Universidad de Oviedo
*Universidá d'Uviéu*
*University of Oviedo*

# Resultados Fundamentales de Computabilidad

**Departamento de Informática**
**Universidad de Oviedo**

# CONTENIDO

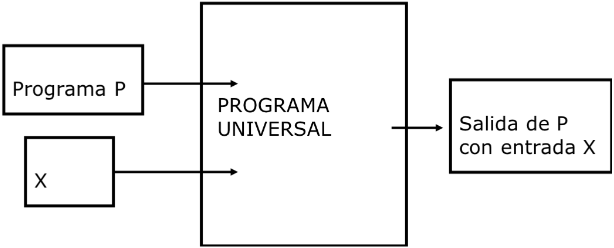
# Parte I

# Codificación de los programas

# Resultados fundamentales y codificación de los programas

- En este tema, estudiaremos dos de los principales resultados de la Computabilidad
  - Universalidad
  - Parametrización
- En estos resultados, las funciones computables (programas) serán la entrada de otras funciones computables (programas)
- Necesitamos una forma de **codificar** programas mediante números naturales

# EL PROGRAMA UNIVERSAL



# Parte II

# Enumeración de los algoritmos

# La idea de la codificación

- Codificar un programa mediante un número natural nos permitiría utilizar programas como entrada de otros programas
- La idea para codificar es muy simple: asignamos un número único a cada programa
- La arquitectura von Neumann está basada en algo similar, ya que los programas y los datos se almacenan de la misma forma
- El primero en utilizar una codificación de un sistema formal fue Kurt Gödel en la demostración de sus famosos Teoremas de Incompletitud

# Nuestra codificación

- Hay muchas formas (equivalentes computacionalmente) de codificar programas
- Una de las más simples consiste en imitar la forma en la que codificamos programas en nuestros ordenadores:
    - Primero, asignamos un número de 8 bits a cada posible símbolo (su código ASCII)
    - Después, concatenamos los números de todos los simbolos del programa
- El resultado será un número natural (que normalmente será enorme) que será único para cada programa y al que llamaremos el **código** del programa

## Un ejemplo

- Considera el siguiente programa

```
begin
    X1:=0
end
```

- Su código en binario sería

| | | | |
|---|---|---|---|
| **01100010** | **01100101** | **01100111** | **01101001** |
| **01101110** | **00001010** | **00100000** | **00100000** |
| **01011000** | **00110001** | **00111010** | **00111101** |
| **00110000** | **00001010** | **01100101** | **01101110** |
| **01100100** | | | |

- Que corresponde al número natural
  33482460930773914958281235596343704383076

## Propiedades de la codificación

- Si extendemos el modelo de los programas while para incluir variables y operaciones de manipulación de caracteres, la codificación descrita es claramente computable
- Por tanto, la función

$$\mathbf{cod} : \mathbf{Programas} \to \mathbb{N}$$

  es computable
- **cod** es también total (todo programa tiene un código) e inyectiva (programas diferentes tienen códigos diferentes)
- Además, dado un número que codifica un programa, podemos obtener de nuevo dicho programa
- Sin embargo, hay números que no son el código de ningún programa (por ejemplo, 0)

# DEFINIENDO *decod*

- Definimos la función $\mathit{decod} : \mathbb{N} \to \mathit{Programas}$ como sigue

$$\mathit{decod}(n) = \begin{cases} P & \text{si existe } P \text{ tal que } \mathit{cod}(P) = n \\ Q & \text{en otro caso} \end{cases}$$

  donde $Q$ es un programa fijo, por ejemplo

```
begin
   X1:=0
end
```

# Propiedades de la codificación (II)

- Obsérvese que *decod* puede devolver $Q$ en infinitos casos (uno para cada número que no codifique ningún programa)
- Esto no es importante porque:
  - Sólo estamos interesados en las funciones computadas por los programas
  - Cada función computable es computada por infinitos programas diferentes (la función semántica de un programa no cambia si añadimos una o más veces una instrucción como $X2 := 0$ al final)
- Es fácil comprobar que $decod(cod(P)) = P$
- También se puede ver que *decod* es total (trivial) y computable (se podría comprobar si un número codifica un programa while correcto utilizando expresiones regulares y gramáticas libres de contexto)

# En resumen

- Hemos definido funciones

$$cod : Programas \to \mathbb{N}$$

y

$$decod : \mathbb{N} \to Programas$$

- Ambas son totales y computables
- *cod* es además inyectiva
- Se cumple $decod(cod(P)) = P$
- *Pregunta: ¿Se cumple* $cod(decod(n)) = n$*?*

# Algo de notación

- Recuerda que a cada programa $P$ le corresponden infinitas funciones computables, una por cada posible aridad:

$$\varphi_P^{(j)} : \mathbb{N}^j \to \mathbb{N}$$

- En lugar de $P$, utilizaremos a menudo el código de $P$ como subíndice en la expresión anterior. Es decir:

$$\varphi_e^{(j)} = \varphi_P^{(j)}$$

si $e = cod(P)$

# Parte III

## Universalidad

# La idea de una Función Universal

- Ahora que ya podemos codificar programas mediante números, podemos tratar las entradas de una función como si fuesen programas
- Por ejemplo, considérese la función

$$\Phi(e, x) = \varphi_e(x)$$

- Esta función, siendo $e$ el código de un programa, y $x$ un número, devolverá el resultado de ejecutar el programa $P = decod(e)$ con entrada $x$

# La idea de una Función Universal

- Decimos que esta función es **universal** porque podemos utilizarla para reproducir el comportamiento de *c*ualquier funcíon computable unaria ya que
  - **Universal en cuanto a programas**: El primer argumento **e** de la función de Universalidad $\Phi(e, x)$ es la codificación de un programa. Al recorrer dicho argumento todos los naturales, se alcanzan todos los programas que existen.
  - **Universal en cuanto a entradas**: El resto de los argumentos son las entradas para el programa $P_e$ que se debe "simular" con $\Phi(e, x)$ , fijada de antemano la aridad $j$ de su función semántica.
- Es más, podemos considerar una función universal para cada aridad:

$$\Phi(e, x_1, x_2, \ldots, x_j) = \varphi_e^{(j)}(x_1, x_2, \ldots, x_j)$$

# La Función Universal es computable

- Aunque al principio parece casi increible, es posible probar que cada función universal $\Phi(e, x_1, x_2, \ldots, x_j)$ es computable
- Podemos proceder como sigue:
  - Primero, obtenemos $P = decod(e)$ (ya sabemos que eso es computable)
  - Después, simulamos la ejecución de $P$ con entrada $x_1, x_2, \ldots, x_j$
  - Para acabar, devolvemos el resultado de la computación anterior
- El segundo paso requiere una demostración larga y tediosa, pero se puede probar que siempre es computable
- Es comparable al funcionamiento de un intérprete o un sistema operativo

# El Teorema de Universalidad

- La conclusión de la diapositiva anterior es tan importante que debemos ponerla en forma de teorema

**Teorema de Universalidad**

Para cada $j \geq 1$, la función universal

$$\begin{array}{rccc} \Phi : & \mathbb{N}^{j+1} & \to & \mathbb{N} \\ & \Phi^{(j+1)}(e, x_1, x_2, \ldots, x_j) & = & \varphi_e^{(j)}(x_1, x_2, \ldots, x_j) \\ & \forall e, x_1, x_2, \ldots, x_j & & \end{array}$$

es computable

## La Macro de Universalidad

- Ya que las funciones universales son computables para cualquier aridad, hay programas while que las computan y los podemos utilizar como macros
- A partir de ahora, en nuestros programas while podremos utilizar macro-instrucciones como

$$Z := U(X, Y)$$

- Cuando esta macro se ejecuta, el valor $\varphi_X(Y)$ (es decir, el resultado de ejecutar el programa $P = decod(X)$ con entrada $Y$) se almacenará en $Z$
- Obsérvese que $\varphi_X(Y)$ podría ser indeterminado, en cuyo caso el programa se quedará permanentemente atascado en la ejecución de la macro

# Parte IV

## PARAMETRIZACIÓN

## La idea de la Parametrización

- A veces, cuando tenemos una función, es útil fijar algunas de sus entradas para obtener otras funciones
- Por ejemplo, si tenemos

$$f(x, y) = x * y$$

y fijamos $x = 2$ obtendremos una nueva función, con una única variable, a la que podemos llamar $g$:

$$g(y) = f(2, y) = 2 * y$$

- Ocurre algo similar en algunos lenguajes de programación, como C++ o Python, cuando definimos funciones con parámetros por defecto

## PARAMETRIZACIÓN Y PROGRAMAS

- Supón que tenemos un programa $P$ que computa $f(x, y) = x * y$. Ahora considera el programa $Q$

```
begin
    X2:=X1;
    X1:=2;
    P
end
```

- Claramente, la función unaria semántica de $Q$ es

$$g(y) = f(2, y) = 2 * y$$

  y si conocemos $cod(P)$ podríamos computar facilmente $cod(Q)$

## Parametrización y Programas (II)

- Podemos generalizar el ejemplo anterior. Considérese el siguiente programa:

```
begin
    X2:=X1;
    X1:=C;
    P
end
```

  donde $C$ es una constante.

- Entonces, la función unaria semántica de este nuevo programa es $h(y) = f(C, y) = C * y$
- Más importante, **podemos computar** el código de este nuevo programa a partir de $cod(P)$ y del valor $C$

# Parametrización y Programas (III)

- El programa $P$ de los ejemplos no tiene nada de particular, así que podemos utilizar *cualquier* programa $P$
- Cuando variamos $P$, estamos considerando todas las funciones computables binarias $\varphi_P(x, y)$ y obtenemos las funciones computables unarias

$$g_{(P,C)}(y) = \varphi_P(C, y)$$

- Nótese que reducimos (fijamos) un parámetro y la nueva función depende de $P$ y de $C$
- De hecho, no sólo podemos considerar funciones de dos variables, sino de todas las variables que queramos

## Parametrización en general

- Considera un programa $P$, un número de variables $m$ que se fijan, y un número de variables $n$ que seguirán libres
- Considera también $m$ constantes $C_1, \ldots, C_m$ y el programa

```
begin
    Xm+1:=X1;
    ...
    Xm+n:=Xn;
    X1:=C1;
    ...
    Xm:=Cm;
    P
end
```

- La función semántica $n$-aria de este programa es $h(x_1, \ldots, x_n) = \varphi_P^{(m+n)}(C_1, \ldots, C_m, x_1, \ldots, x_n)$
- Y, de nuevo, **podemos computar** el código de este nuevo programa a partir de $cod(P)$ y de los valores $C_1, \ldots, C_m$

# El Teorema de Parametrización

- Todo el razonamiento anterior nos lleva al Teorema de Parametrización (también conocido como el Teorema *s-m-n*)

**Teorema de Parametrización**

Para cada $m \geq 1$ y $n \geq 1$ existe una función total y computable $s_n^m$ tal que

$$\varphi^{(n)}_{s_n^m(e,y_1,\ldots,y_m)}(x_1,\ldots,x_n) = \varphi_e^{(m+n)}(y_1,\ldots,y_m,x_1,\ldots,x_n)$$

para todo $e, y_1,\ldots,y_m,x_1,\ldots,x_n$