CODIFICACIÓN DE PARES DE PALABRAS

	ε	**0**	**1**	**00**	**01**	**10**	**11**	**000**
ε	ε	1	10	010	111	0101	1100	00100
0	0	01	001	110	0100	1011	00011	
1	00	000	101	0011	1010	00010	...	
00	11	100	0010	1001	00001	01010		
01	011	0001	1000	00000	01001			
10	0000	0111	1111	01000				
11	0110	1110	00111					
000	1101	00110						
001	00101							

Σ = {a,b,c}

	ε	**a**	**b**	**c**	**aa**	**ab**	**ac**	**ba**
ε	ε	b	ab	bc	aab	acb	bbc	cbb
a	a	aa	bb	aaa	aca	bbb	cba	
b	c	ba	cc	abc	bba	cac	...	
c	ac	cb	abb	bac	cab	aaab		
aa	ca	aba	bab	caa	aaaa			
ab	aac	baa	bcc	ccc				
ac	acc	bcb	ccb					
ba	bca	cca						
bb	cbc							

CASO GENERAL

	w_0	**w_1**	**w_2**	**w_3**	**w_4**	**w_5**	**w_6**	**w_7**
w_0	w_0	w_2	w_5	w_9	w_{14}	w_{20}	w_{27}	w_{35}
w_1	w_1	w_4	w_8	w_{13}	w_{19}	w_{26}	w_{34}	
w_2	w_3	w_7	w_{12}	w_{18}	w_{25}	w_{33}	...	
w_3	w_6	w_{11}	w_{17}	w_{24}	w_{32}	w_{41}		
w_4	w_{10}	w_{16}	w_{23}	w_{31}	w_{40}			
w_5	w_{15}	w_{22}	w_{30}	w_{39}				
w_6	w_{21}	w_{29}	w_{38}					
w_7	w_{28}	w_{37}						
w_8	w_{36}							

FUNCIÓN CODIFICADORA

$$\mathbf{cod^2}: \mathbf{\Sigma^* \times \Sigma^* \rightarrow \Sigma^*}$$

$$\text{cod}^2(\mathbf{w_i}, \mathbf{w_j}) = \mathbf{w_k}$$

	$\mathbf{w_0}$	$\mathbf{w_1}$	$\mathbf{w_2}$	$\mathbf{w_3}$	$\mathbf{w_4}$		$\mathbf{w_j}$	
$\mathbf{w_0}$	w_0	w_2	w_5	w_9	w_{14}			
$\mathbf{w_1}$	w_1	w_4	w_8	w_{13}	w_{19}			
$\mathbf{w_2}$	w_3	w_7	w_{12}	w_{18}	w_{25}			
$\mathbf{w_3}$	w_6	w_{11}	w_{17}	w_{24}	w_{32}			
$\mathbf{w_4}$	w_{10}	w_{16}	w_{23}	w_{31}	w_{40}			
						...		
$\mathbf{w_i}$							$\mathbf{w_k}$	

- Definición inductiva:

  $\text{cod}^2(\varepsilon, \varepsilon) = \varepsilon$

  $\text{cod}^2(\text{sig}(\mathbf{w_i}), \varepsilon) = \text{sig}(\text{cod}^2(\varepsilon, \mathbf{w_i}))$

  $\text{cod}^2(\mathbf{w_i}, \text{sig}(\mathbf{w_j})) = \text{sig}(\text{cod}^2(\text{sig}(\mathbf{w_i}), \mathbf{w_j}))$

PROPIEDADES DE *COD*^{2}

- Total
- Inyectiva
- Sobreyectiva
- Computable

```
LINEA := ε;
COLUMNA := ε;
X0 := ε;
-- X0 = cod_2(LINEA,COLUMNA)
while LINEA /=X1 or COLUMNA /=X2 loop
      if LINEA = ε then
            LINEA := sig(COLUMNA );
            COLUMNA := ε;
      else LINEA := ant(LINEA );
            COLUMNA := sig(COLUMNA);
      end if;
      X0 := sig(X0);
end loop;
```

FUNCIONES DECODIFICADORAS

$$\mathbf{decod_{2,1}}: \Sigma^* \rightarrow \Sigma^*$$

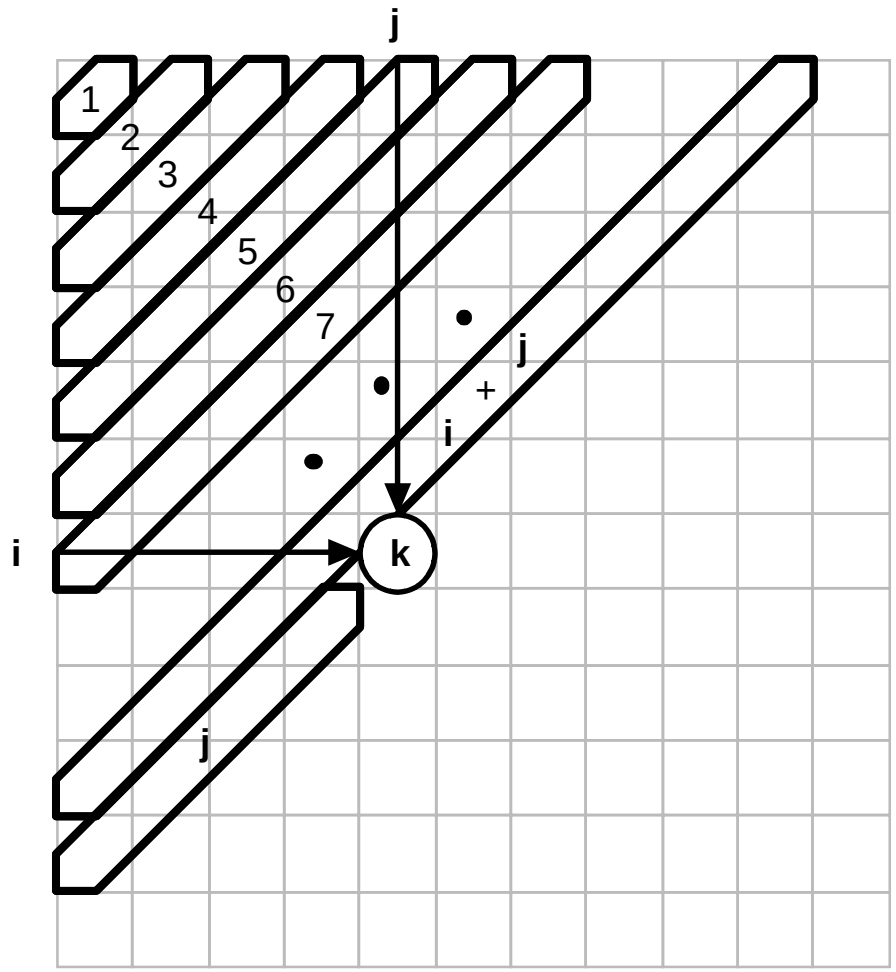
$$\mathbf{decod_{2,2}}: \Sigma^* \rightarrow \Sigma^*$$

$$\mathbf{w_k} = \mathrm{cod}^2(\mathbf{w_i}, \mathbf{w_j}) \quad \Rightarrow \quad \begin{cases} \mathrm{decod}_{2,1}(\mathbf{w_k}) = \mathbf{w_i} \\ \mathrm{decod}_{2,2}(\mathbf{w_k}) = \mathbf{w_j} \end{cases}$$

- Existen por ser **cod²** biyectiva
- Totales y sobreyectivas
- Computables:

```
LINEA := ε;  COLUMNA := ε;  AUX := ε;
-- AUX = cod_2(LINEA,COLUMNA)
while  AUX /= X1  loop
      if  LINEA = ε  then
            LINEA := sig(COLUMNA );  COLUMNA := ε;
      else LINEA := ant(LINEA );
            COLUMNA := sig(COLUMNA);
      end  if;
      AUX := sig(AUX);
end loop;
X0 := LINEA;        -- respectivamente X0 := COLUMNA;
```

CÁLCULO NUMÉRICO DE *COD* Y *DECOD*



- Para llegar la palabra **k**-ésima hay que contar:
  - las que constituyen las diagonales 1..**i**+**j**
  - las **j**+1primeras de la diagonal **i**+**j**+1
  - descontar 1 (se comienza desde 0)

$$\mathbf{k} = \sum_{p=1}^{\mathbf{i}+\mathbf{j}} p + \mathbf{j} + 1 - 1 = \frac{(\mathbf{i}+\mathbf{j}) * (\mathbf{i}+\mathbf{j}+1)}{2} + \mathbf{j}$$

- Para la inversa se puede usar la estimación:

$$\mathbf{i}+\mathbf{j} \approx \sqrt{2 * \mathbf{k}}$$

CÓDIGOS DE n-TUPLAS

$$\mathbf{cod^n}: \Sigma^{*n} \rightarrow \Sigma^*$$
$$\mathbf{decod_{n,1}}: \Sigma^* \rightarrow \Sigma^*$$
$$\mathbf{decod_{n,2}}: \Sigma^* \rightarrow \Sigma^*$$
$$\ldots$$
$$\mathbf{decod_{n,n}}: \Sigma^* \rightarrow \Sigma^*$$

$$\mathrm{cod}^{\mathbf{k}}(\mathbf{w_{i_1}}, \mathbf{w_{i_2}}, \ldots, \mathbf{w_{i_n}}) = \mathbf{w_k} \Rightarrow \begin{cases} \mathrm{decod}_{\mathbf{n},1}(\mathbf{w_k}) = \mathbf{w_{i_1}} \\ \mathrm{decod}_{\mathbf{n},2}(\mathbf{w_k}) = \mathbf{w_{i_2}} \\ \ldots \\ \mathrm{decod}_{\mathbf{n},\mathbf{n}}(\mathbf{w_k}) = \mathbf{w_{i_n}} \end{cases}$$

- Definición inductiva:

$$\mathrm{cod}^1(\mathbf{z}) = \mathbf{z}$$

$$\mathrm{cod}^{\mathbf{n}+1}(\mathbf{z_1}, \ldots, \mathbf{z_{n+1}}) = \mathrm{cod}^2(\mathbf{z_1}, \mathrm{cod}^{\mathbf{n}}(\mathbf{z_2}, \ldots, \mathbf{z_{n+1}}))$$

$$\mathrm{decod}_{\mathbf{k},\mathbf{i}}(\mathbf{w}) = \begin{cases} \mathbf{w} & \mathbf{i} = 1 \wedge \mathbf{k} = 1 \\ \mathrm{decod}_{2,1}(\mathbf{w}) & \mathbf{i} = 1 \wedge \mathbf{k} > 1 \\ \mathrm{decod}_{\mathbf{k}-1,\mathbf{i}-1}(\mathrm{decod}_{2,2}(\mathbf{w})) & \mathbf{i} > 1 \wedge \mathbf{k} > 1 \end{cases}$$

- Todas son computables

IMPLEMENTACIÓN DE TIPOS DE DATOS

- Computación y datos:
  - En la máquina: los datos son expresados mediante cadenas de símbolos
  - En la realidad: los datos son interpretados por los humanos

- Para poder utilizar un tipo de datos:
  - Tenemos que describir cómo se van a representar sus elementos mediante cadenas de símbolos
  - Tenemos que diseñar programas que manipulen adecuadamente dichas cadenas de símbolos

- Función de INTERPRETACIÓN $\mathfrak{I}_T: \sum^* \rightarrow T$
  - Es un **convenio** que describe el **significado** de las palabras cuando estas representan **datos del tipo** T

$$\text{'}\mathbf{2AF}\text{'} \longrightarrow 687$$

$$\text{'}\mathbf{3E\text{-}02}\text{'} \longrightarrow 0.03$$

$$\text{'}\mathbf{04}\text{'} \longrightarrow \text{Abril}$$

  - Idealmente **biyectiva**

IMPLEMENTACIÓN DE OPERACIONES DE UN TIPO DE DATOS T

- Computación de ψ: $T_1 \times \ldots \times T_k \to T$ mediante **P**
  - No podemos manipular elementos de T_i
  - Manipulamos las palabras que los representan

$$\begin{array}{ccccc} & & \Psi & & \\ & T_1 \times \ldots \times T_k & \Longrightarrow & T & \\ (\mathfrak{I}_{T_1}, \ldots, \mathfrak{I}_{T_k}) & \Uparrow & & \Uparrow & \mathfrak{I}_T \\ & \Sigma^* \times \ldots \times \Sigma^* & \underset{\varphi_P^k}{\Longrightarrow} & \Sigma^* & \end{array}$$

- ¿Qué debe hacer **P** con los datos $z_1, z_2, \ldots, z_k$?

  Si $\psi(\mathfrak{I}_{T_1}(z_1), \mathfrak{I}_{T_2}(z_2), \ldots, \mathfrak{I}_{T_k}(z_k))\uparrow$ entonces

  $$\varphi_P^k(z_1, z_2, \ldots, z_k)\uparrow$$

  Si $\psi(\mathfrak{I}_{T_1}(z_1), \mathfrak{I}_{T_2}(z_2), \ldots, \mathfrak{I}_{T_k}(z_k))\downarrow$ entonces

  $$\varphi_P^k(z_1, z_2, \ldots, z_k)\downarrow$$

  y además

  $$\mathfrak{I}_T(\varphi_P^k(z_1, z_2, \ldots, z_k)) = \psi(\mathfrak{I}_{T_1}(z_1), \mathfrak{I}_{T_2}(z_2), \ldots, \mathfrak{I}_{T_k}(z_k))$$

- Si $\mathfrak{I}_T$ es biyectiva decimos entonces que la implementación es **estricta**)

IMPLEMENTACIÓN DE NATURALES

	$\Sigma = \{a,b\}$	N
w_0	ε	0
w_1	**a**	1
w_2	**b**	2
w_3	**aa**	3
...	...	...
w_{108}	**babbab**	108
...	...	...

- INTERPRETACION:

$$\mathfrak{I}_N(\varepsilon) = 0$$

$$\mathfrak{I}_N(sig(w)) = \mathfrak{I}_N(w)+1$$

- REPRESENTACIÓN:

$$\mathfrak{R}_N(n) = \mathfrak{I}_N^{-1}(n) = w_n$$

- Es estricta

EJEMPLO: SUMA DE NATURALES

- Ejemplo con el alfabeto {**a**, **b**, **c**}:
  - 1347 + 228 = 1575
  - El programa que compute la suma:
    - si recibe $\mathbf{w_{1347}}$ (**abaabac**) y $\mathbf{w_{228}}$ (**baccc**) en X1 y X2
    - entonces produce $\mathbf{w_{1575}}$ (**abcccbc**) en X0

- Programa para computar (*implementar*) la suma:

```
X0:= X1;
AUX := X2;
-- X0+AUX = X1+X2
while nonem?(AUX) loop
      X0 := sig(X0);
      AUX := ant(AUX);
end loop;
```

ANTES Y DESPUÉS DE LA IMPLEMENTACIÓN

- Requisitos previos:
  - Definir con precisión el dominio o conjunto de datos T
  - Elegir un conjunto de funciones **F** (**operaciones básicas**) del Tipo de Datos a partir del cuál se puedan programar todas las demás
- Pasos a seguir:
  - Definir la función de interpretación $\mathfrak{I}_T: \sum^* \rightarrow \mathsf{T}$, a ser posible estricta (es útil pensar antes en términos de $\mathfrak{R}_T$)
  - Demostrar que todas las funciones de **F** son computables (hay que considerar la interpretación de las palabras)
  - Si la interpretación no es estricta es necesario implementar la operación IGUALDAD
- Beneficios obtenidos:
  - Podemos utilizar las **constantes** del Tipo en los macropogramas
  - Podemos utilizar las **operaciones básicas** del Tipo en los macropogramas
  - podemos demostrar la computabilidad de otras funciones del Tipo (ya no es necesario considerar la interpretación de las palabras)
- CONCLUSIÓN: El tipo de Datos queda incorporado al lenguaje de los macroprogramas

OPERACIONES CON NATURALES

- OPERACIONES BASICAS

  * IGUALDAD No es necesaria (interpretación estricta). Tampoco la DESIGUALDAD

  * SUCESOR succ : $N \rightarrow N$

  X0:= sig(X1);

  * PREDECESOR pred : $N \rightarrow N$

  X0:= ant(X1);

- OTRAS OPERACIONES:

  * SUMA

  * DIFERENCIA .-. : $N \times N \rightarrow N$

```
X0:= X1;   AUX := X2;
-- X0-AUX = X1-X2
while  not  AUX = 0  loop
      X0 := pred(X0);
      AUX := pred(AUX);
end  loop;
```

- EJEMPLO DE OPERACIÓN EXTERNA:

  * LONGITUD $|.| : \Sigma^* \rightarrow N$

```
AUX:= X1;
X0:= 0;
-- X0+|AUX| = |X1|
while nonem?(AUX) loop
      X0:= succ(X0);
      AUX:= cdr(AUX);
end loop;
```

- EJEMPLO DE PREDICADO:

  * MENOR O IGUAL $.\leq. : N \times N \rightarrow B$

```
while X1<>0 and X2<>0 loop
      X1:= pred(X1);
      X2:= pred(X2);
end loop;
if X1=0 then X0:= 'a';
else X0:= ε;
end if;
```

IMPLEMENTACIÓN DE BOOLEANOS

	$\Sigma = \{a,b\}$	B
w_0	ε	false
w_1	**a**	true
w_2	**b**	true
w_3	**aa**	true
...	...	...
w_{108}	**babbab**	true
...	...	...

- INTERPRETACION:

$$\mathfrak{I}_B(w) = \begin{cases} \textbf{false} & w = \varepsilon \\ \textbf{true} & w \neq \varepsilon \end{cases}$$

- No existe la función de representación (la implementación no es estricta)

OPERACIONES CON BOOLEANOS

- OPERACIONES BASICAS

  * IGUALDAD $.\leftrightarrow. : B \times B \rightarrow B$

  X0:= ε;

  if X1 = X2 or (nonem?(X1) and nonem?(X2)) then

  X0:= $cons_a$(X0);

  end if;

  * NEGACIÓN $\neg. : B \rightarrow B$

  X0:= ε;

  if not nonem?(X1) then X0:= $cons_a$(X0); end if;

  * CONJUNCIÓN $.\wedge. : B \times B \rightarrow B$

  X0:= ε;

  if nonem?(X1) and nonem?(X2) then

  X0:= $cons_a$(X0);

  end if;

  * DISYUNCIÓN $.\vee. : B \times B \rightarrow B$

  X0:= ε;

  if nonem?(X1) or nonem?(X2) then

  X0:= $cons_a$(X0);

  end if;

LA FUNCIÓN DECOD

- decod: $N \times N \times \Sigma^* \rightarrow \Sigma^*$

decod(i, j, w) = $decod_{i,j}(w)$

```
if X1 < X2 or X1 = 0 or X2 = 0  then
      X0 := ⊥⊥ ;
elsif  X1 = 1  then  X0 := X3;
else  AUX := X3;
      if  X1 = X2  then
            MAX := pred(X2);
      else  MAX := X2;
      end if;
      for  IND  in  1..MAX  loop
            X0 := decod_2_1(AUX);
            AUX := decod_2_2(AUX);
      end loop;
      if  X1 = X2  then
            X0 := AUX;
      end if;
end if;
```

IMPLEMENTACIÓN DE PILAS: INTERPRETACIÓN

- REPRESENTACIÓN de una **pila** mediante una **palabra**:

$$<] \quad \Rightarrow \quad \varepsilon$$

$$<aa, aba, bb] \quad \Rightarrow \quad cod^4(aa, aba, sig(bb), \varepsilon)$$

- INTERPRETACIÓN de una **palabra** como la **pila** que representa:

$$\Im_P(w) = \begin{cases} <] & w = \varepsilon \\ < ant(decod_{2,1}(w))] & w \neq \varepsilon \wedge decod_{2,2}(w) = \varepsilon \\ empilar(decod_{2,1}(w), \Im_{\text{œ}}(decod_{2,2}(w))) & c.c. \end{cases}$$

OPERACIONES CON PILAS

- **P_vacía?:** $P \rightarrow B$

```
X0 := X1 = ε;
```

- **empilar**: $\Sigma^* \times P \rightarrow P$

```
if X2 = ε then X0 := cod_2(sig(X1), ε);
else X0 := cod_2(X1,X2); end if;
```

- **desempilar:** $P \rightarrow P$

```
if X1 = ε then X0 := ⊥⊥;
else X0 := decod_2_2(X1); end if;
```

- **cima:** $P \rightarrow \Sigma^*$

```
if X1 = ε then X0 := ⊥⊥;
elsif decod_2_2(X1) = ε then
      X0 := ant(decod_2_1(X1));
else X0 := decod_2_1(X1); end if;
```

IMPLEMENTACIÓN DE VECTORES DINAMICOS: INTERPRETACIÓN

- REPRESENTACIÓN de un **vector** mediante una **palabra**:

No existe el vector vacío

$$(\varepsilon) \quad \Rightarrow \quad \mathbf{cod^2(0, \varepsilon) = \varepsilon}$$

$$\mathbf{(aa, \varepsilon, aa, b)} \quad \Rightarrow \quad \mathbf{cod^5(3, aa, \varepsilon, aa, b)}$$

- INTERPRETACIÓN de una **palabra** como el **vector** que representa:

$\mathfrak{I}_V(\mathbf{w}) = (\mathbf{z_0}, \mathbf{z_1}, \ldots, \mathbf{z_k})$ donde

- $\mathbf{k} = \mathfrak{I}_N(\text{decod}_{2,1}(\mathbf{w}))$
- $\forall i\ (0 \leq i \leq \mathbf{k} \rightarrow \mathbf{z_i} = \text{decod}_{k+1,i+1}(\text{decod}_{2,2}(\mathbf{w}))$

OPERACIONES CON VECTORES I

- **ult_indice:** $V \rightarrow N$

```
X0 := decod_2_1 (X1);
```

- **acceso:** $V \times N \rightarrow \Sigma^*$

```
if  X2 <= decod_2_1(X1)  then
      X0 := decod(decod_2_1(X1) + 1,
                  X2 + 1, decod_2_2(X1));
else X0 := ⊥⊥ ;
end if;
```

- Notación para macros:

acceso (M, I) ⇒ M(I)

M:= modifica(M, I, W) ⇒ M(I):= W;

(Macro-asignación a componente de un vector)

OPERACIONES CON VECTORES II

- modifica: $V \times N \times \Sigma^* \rightarrow V$

```
AUX := X1; PILAUX := <];
for IND in 0..pred(decod_2_1(X1)) loop
      AUX := decod_2_2(AUX);
      PILAUX := empilar(decod_2_1(AUX), PILAUX);
end loop;
PILAUX:=empilar(decod_2_2(AUX), PILAUX);
-- Los elementos de X1 están invertidos en PILAUX
if X2 < decod_2_1(X1) then
      -- X2 es un índice del vector distinto del último
      X0 := cima (PILAUX);
      for IND in suc(X2) .. pred(decod_2_1(X1)) loop
            PILAUX := desempilar(PILAUX);
            X0:= cod_2 (cima(PILAUX), X0);
      end loop;
      X0:= cod_2(X3, X0);
      PILAUX := desempilar(desempilar(PILAUX));
      MAXIND := decod_2_1(X1);
-- El valor modificado y los posteriores ya están en X0
```

```
    else
        -- X2 es el último índice del vector o uno posterior
        X0:= X3;
        for IND in suc(decod_2_1(X1))..pred(X2) loop
            X0:= cod_2 (ε, X0);
        end loop;
        -- Si era uno posterior hemos rellenado los huecos
        if X2 = decod_2_1 (X1) then
            PILAUX := desempilar(PILAUX);
        end if;
        -- Si era el último lo hemos retirado de la pila
        MAXIND := X2;
    -- El valor modificado y los posibles huecos ya están en X0
    end if;
    -- Sólo queda volcar los valores de PILAUX
    while not P_vacía?(PILAUX) loop
        X0:= cod_2 (cima(PILAUX), X0);
        PILAUX := desempilar(PILAUX);
    end loop;
    X0:= cod_2 (MAXIND, X0);
```