
```

$$\psi(x) = \begin{cases} \textbf{true} & \varphi_x \textbf{ no es inyectiva} \\ \bot & \textbf{c.c.} \end{cases}$$

$$\varphi_x \text{ no es inyectiva} \Leftrightarrow$$
$$\exists y \exists z\ (y \neq z \wedge \varphi_x(y) = \varphi_x(z)) \Leftrightarrow$$
$$\exists y \exists z \exists p \exists r\ (y \neq z \wedge E(x,y,p,r) \wedge E(x,z,p,r))$$

```
PASOS := 0;  X0 := false;
while not X0 loop
    for DATO1 in 0..PASOS loop
        for DATO2 in 0..ant(DATO1) loop
            if T(X1, DATO1, PASOS) and
                T(X1, DATO2, PASOS) then
                    R1 := Φ(X1, DATO1);
                    R2 := Φ(X1, DATO2);
                    X0 := X0 ∨ (R1=R2);
            end if;
        end loop;
    end loop;
    PASOS := succ(PASOS);
end loop;
```

- Si queremos utilizar las funciones de codificación:




```
CUAT := ε;
while not  (decod_4_1(CUAT)<> decod_4_2(CUAT) and
            E(X1, decod_4_1(CUAT), decod_4_3(CUAT),
                  decod_4_4(CUAT) and
            E(X1, decod_4_2(CUAT), decod_4_3(CUAT),
                  decod_4_4(CUAT))
      loop
            CUAT := sig(CUAT);
end loop;
```