

Paradigmas de Programación

Departamento de Informática
Universidad de Valladolid

Curso 2012-13

Grado en Ingeniería Informática
Grado en Ingeniería Informática de Sistemas





Definición

paradigma.

(Del lat. *paradigma*, y este del gr. παράδειγμα).

1. m. Ejemplo o ejemplar.
2. m. Teoría cuyo núcleo central se acepta sin cuestionar y que suministra la base y modelo para resolver problemas y avanzar en el conocimiento; p. ej., en la ciencia, las leyes del movimiento y la gravitación de Newton y la teoría de la evolución de Darwin.

Real Academia Española © Todos los derechos reservados

- “*Un **paradigma de programación** indica un método de realizar cálculos y la manera en que se deben estructurar y organizar las tareas que debe llevar a cabo un programa*”
- Los paradigmas fundamentales están asociados a determinados **modelos de cómputo**.
- También se asocian a un determinado **estilo de programación**
- Los lenguajes de programación suelen implementar, a menudo de forma parcial, **varios** paradigmas.

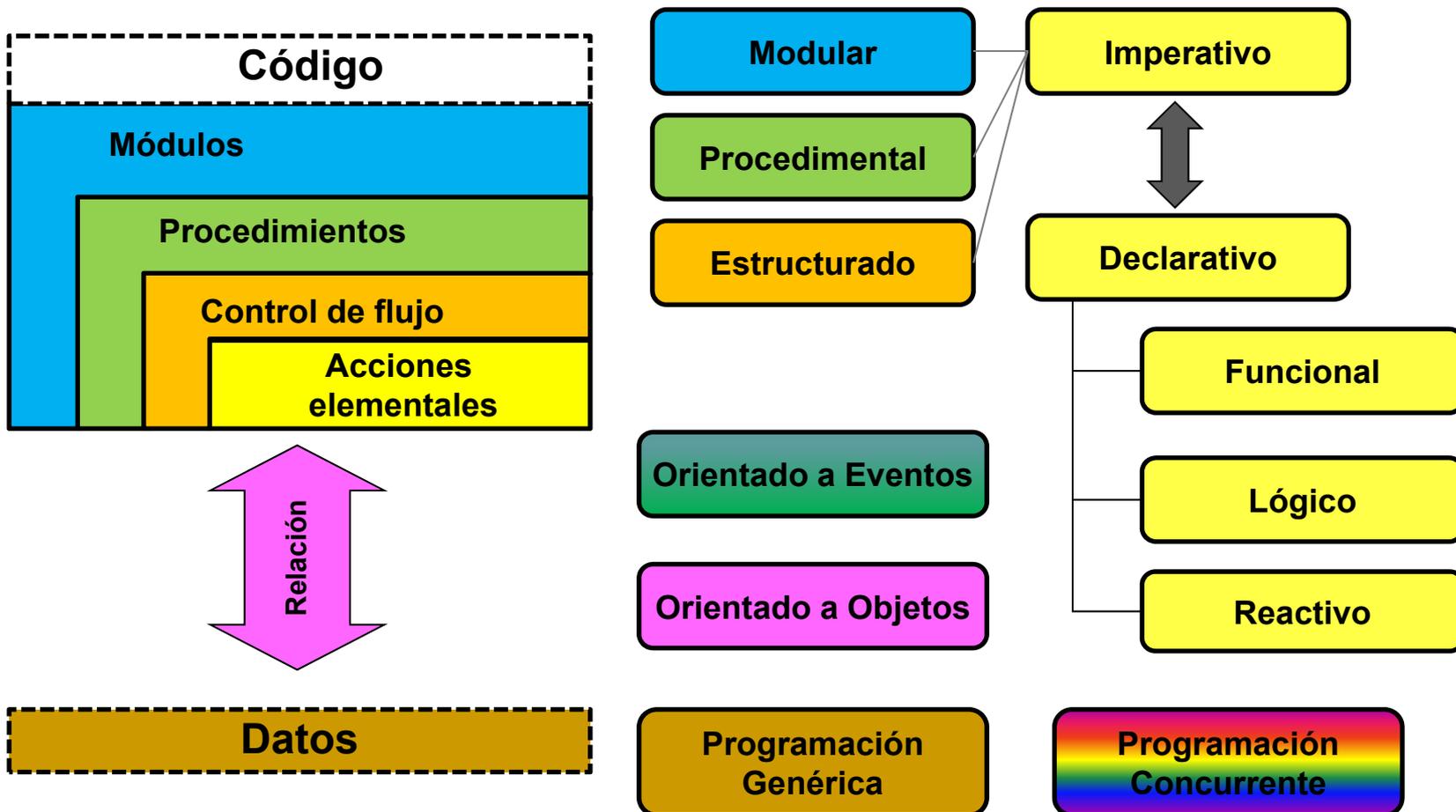


Tipos de paradigmas

- Los **paradigmas fundamentales** están basados en diferentes **modelos de cómputo** y por lo tanto afectan a las construcciones más básicas de un programa.
- La división principal reside en el enfoque **imperativo** (indicar el **cómo** se debe calcular) y el enfoque **declarativo** (indicar el **qué** se debe calcular).
 - El enfoque declarativo tiene varias ramas diferenciadas: el paradigma **funcional**, el paradigma **lógico**, la programación **reactiva** y los lenguajes **descriptivos**.
- Otros paradigmas se centran en la estructura y organización de los programas, y son compatibles con los fundamentales:
 - Ejemplos: Programación estructurada, modular, **orientada a objetos**, **orientada a eventos**, programación genérica.
- Por último, existen paradigmas asociados a la **conurrencia** y a los **sistemas de tipado**.



El zoo de los paradigmas





Paradigma Imperativo

- Describe **cómo** debe realizarse el cálculo, no el **porqué**.
- Un cómputo consiste en una serie de sentencias, ejecutadas según un control de flujo **explícito**, que **modifican el estado** del programa.
- Las variables son **celdas de memoria** que contienen datos (o referencias), pueden ser modificadas, y representan el **estado** del programa.
- La sentencia principal es la **asignación**.
- Es el estándar 'de facto'.
 - Asociados al paradigma imperativo se encuentran los paradigmas **procedural**, **modular**, y la programación **estructurada**.
 - El lenguaje representativo sería FORTRAN-77, junto con COBOL, BASIC, PASCAL, C, ADA.
 - También lo implementan Java, C++, C#, Eiffel, Python, ...



Paradigma Declarativo

- Describe **que** se debe calcular, sin explicitar el **cómo**.
- No existe un orden de evaluación prefijado.
- Las variables son **nombres** asociados a **definiciones**, y una vez instanciadas son **inmutables**.
- No existe sentencia de asignación.
- El control de flujo suele estar asociado a la composición funcional, la recursividad y/o técnicas de reescritura y unificación.
 - Existen distintos grados de **pureza** en las variantes del paradigma.
 - Las principales variantes son los paradigmas **funcional**, **lógico**, la programación **reactiva** y los lenguajes descriptivos.



Programación Funcional

- Basado en los modelos de cómputo **cálculo lambda** (Lisp, Scheme) y **lógica combinatoria** (familia ML, Haskell)
- Las funciones son elementos de **primer orden**
- Evaluación por **reducción funcional**. Técnicas: recursividad, parámetros acumuladores, CPS, Mónadas.
- Familia LISP (Common-Lisp, Scheme):
 - Basados en **s-expresiones**.
 - Tipado debil.
 - Meta-programación
- Familia ML (Miranda, Haskell, Scala):
 - Sistema estricto de tipos (**tipado algebraico**)
 - Concordancia de patrones.
 - Transparencia referencial
 - Evaluación perezosa (estruct. de datos infinitas)



Programación Lógica

- Basado en la **lógica de predicados** de primer orden
- Los programas se componen de hechos, predicados y relaciones.
- Evaluación basada en resolución SLD: unificación + backtracking.
- La ejecución consiste en la resolución de un problema de decisión, los resultados se obtienen mediante la instanciación de las variables libres.
- Lenguaje representativo: PROLOG

Programación Reactiva (Dataflow)



- Basado en la **teoría de grafos**.
- Un programa consiste en la especificación del flujo de datos entre operaciones.
- Las variables se encuentran **ligadas** a las operaciones que proporcionan sus valores. Un cambio de valor de una variable se **propaga** a todas las operaciones en que participa.
- Las hojas de cálculo se basan en este modelo.
- Lenguajes representativos: Simulink, Oz, Clojure.

```
A := 10
B := A + 1
print B → 11
A := 3
print B → 4
```

Ejemplo: Algoritmo de Euclides



- Cálculo del máximo común divisor
 - Primer algoritmo no trivial. Euclides, año 300 A.C.

$$\text{mcd}(a, b) = \max\{d : a \mid d, b \mid d\}$$

- donde $a \mid d$ significa que a es divisible exactamente por d :

$$a \mid d \Leftrightarrow \exists n : a = n \cdot d \qquad b \mid d \Leftrightarrow \exists m : b = m \cdot d$$

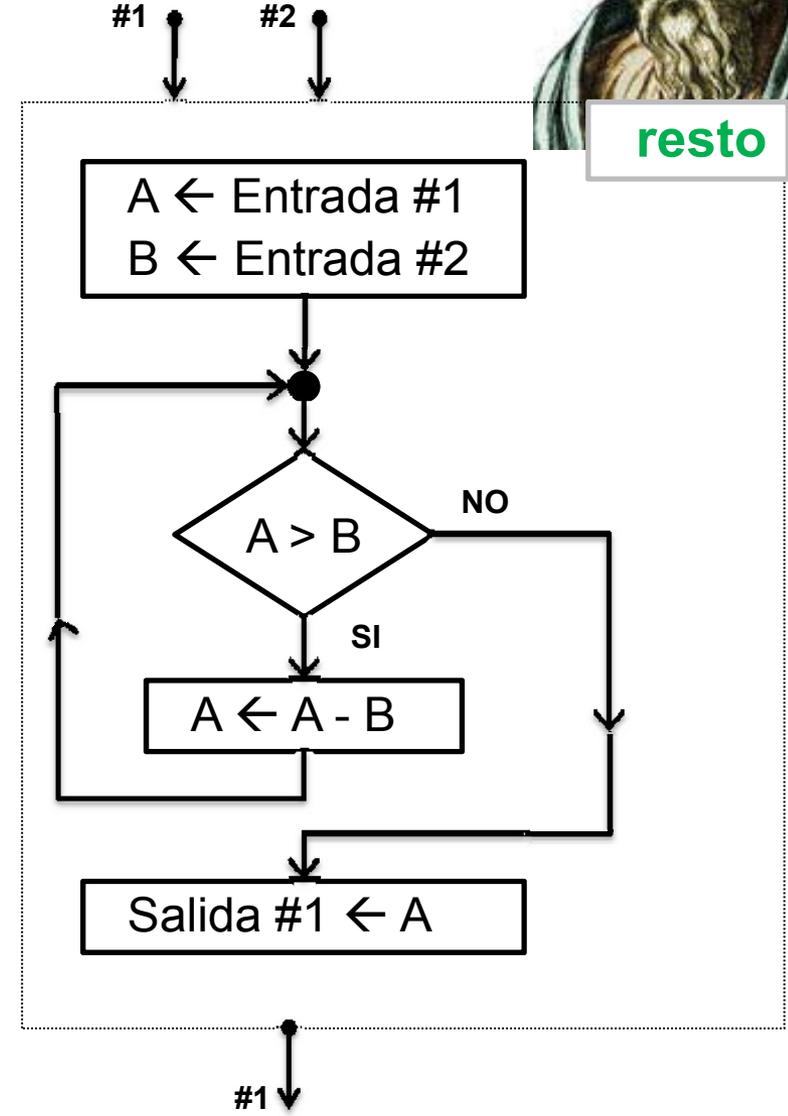
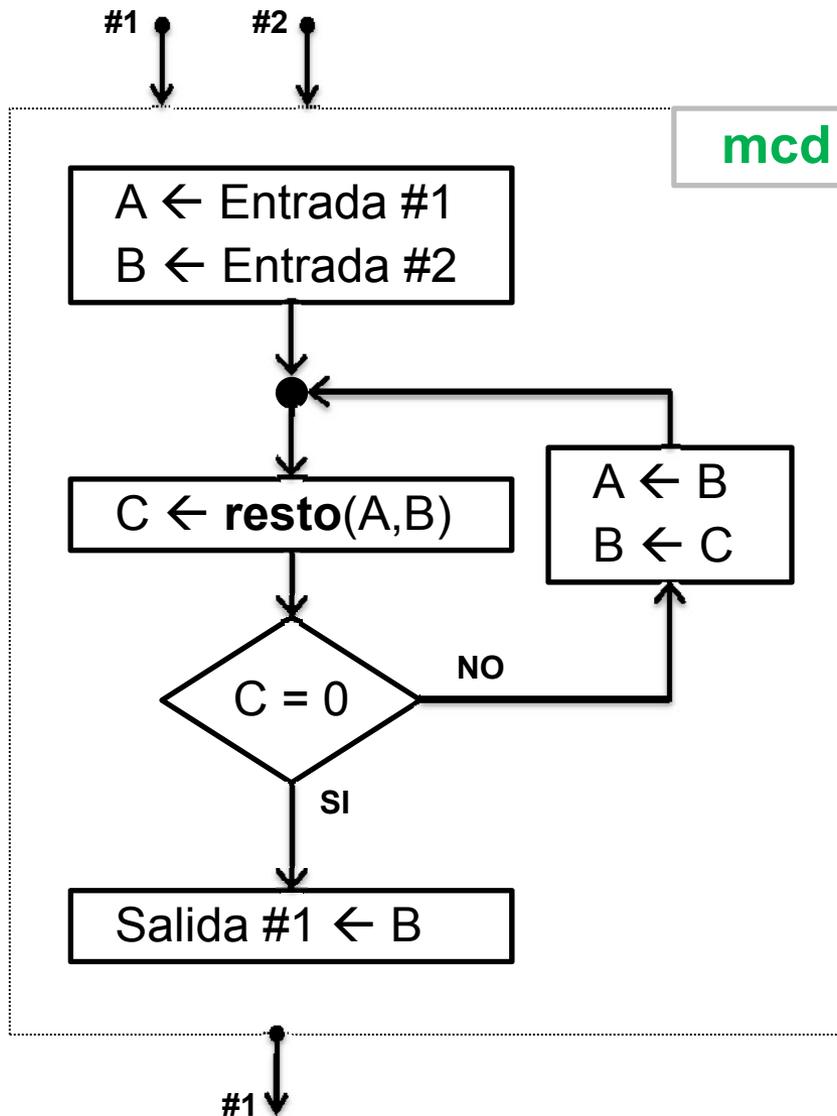
- si a y b son divisibles por d , y $a > b$, entonces:

$$a - b = (n - m) \cdot d \Rightarrow (a - b) \mid d$$

- y por lo tanto (restar sucesivamente b es equivalente a hallar el resto de dividir a por b):

$$a > b \Rightarrow \text{mcd}(a, b) = \text{mcd}(a - b, b) = \text{mcd}(a \bmod b, b)$$

Diagrama de flujo



FORTRAN-77



- Imperativo, procedural, no estructurado

```
FUNCTION MCD(NA, NB)
  IA = NA
  IB = NB
1  IF (IB.NE.0) THEN
    ITMP = IA
    IA = IB
    IB = MOD(ITMP, IB)
    GOTO 1
  END IF
  MCD = IA
  RETURN
END
```

Annotations:

- Paso por referencia (points to the function arguments NA, NB)
- Tipado implícito (points to the variable assignments IA = NA and IB = NB)
- Salto (points to the GOTO 1 statement)

PASCAL



- Imperativo, procedural, estructurado

```
function MCD(a,b: integer): integer;  
var c: integer;  
begin  
  while b <> 0 do  
  begin  
    c := a;  
    a := b;  
    b := c mod b  
  end;  
  MCD := b  
end;
```

Paso por valor

Tipado explícito

SCHEME, HASKELL, PROLOG



- Lenguajes funcionales y lógicos (recursividad)
 - Scheme

```
(define (mcd a b)
  (if (= b 0) a
    (mcd b (modulo a b))))
```

s-expresiones

- Haskell

```
mcd :: Int -> Int -> Int
mcd a 0 = a
mcd a b = mcd b (rem a b)
```

tipado estricto

concordancia
de patrones

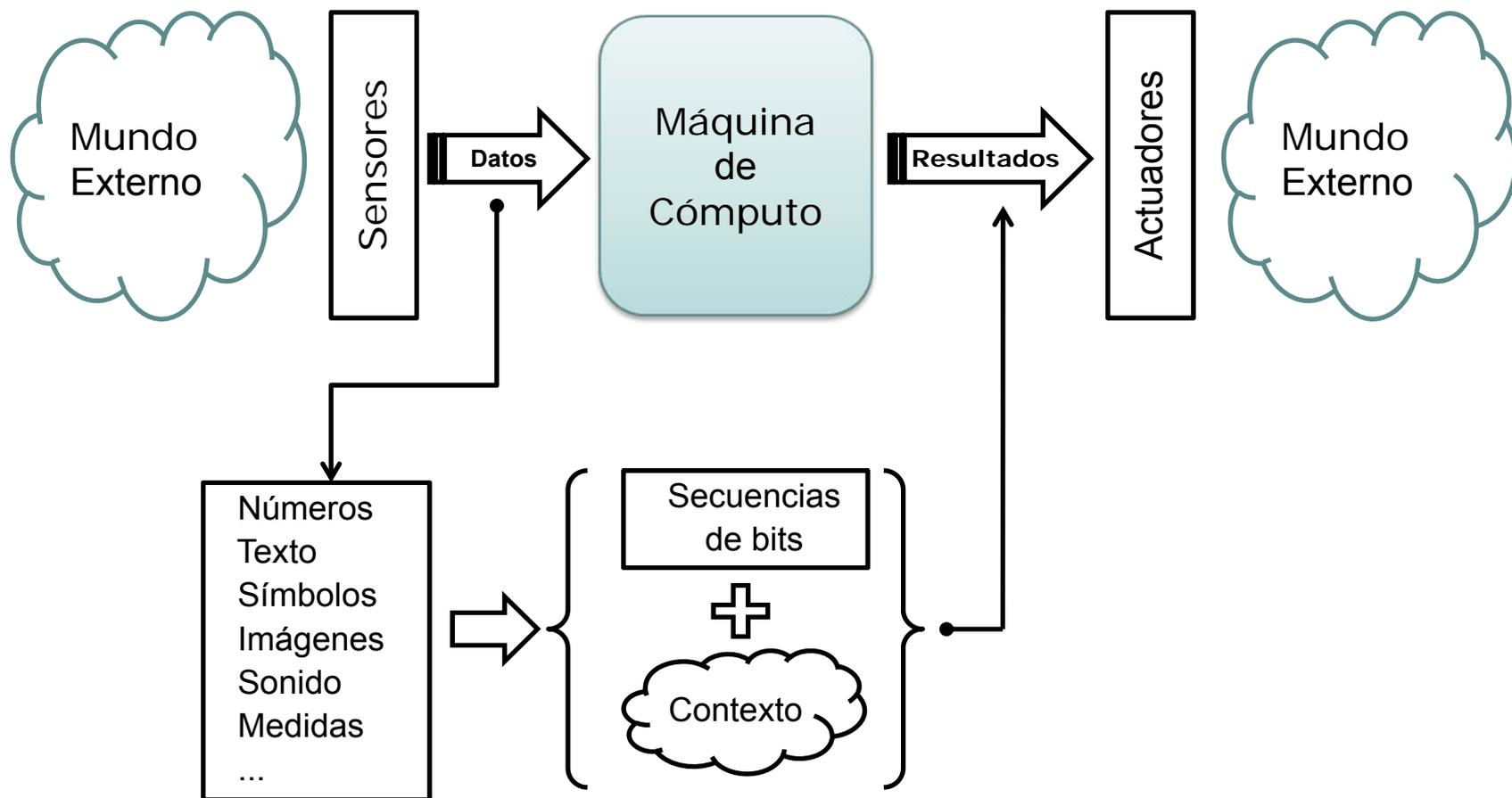
- Prolog

```
mcd(A,0,D) :- A = D.
mcd(A,B,D) :- B > 0, C is A mod B, mcd(B,C,D).
```

predicados, unificación



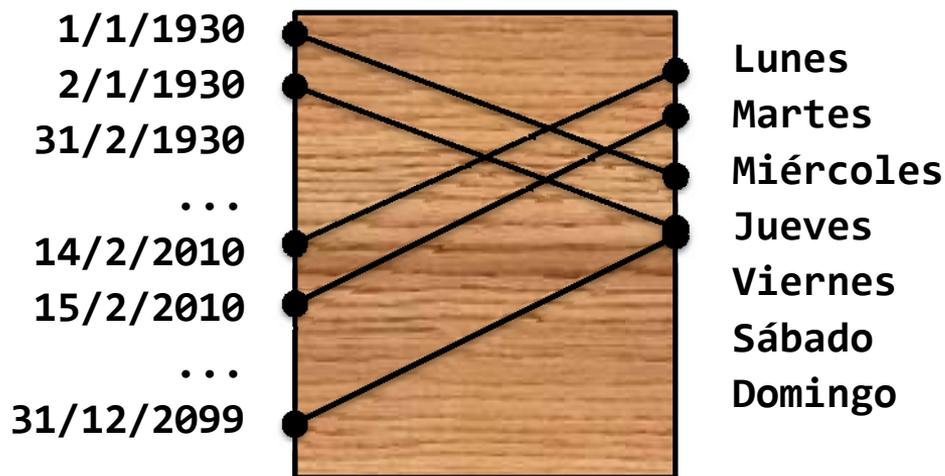
Modelos de Cómputo





Modelos de cómputo

- El concepto de cómputo puede modelizarse por el concepto matemático de **función**:
 - “Aplicación de un **dominio** de valores a un **rango** de resultados donde cada valor puede estar asociado como máximo a un resultado”
 - Usamos el modelo “caja de conexiones” para las funciones
 - Ejemplo: función que devuelve el día de la semana.



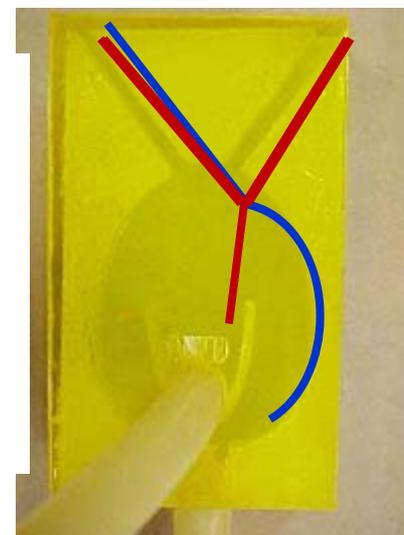
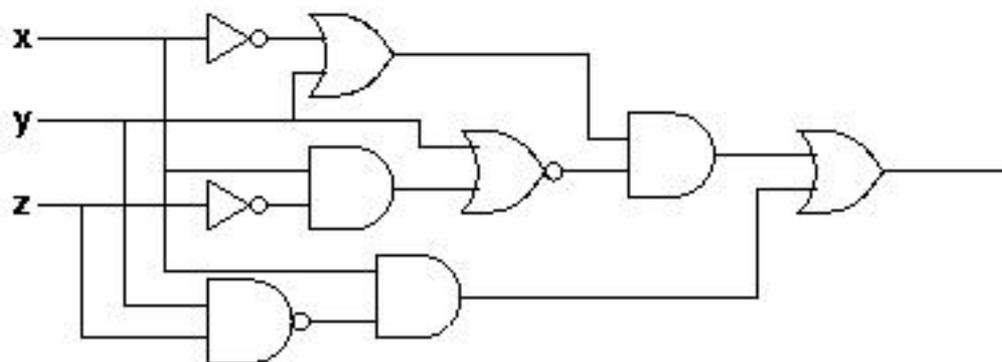
Máquinas y modelos de cómputo



- Jerarquía de niveles según capacidad expresiva y poder de cómputo :
 - Circuitos combinacionales
 - Máquinas de estado finito / Autómatas secuenciales
 - Máquinas de Turing / Máquinas de registros (RAM)
- Modelos formales
 - Funciones parciales recursivas
 - Cálculo lambda / Lógica combinatoria
 - Lógica de predicados + unificación
 - Sistemas de reescritura
- Arquitecturas
 - Modelo Von-Neumman
 - Modelo Harvard
 - Paralelismo

Circuitos combinatoriales

- Basados en la lógica booleana
- Las entradas y los resultados no se pueden secuenciar
- Conjunto minimalista de elementos: reles, puertas NAND, etc.



AND

XOR

Máquina de Antikythera



- La máquina calculadora (no trivial) más antigua: Año 150 A.C.

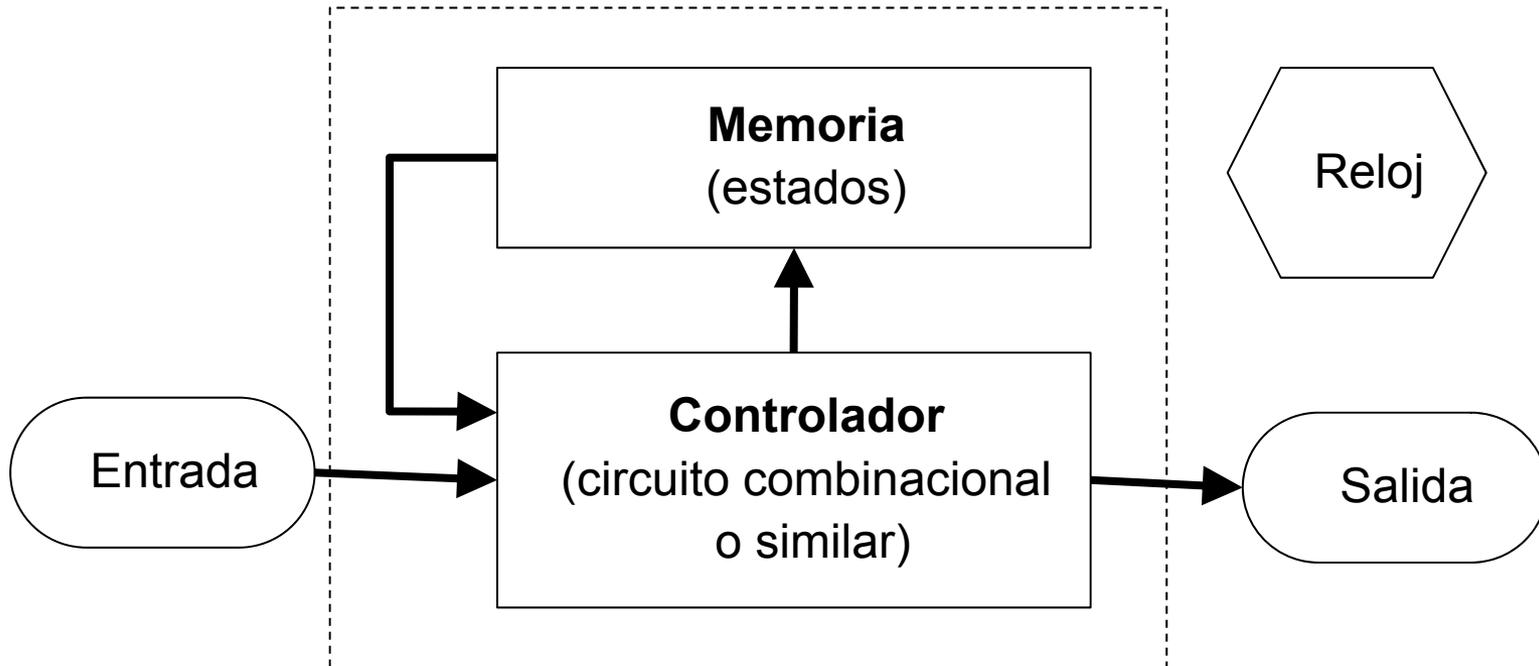


- Video: <http://www.youtube.com/watch?v=MqhuAnySPZ0>

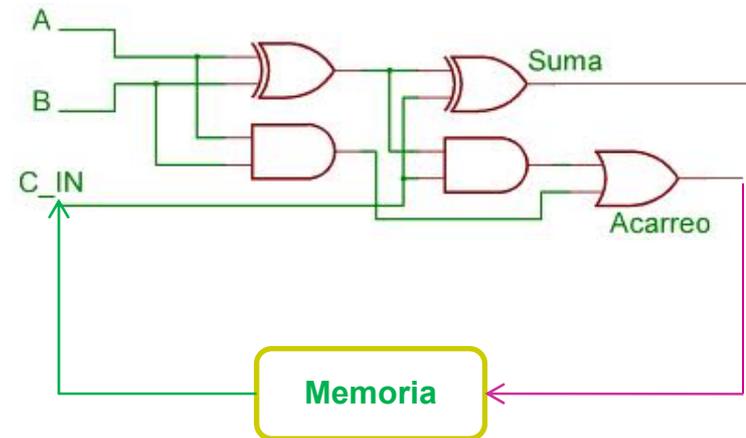
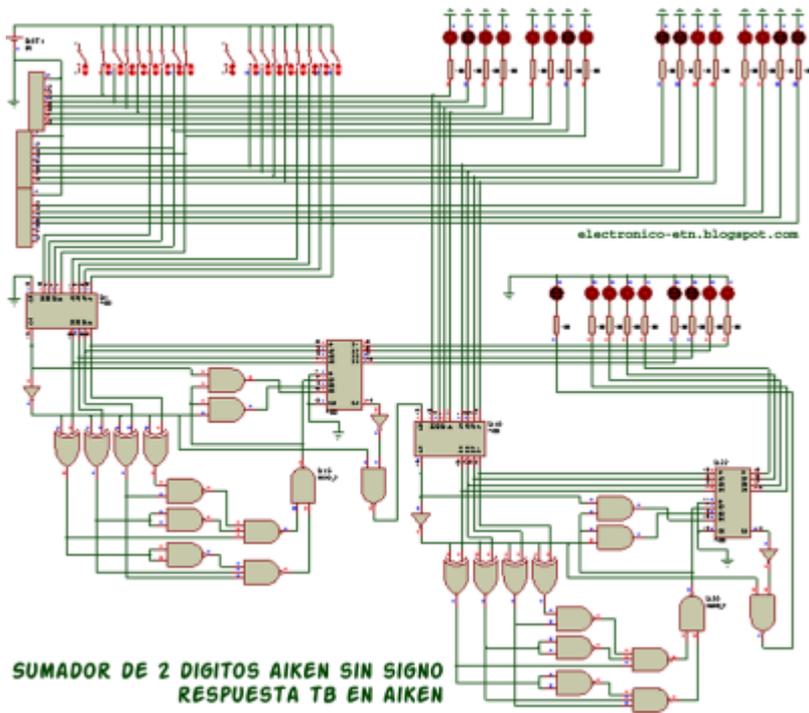
Máquinas secuenciales (maq. estado finito / autómatas)



- Circuito combinacional + memoria (estado) + reloj
- Se pueden secuenciar los datos de entrada y salida
- Los datos de entrada pueden controlar el flujo de ejecución



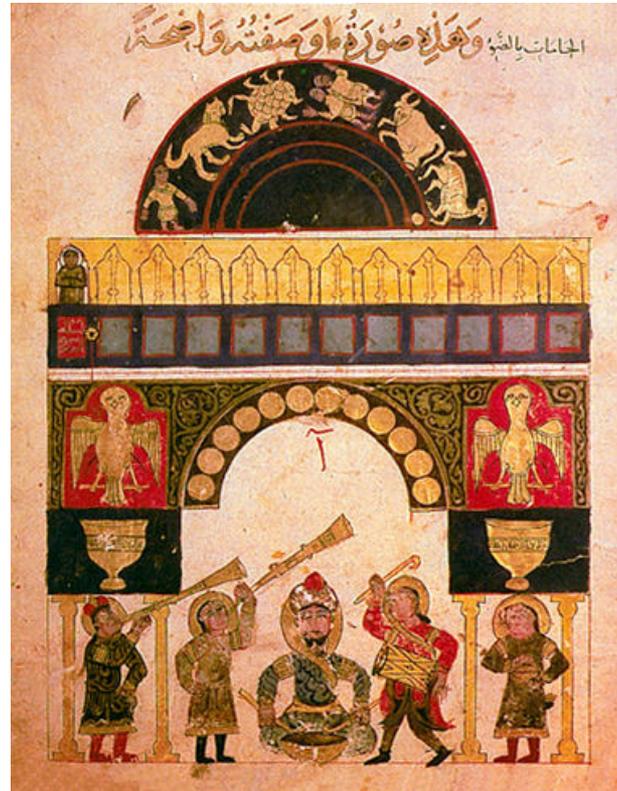
Sumador secuencial



Reloj del Castillo



- El autómata programable más antiguo: Al-Jazari, año 1206 D.C.

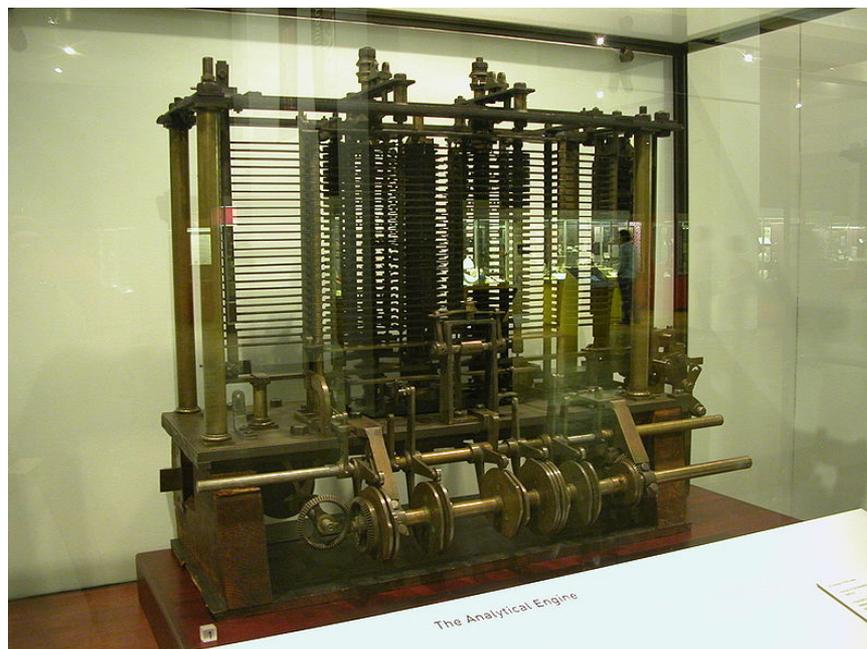


- Video: <http://www.youtube.com/watch?v=0pGXL5OKGqs>



Máquina Analítica

- La primera máquina computadora universal (si se hubiera construido)
- Charles Babbage, 1837
- Primer programa de la historia: Ada Lovelace

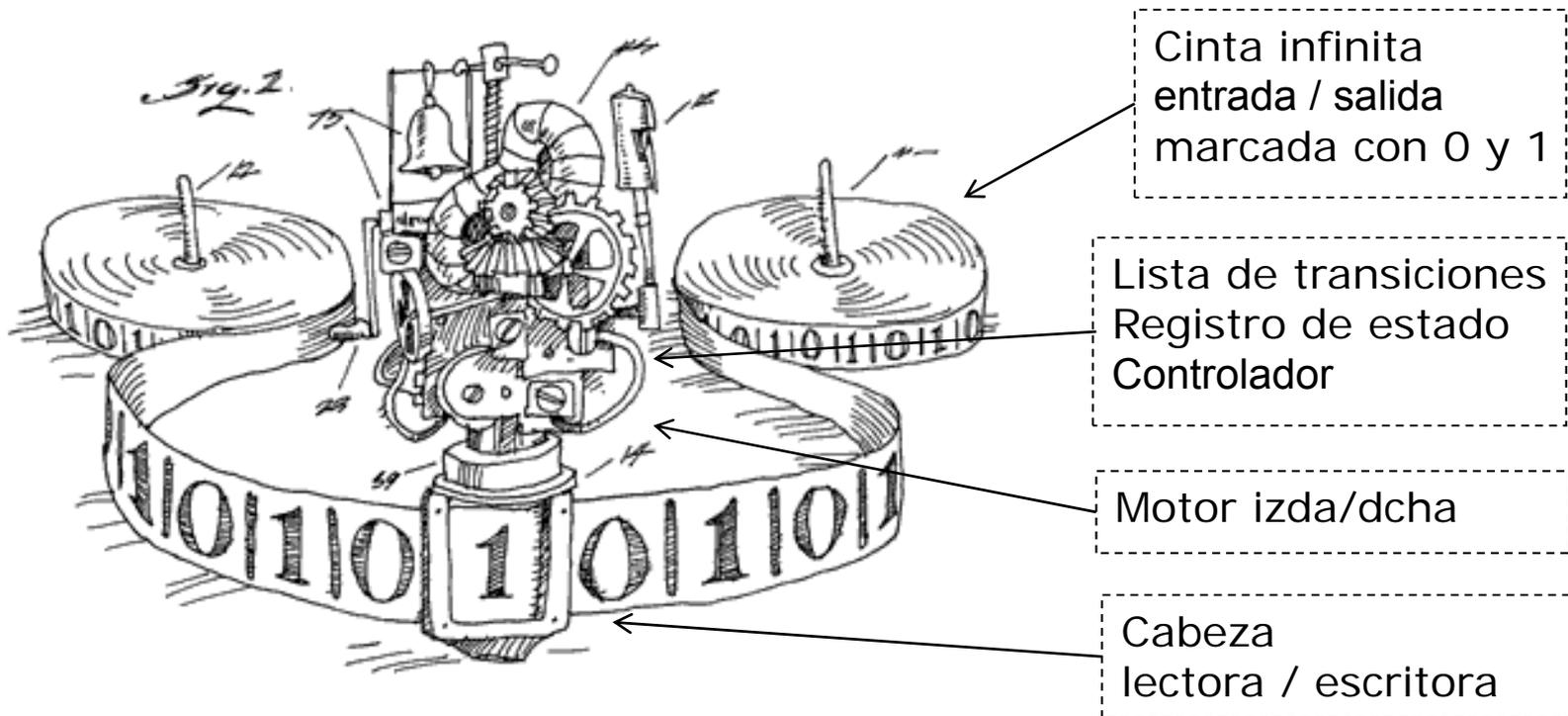


- Video: <http://www.youtube.com/watch?v=88GYbyMaaN8&NR=1>

Máquinas de Turing

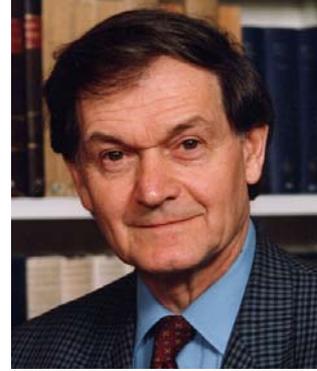


- Alan Turing, 1936 – Modelo abstracto de cómputo



- Existen muchas MT distintas, definidas por su lista de transiciones, cada una resuelve un problema particular.
- Video máquina real: <http://aturingmachine.com/>

Máquinas de Turing (versión de Penrose)



- Cinta de entrada:
 - Alfabeto de sólo dos símbolos: 0 (blanco) y 1 (punto)
 - Al comienzo la entrada se sitúa a la derecha de la cabeza
 - Al finalizar la salida se encuentra a la izquierda de la cabeza
- Controlador:
 - Cada máquina tiene n **estados** posibles (numerados $0..n-1$)
 - La máquina comienza siempre en el estado 0
 - Dispone de un único registro que almacena el **estado actual**
 - La **lista de transiciones** tiene n filas, una por cada estado, y dos columnas, una por cada valor posible de la celda actual (0 ó 1)
 - Cada transición indica lo siguiente:
 - **Nuevo estado** al que pasa la máquina
 - Símbolo (0 ó 1) que se **escribe** en la celda actual
 - **Movimiento** de la cabeza: **I** (izquierda), **D** (derecha), **S** (derecha y parada)



Máquinas de Turing

- Codificación **unaria**:

- La máquina recibe una lista de enteros positivos no nulos
- Cada número se separa del siguiente por el símbolo **0** (blanco)
- El valor del entero es el número de **1** consecutivos
- Ejemplo: Entradas (2,5,1):

..011011111010..

- Codificación **general**:

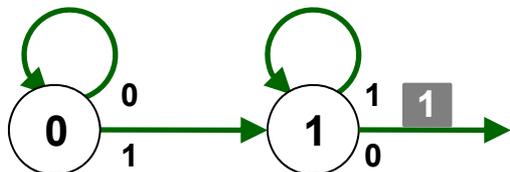
- Es un proceso de dos etapas de traducción
- La máquina recibe una secuencia de números enteros positivos y símbolos cualesquiera (un número finito de posibles símbolos).
- Los números se codifican en **binario** (números **0** y **1**)
- El resto de símbolos se indexan por números del **2** en adelante.
- Esta secuencia de números se convierte a **binario expandido**:

0 → **0**, **1** → **01**, **2** → **011**, **3** → **0111**, **4** → **01111**, ...



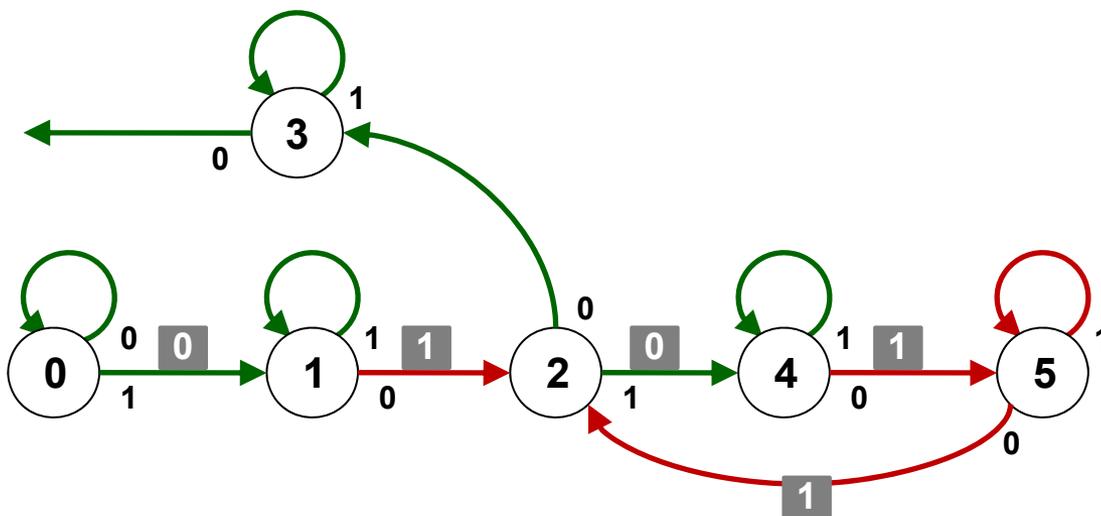
Máquinas de Turing – INC, DUP

- Incremento en uno (unaria)



	0	1
0	0 0 D	1 1 D
1	0 1 S	1 1 D

- Multiplicar por 2 (unaria)

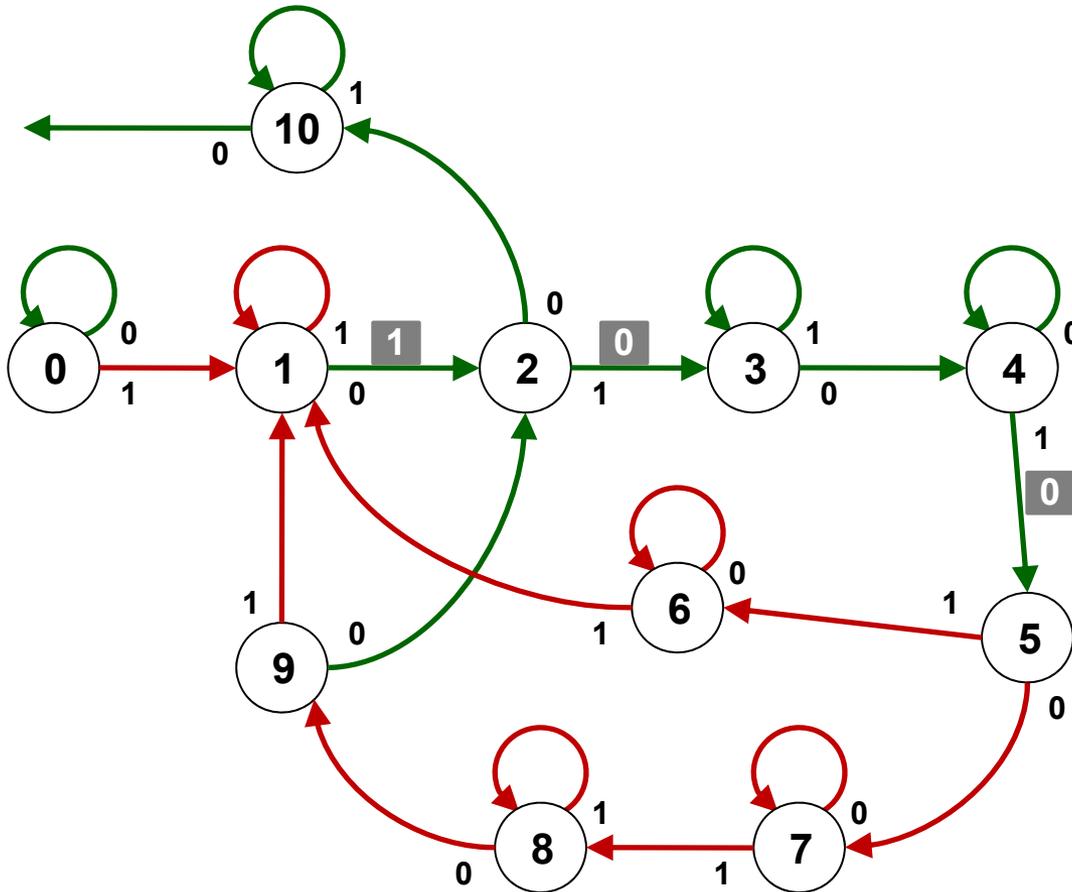


	0	1
0	0 0 D	1 0 D
1	2 1 I	1 1 D
2	3 0 D	4 0 D
3	0 1 S	3 1 D
4	5 1 I	4 1 D
5	2 1 I	5 1 I

Máquina de Turing - MCD



- Cálculo del máximo común divisor (unaria)

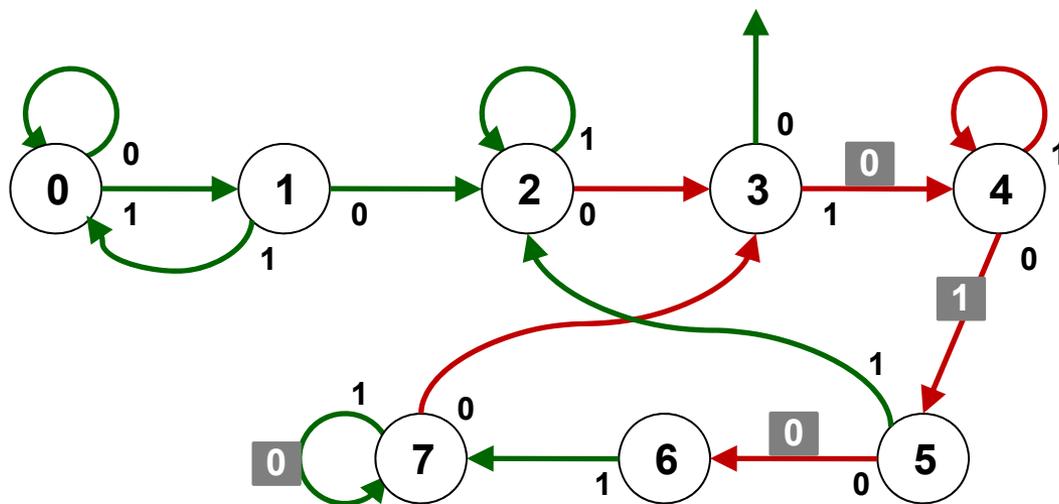


	0	1
0	0 0 D	1 1 I
1	2 1 D	1 1 I
2	10 0 D	3 0 D
3	4 0 D	3 1 D
4	4 0 D	5 0 D
5	7 0 I	6 1 I
6	6 0 I	1 1 I
7	7 0 I	8 1 I
8	9 0 I	8 1 I
9	2 0 D	1 1 I
10	0 0 S	10 1 D

Máquina de Turing – INC



- Incremento (general)



	0	1
0	0 0 D	1 1 D
1	0 0 D	2 1 D
2	3 0 I	2 1 D
3	0 1 S	4 0 I
4	5 1 I	4 1 I
5	6 0 D	2 1 D
6	0 0 D	7 1 D
7	3 1 D	7 0 D

- El algoritmo se basa en que para incrementar un número en binario basta con localizar el último **0** y cambiarlo por **1** y todos los siguientes **1** por **0**:

$$1010\mathbf{0}111 + 1 = 1010\mathbf{1}000$$



Máquina de Turing Universal

- Cada máquina de Turing realiza un determinado cómputo (resuelve un determinado problema)
- Cada máquina de Turing esta completamente determinada por su tabla de transiciones.
- Es posible codificar en binario la tabla de transiciones de una máquina de Turing:
 - Pasamos los estados a binario, elegimos la codificación para movimientos $D \rightarrow 2$, $I \rightarrow 3$, $S \rightarrow 4$
 - Para ahorrar espacio quitamos la primera transición y convertimos las transiciones $(0,0,x) \rightarrow x$, $(0,1,x) \rightarrow 1x$
 - Pasamos la secuencia a binario expandido y eliminamos el 110 final.
- ¡Cada máquina de Turing está representada por un número entero positivo!



Máquina de Turing Universal

- Ejemplo: Máquina INC unaria:

- Tabla de transiciones, en secuencia:

00D11D01S11D

- Quitando primera transición y convirtiendo 01S en 1S:

11214112

- Conviertiendo a binario expandido:

0101011010111101010110

- Quitando los tres dígitos finales y traduciendo a decimal:
- La máquina de Turing INC unaria es la 177.642-ava máquina de Turing



Máquina de Turing Universal

- Máquina de Turing Universal: Recibe como parámetro el número de **otra máquina** de Turing y una lista de parámetros.
- Devuelve como resultado el cálculo que hubiera realizado la otra máquina si se hubiera ejecutado con esos parámetros.
 - Sea **TU** la máquina universal, y **T_n** la máquina con número **n**:
- **$TU(n, p_1 \dots p_m) = T_n(p_1 \dots p_m)$**
- La máquina universal es capaz de **simular** cualquier otra máquina de Turing.
- La máquina universal tiene su propio número:

Máquina de Turing Universal



724485533533931757719839503961571123795236067255655963110814479660650505940424109
031048361363235936564444345838222688327876762655614469281411771501784255170755408
565768975334635694247848859704693472573998858228382779529468346052106116983594593
879188554632644092552550582055598945189071653741489603309675302043155362503498452
983232065158304766414213070881932971723415105698026273468642992183817215733348282
307345371342147505974034518437235959309064002432107734217885149276079759763441512
307958639635449226915947965461471134570014504816733756217257346452273105448298078
496512698878896456976090663420447798902191443793283001949357096392170390483327088
259620130177372720271862591991442827543742235135567513408422229988937441053430547
104436869587640517812801943753081387063994277282315642528923751456544389905278079
324114482614235728619311833261065612275553181020751108533763380603108236167504563
585216421486954234718742643754442879006248582709124042207653875426445413345174856
629157429990950262300973373813772416217274772361020678685400289356608569682262014
198248621698902609130940298570600174300670086896759034473417412787425581201549366
393899690581773859165405535670409282133222163141097871081459978669599704509681841
906299443656015145490488092208448003482249207730403043188429899393135266882349662
101947161910701461968523192847482034495897709553561107027581748733327296678998798
473284098190764851272631001740166787363477605857245036964434897992034489997455662
402937487668839751404451665707750060513883991668814072545544665222050724262392379
211525318162512536305093172863142200406457130527580230766518335199568913974813750
4926429605010013651980186945639498



Computabilidad

- **Algoritmo**: Procedimiento **sistemático** que permite resolver un problema en un número **finito** de pasos, cada uno de ellos especificado de manera **efectiva** y sin **ambigüedad**.
- **Función computable**: Aquella que puede ser calculada mediante un dispositivo **mecánico** dado un tiempo y espacio de almacenamiento **ilimitado** (pero finito)
- No importa la eficiencia, sino la posibilidad de ser calculada.
- ¿Existen funciones no computables?
- **Entscheidungsproblem**: Décima pregunta de Hilbert (Bolonia, 1928): ¿Existe un procedimiento mecánico (algorítmico) general para resolver toda cuestión matemática bien definida?



Computabilidad

- **Algoritmo**: Procedimiento **sistemático** que permite resolver un problema en un número **finito** de pasos, cada uno de ellos especificado de manera **efectiva** y sin **ambigüedad**.
- **Función computable**: Aquella que puede ser calculada mediante un dispositivo **mecánico** dado un tiempo y espacio de almacenamiento **ilimitado**.
- No importa la eficiencia, sino la posibilidad de ser calculada.
- **Entscheidungsproblem**: Décima pregunta de Hilbert (Bolonia, 1928): ¿Existe un procedimiento mecánico (algorítmico) general para resolver toda cuestión matemática bien definida?

NO
(Gödel, 1931)
(Turing, 1937)



Tesis Church-Turing

- Existen problemas bien definidos para los cuales no es posible encontrar un procedimiento mecánico que devuelva una solución en un tiempo finito.
 - **El problema de la detención**
 - **El problema del castor afanoso**
- **Tesis Church-Turing:** Toda función computable es calculable mediante una máquina de Turing.
 - Indemostrable, pero considerada cierta por la mayoría.
- Equivalencia entre distintos sistemas formales:
 - Máquina de Turing \leftrightarrow Cálculo lambda
 - Calculo lambda \leftrightarrow Funciones recursivas
 - etc.



¿Super-Turing?

- Posibilidades de superar al modelo de Turing:
 - Múltiples cintas
 - Cintas en 2D, 3D, nD
 - Controlador trabajando en paralelo con varias cintas
 - Acceso directo a posición en cinta (modelo RAM)
 - ...
- Todas tienen un poder **equivalente** al de una máquina normal (pueden ser simuladas).
- Las alternativas mejoran la **eficiencia**, pero no amplían el conjunto de lo que es computable.



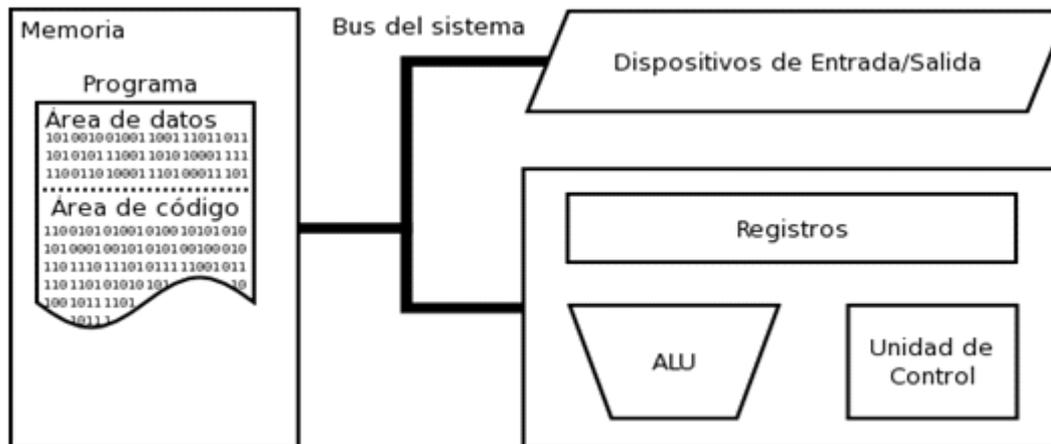
Máquinas de Registros (RAM)

- Derivadas del modelo de Turing, pero en vez de cinta secuencial con una memoria de **acceso directo**.
- El controlador dispone de un número finito de registros internos, que definen su estado.
- Un programa consiste en una serie de instrucciones, leídas de memoria, entre las cuales existen los tipos:
 - Copia de datos entre dirección de memoria y registros.
 - Operaciones aritméticas en registros
 - Salto condicional según valor de registro
 - Indirección (contenido de registros son direcciones de memoria)
- Es el modelo en que se basan la gran mayoría de computadoras.

Arquitectura Von-Neumman



- El programa y los datos se almacenan juntos en memoria.
- Existe un registro que indica la posición de memoria donde se encuentra la instrucción actual.
- **Arquitectura Harvard:** código y datos se almacenan en memorias separadas.





RAM minimalista

- Es posible tener una RAM con un sólo tipo de instrucción: **subleq a,b,c**
 - Esta instrucción resta el contenido de las posiciones de memoria **a** y **b**, almacena el resultado en **b**, y si es negativo salta a la instrucción situada en **c**.
 - Se necesita que una posición de memoria (**Z**) almacene **0**
- Cualquier otra instrucción puede sintetizarse a partir de **subleq**:

```
SALTO c ≡ subleq Z, Z, c
```

```
SUMA a, b ≡ subleq a, Z, c  
           c : subleq Z, b, d  
           d : subleq Z, Z, e  
           e : ...
```

```
MOV a, b ≡ subleq b, b, c  
           c : subleq a, Z, d  
           d : subleq Z, b, e  
           e : subleq Z, Z, f  
           f : ...
```

Funciones Primitivas Recursivas



- Kurt Gödel, 1931
- Modelo de cómputo formal, basado en la reducción al mínimo de los posibles elementos que se pueden usar para definir una función:
 - Se restringen las funciones a aquellas cuyos argumentos y **único** resultado son números naturales.
 - Se puede utilizar el valor constante **0 (función cero)**
 - Se puede usar **+1 (función sucesor)**
 - Se puede acceder a un argumento (**funciones proyectoras**)
- Composición:
 - El resultado de una función puede servir de argumento de otra

- Recursión primitiva:
$$h(0, \bar{x}) = f(\bar{x})$$
$$h(y + 1, \bar{x}) = g(y, h(y, \bar{x}), \bar{x})$$



Funciones Primitivas Recursivas

- Suma:

$$\text{suma}(0, b) = b$$

$$\text{suma}(a + 1, b) = \text{suma}(a, b) + 1$$

$$\text{suma}(0, b) = P_1^1(x)$$

$$\text{suma}(a + 1, b) = S(P_2^3(a, \text{suma}(a, b), b))$$

- Predecesor, Resta ($\text{resta}(a, b) = b - a$ si $b > a$, 0 si $b \leq a$)

$$\text{pred}(0) = 0$$

$$\text{pred}(a + 1) = a$$

$$\text{resta}(0, b) = b$$

$$\text{resta}(a + 1, b) = \text{pred}(\text{resta}(a, b))$$

- Condicional, Máximo Común Divisor:

$$\text{cond}(0, a, b) = a$$

$$\text{cond}(p + 1, a, b) = b$$

$$\begin{aligned} \text{mcd}(a, b) = \text{cond}(b, a, &\text{cond}(\text{resta}(a, b), \\ &\text{mcd}(\text{resta}(b, a), b), \\ &\text{mcd}(\text{resta}(a, b), a))) \end{aligned}$$



Funciones Primitivas Recursivas

- Son **finitas**: Su evaluación requiere un número finito de pasos.
- Son equivalentes a un lenguaje de programación donde los bucles tengan un número máximo de iteraciones.
 - Por ejemplo, Pascal sólo con bucles **for** (se permite sentencia **break** para salida anticipada) y sin llamadas recursivas.
- No pueden calcular **todas** las funciones computables.
- Para ello necesitan el **operador μ** → **Funciones Recursivas Generales**

$$\mu[f(y, \bar{x})] = \min\{y : f(y, \bar{x}) = 0\}$$

- Equivalentes a lenguajes con bucles tipo **while**.



Función de Ackermann

- Ejemplo de función computable **no primitiva recursiva**:

$$\phi(a, b, 0) = a + b$$

$$\phi(a, b, 1) = a \times b = \underbrace{a + a + \dots + a}_{b \text{ veces}}$$

$$\phi(a, b, 2) = a^b = a \uparrow b = \underbrace{a \times a \times \dots \times a}_{b \text{ veces}}$$

$$\phi(a, b, 3) = \underbrace{a^{a^{\dots^a}}}_{b \text{ veces}} = a \uparrow \uparrow b = \underbrace{a \uparrow (a \uparrow (\dots \uparrow a))}_{b \text{ veces}}$$

$$\phi(a, b, 4) = a \uparrow \uparrow \uparrow b = \underbrace{a \uparrow \uparrow (a \uparrow \uparrow (\dots \uparrow \uparrow a))}_{b \text{ veces}}$$

$$\phi(a, b, n) = \underbrace{a \uparrow \dots \uparrow}_{n \text{ veces}} b = \underbrace{a \uparrow \dots \uparrow (a \uparrow \dots \uparrow (\dots a \uparrow \dots \uparrow))}_{b \text{ veces}}_{n-1 \text{ veces} \quad n-1 \text{ veces} \quad n-1 \text{ veces}}$$



Cálculo lambda

- Alonzo Church, 1936
- El “lenguaje de programación” más sencillo (salvo quizás la lógica combinatoria)
- Simplificación extrema del cálculo:
 - No importa el nombre de las funciones ni de los argumentos:
 $f(x,y) = x^2 + y^2$ y $g(a,b) = a^2 + b^2$ son la misma función.

$$(x, y) \rightarrow x^2 + y^2$$

- Toda función de más de un argumento se puede considerar como una función de **un solo** argumento que devuelve no un valor sino **una función: Currificación**

$$x \rightarrow y \rightarrow x^2 + y^2$$

- No se necesitan números: Todo puede ser representado únicamente mediante **funciones**.



Cálculo lambda - Notación

- Una expresión lambda puede ser:
 - Una **variable** ($a, b, c \dots$)
 - Una **abstracción**: $\lambda x. t$ (donde x es una variable y t es una expresión lambda)
 - Una **aplicación**: $f g$ (donde f y g son expresiones lambda)
- Convenciones:
 - Las variables representan funciones.
 - Se pueden usar paréntesis para indicar el orden de evaluación.
 - Las aplicaciones son asociativas hacia la izquierda: $f g h = (f g) h$
 - Las abstracciones se extienden todo lo posible hacia la derecha
 - Dentro del término de una abstracción, la variable de la abstracción se denomina **ligada** (el resto son variables **libres**).
 - Las abstracciones se pueden **contraer**: $\lambda x. \lambda y. t \equiv \lambda x y. t$



Cálculo lambda - Reducciones

- Las operaciones que permiten manipular expresiones lambda son:
 - La **α -reducción** (renombrado): Es posible cambiar el nombre de las variables ligadas.

$$\lambda x. x y \equiv \lambda a. a y$$

- La **β -reducción**: Al **aplicar** una abstracción a otra expresión, podemos sustituir la expresión por el término de la abstracción donde se han **sustituido** todas las apariciones de la variable ligada por la expresión aplicada:

$$(\lambda x. x (y x))(z w) \equiv (z w) (y (z w))$$

- La **η -reducción**: Si el término de una abstracción es una aplicación donde en la primera expresión no aparece la variable ligada y la segunda expresión es la variable, se puede sustituir la abstracción por la expresión:

$$\lambda x. f x \equiv f$$



Cálculo lambda - Representación

- Representación de los números naturales, incremento, suma, producto, predecesor, resta:

$$\mathbf{0} \equiv \lambda f x . x$$

$$\mathbf{1} \equiv \lambda f x . f x$$

$$\mathbf{2} \equiv \lambda f x . f (f x)$$

$$\mathbf{3} \equiv \lambda f x . f (f (f x))$$

$$\mathbf{4} \equiv \lambda f x . f (f (f (f x)))$$

$$\mathbf{Succ} \equiv \lambda n f x . f (n f x)$$

$$\mathbf{Sum} \equiv \lambda m n . m \mathbf{Succ} n$$

$$\mathbf{Mul} \equiv \lambda m n . m (\mathbf{Sum} n) \mathbf{0}$$

$$\mathbf{Pred} \equiv \lambda n f x . n (\lambda g h . h (g f)) (\lambda n . x) (\lambda n . n)$$

$$\mathbf{Sub} \equiv \lambda n m . n \mathbf{Pred} m$$



Cálculo lambda - Representación

- Representación de los valores lógicos, condicional, test si valor nulo, test menor o igual:

$$\mathbf{T} \equiv \lambda x y . x$$

$$\mathbf{F} \equiv \lambda x y . y$$

$$\mathbf{If} \equiv \lambda p a b . p a b$$

$$\mathbf{Is0} \equiv \lambda n . n (\lambda x . \mathbf{F}) \mathbf{T}$$

$$\mathbf{Leq} \equiv \lambda n m . \mathbf{Is0} (\mathbf{Sub} n m)$$

- Recursividad (combinador Y):

$$\mathbf{Y} \equiv \lambda g . (\lambda x . g (x x)) (\lambda x . g (x x))$$

$$\mathbf{Y} f = (\lambda x . f (x x)) (\lambda x . f (x x))$$

$$= f ((\lambda x . f (x x)) (\lambda x . f (x x)))$$

$$= f (\mathbf{Y} f)$$



Cálculo lambda - MCD

- El cálculo del máximo común divisor se puede expresar:

$$\text{Mcd}\theta \equiv \lambda r a b . \text{If } (\text{Is}\theta b) a (\text{If } (\text{Leq } b a) \\ (r a (\text{Sub } a b)) (r b (\text{Sub } b a)))$$

$$\text{Mcd} \equiv \lambda a b . \text{Mcd}\theta (Y \text{Mcd}\theta) b a$$

- Expandiendo las definiciones:

$$\lambda a b . (\lambda r c d . (\lambda p e f . p e f) ((\lambda n . n (\lambda x g y . y) (\lambda x y . x)) d) c ((\lambda p h i . p h i) ((\lambda n m . (\lambda j . j (\lambda x k y . y) (\lambda x y . x)) ((\lambda l o . l (\lambda p f x . p (\lambda g h . h (g f)) (\lambda q . x) (\lambda s . s)) o) n m)) d c) (r c ((\lambda n m . n (\lambda t f x . t (\lambda g h . h (g f)) (\lambda u . x) (\lambda v . v)) m) c d)) (r d ((\lambda n m . n (\lambda w f x . w (\lambda g h . h (g f)) (\lambda y . x) (\lambda z . z)) m) d c)))) ((\lambda g . (\lambda x . g (x x)) (\lambda x . g (x x))) (\lambda r a' b' . (\lambda p c' d' . p c' d') ((\lambda n . n (\lambda x e' y . y) (\lambda x y . x)) b') a' ((\lambda p f' g' . p f' g') ((\lambda n m . (\lambda h' . h' (\lambda x i' y . y) (\lambda x y . x)) ((\lambda j' k' . j' (\lambda l' f x . l' (\lambda g h . h (g f)) (\lambda m' . x) (\lambda n' . n')) k') n m)) b' a') (r a' ((\lambda n m . n (\lambda o' f x . o' (\lambda g h . h (g f)) (\lambda p' . x) (\lambda q' . q')) m) a' b')) (r b' ((\lambda n m . n (\lambda r' f x . r' (\lambda g h . h (g f)) (\lambda s' . x) (\lambda t' . t')) m) b' a'))))) b a$$



Modelos de Cómputo

- Las máquinas de Turing, y sus extensiones en las máquinas RAM sirven de inspiración al **paradigma imperativo**:
 - Un cómputo es una secuencia de operaciones..
 - ..que **modifican el estado** del programa (registros)..
 - ..y cuyos resultados **determinan la secuencia** de ejecución.
- El cálculo lambda (y su variante la lógica combinatoria) sirve de inspiración al **paradigma funcional**:
 - Un cómputo consiste en una **expresión** que puede ser transformada en otras mediante **reglas de reescritura**.
 - El orden de evaluación es **irrelevante**.
- Las máquinas de Turing, el cálculo lambda y las funciones recursivas son **equivalentes**.



Lenguajes de Programación

- Lenguaje artificial diseñado para expresar cálculos que pueden ser llevados a cabo por una máquina.
 - Basado en un **modelo de cómputo** (que puede o no coincidir con el de la máquina en que se va a ejecutar)
 - Define un **nivel de abstracción** más elevado (más cercano al programador)
 - Debe traducirse a un código que pueda entender el procesador: el **código máquina**.
- Modos de traducción:
 - Lenguaje Compilado
 - Lenguaje Interpretado (Entorno interactivo)
 - Lenguaje traducido a Código Intermedio (Java → Bytecodes, .NET → IDL)



Estrategias de traducción

- Código compilado:

Programa

```

22
23 function compare_name(sequence a, sequence b)
24 -- Compare two sequences (records) according to NAME.
25   return compare(a[NAME], b[NAME])
26 end function
27
28 function compare_pop(sequence a, sequence b)
29 -- Compare two sequences (records) according to POPULATION.
30 -- Note: comparing b vs. a, rather than a vs. b, makes
31 -- the bigger population come first.
32   return compare(b[POPULATION], a[POPULATION])
33 end function
34
35 sequence sorted_by_pop, sorted_by_name
36 integer by_pop, by_name
37
38 by_pop = routine_id('compare_pop')
39 by_name = routine_id('compare_name')
40
41 sorted_by_pop = custom_sort(by_pop, statistics)
42 sorted_by_name = custom_sort(by_name, statistics)
43
44 puts(1, "sorted by population(1) sorted by name(1)")
45 for i = 1 to length(sorted_by_pop) do
46   printf(1, "%12s %0.1f(1)%12s %0.1f(1)",
47     sorted_by_pop[i] & sorted_by_name[i])
48 end for
49
50

```

Módulos

```

44 puts(1, "sorted by population(1) sorted by name(1)")
45 for i = 1 to length(sorted_by_pop) do
46   printf(1, "%12s %0.1f(1)%12s %0.1f(1)",
47     sorted_by_pop[i] & sorted_by_name[i])
48 end for
49

```

```

44 puts(1, "sorted by population(1) sorted by name(1)")
45 for i = 1 to length(sorted_by_pop) do
46   printf(1, "%12s %0.1f(1)%12s %0.1f(1)",
47     sorted_by_pop[i] & sorted_by_name[i])
48 end for
49

```

Código Máquina

```

C049: C0 4C 2B C0 AD 00 DC C9 8D
C048: 6F D0 E0 AD 83 C1 C9 05 2B
C059: F0 D9 EE 83 C1 A9 01 8D 87
C058: FD C8 AE 83 C1 BD 69 C1 FB
C069: AA A9 BA 9D 00 D0 A9 86 0E
C068: 9D 01 D9 A9 E3 8D FF 07 F9
C079: AE 83 C1 AD 15 D0 5D 6F C4
C078: C1 8D 15 D0 A9 01 8D FC E2
C089: C8 9D 75 C1 4C 2B C0 A2 F8
C088: 00 BD CF C4 9D 83 06 A9 AB
C099: 01 9D 83 DA E8 E0 21 D0 49
C098: F0 60 60 EE FA C8 AD FA A5
C0A9: C8 C9 02 D0 F5 A9 00 8D 33
C0A8: FA C8 AD FC C8 F0 25 AE A4
C0B9: 83 C1 BD 69 C1 AA DE 01 69
C0B8: D0 FE 00 D0 FE 00 D0 EE 18
C0C9: FB C8 AD FB C8 C9 06 D0 98
C0C8: 98 A9 00 8D FC C8 8D FB 57
C0D9: C8 4C 18 C1 AE 83 C1 BD 71
C0D8: 69 C1 AA DE 01 D0 DE 00 3E
C0E9: D0 DE 00 D0 EE FB C8 AD C2
C0E8: FB C8 C9 06 D0 2A A9 00 22
C0F9: 8D FB C8 8D FD C8 AE 83 C9
C0F8: C1 A9 01 9D 7B C1 A9 E0 CA
C109: 8D FF 07 AD 7C 05 8D 81 D2
C108: C1 20 84 C1 AD 20 89 8D 15
C119: F8 89 AD 21 89 8D F9 89 FB

```

Compilación

Ejecución

Entorno (SO)

Librerías estáticas

```

C0E8: FB C8 C9 06 D0 2A A9 00 22
C0F0: 8D FB C8 8D FD C8 AE 83 C9
C0F8: C1 A9 01 9D 7B C1 A9 E0 CA
C109: 8D FF 07 AD 7C 05 8D 81 D2
C108: C1 20 84 C1 AD 20 89 8D 15
C110: F8 89 AD 21 89 8D F9 89 FB

```

Librerías dinámicas

```

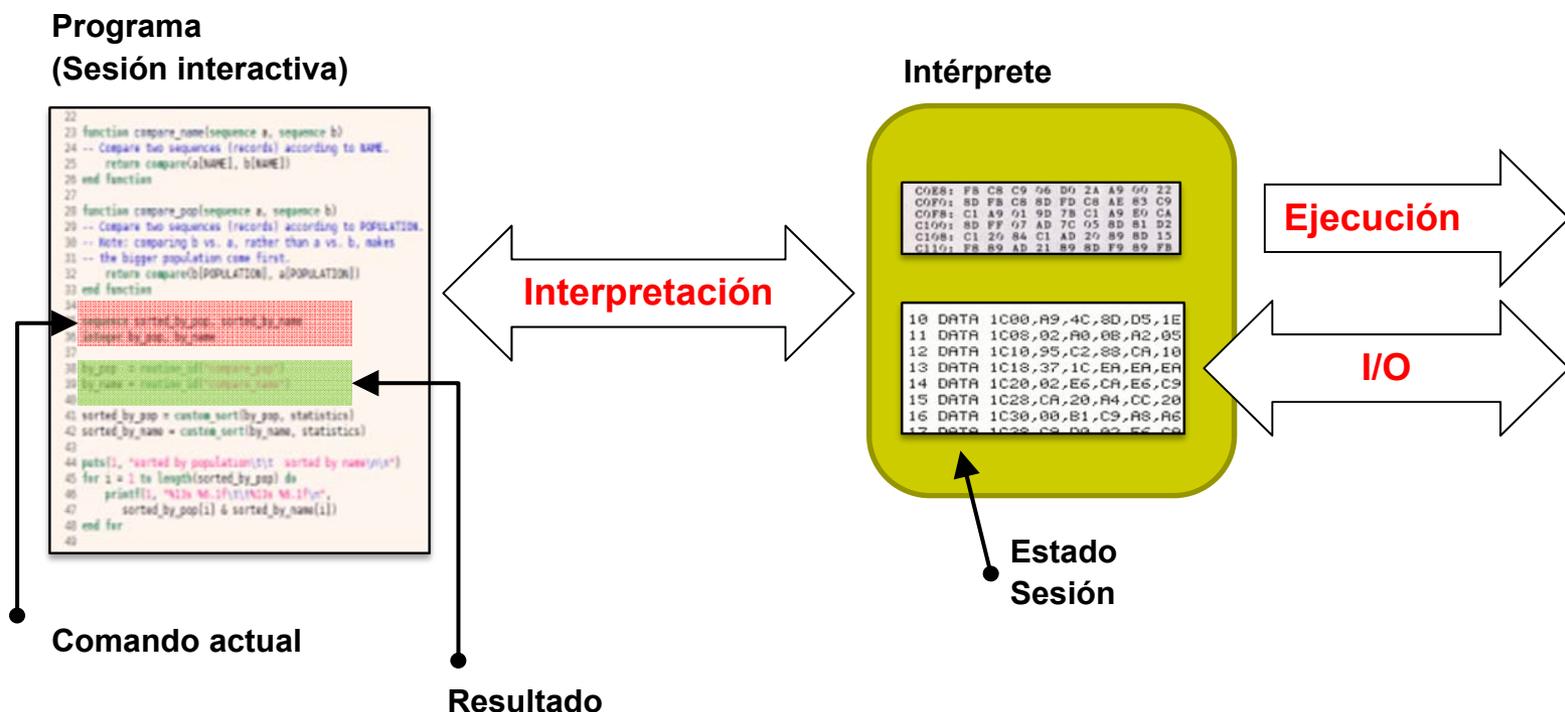
C0E8: FB C8 C9 06 D0 2A A9 00 22
C0F0: 8D FB C8 8D FD C8 AE 83 C9
C0F8: C1 A9 01 9D 7B C1 A9 E0 CA
C109: 8D FF 07 AD 7C 05 8D 81 D2
C108: C1 20 84 C1 AD 20 89 8D 15
C110: F8 89 AD 21 89 8D F9 89 FB

```



Estrategias de traducción

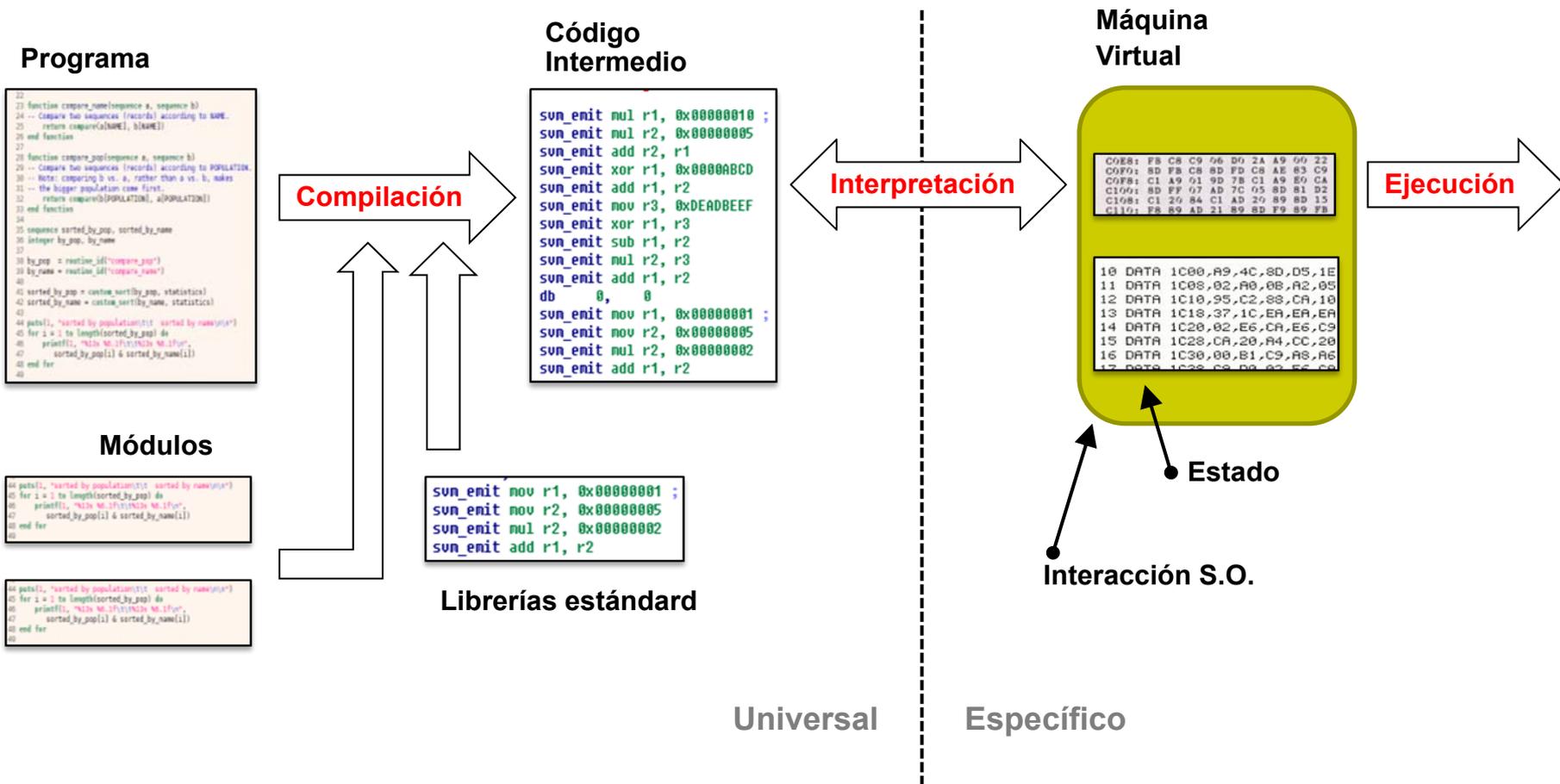
- Código interpretado:





Estrategias de traducción

- Código intermedio: (Pascal IDL, Java, .NET)



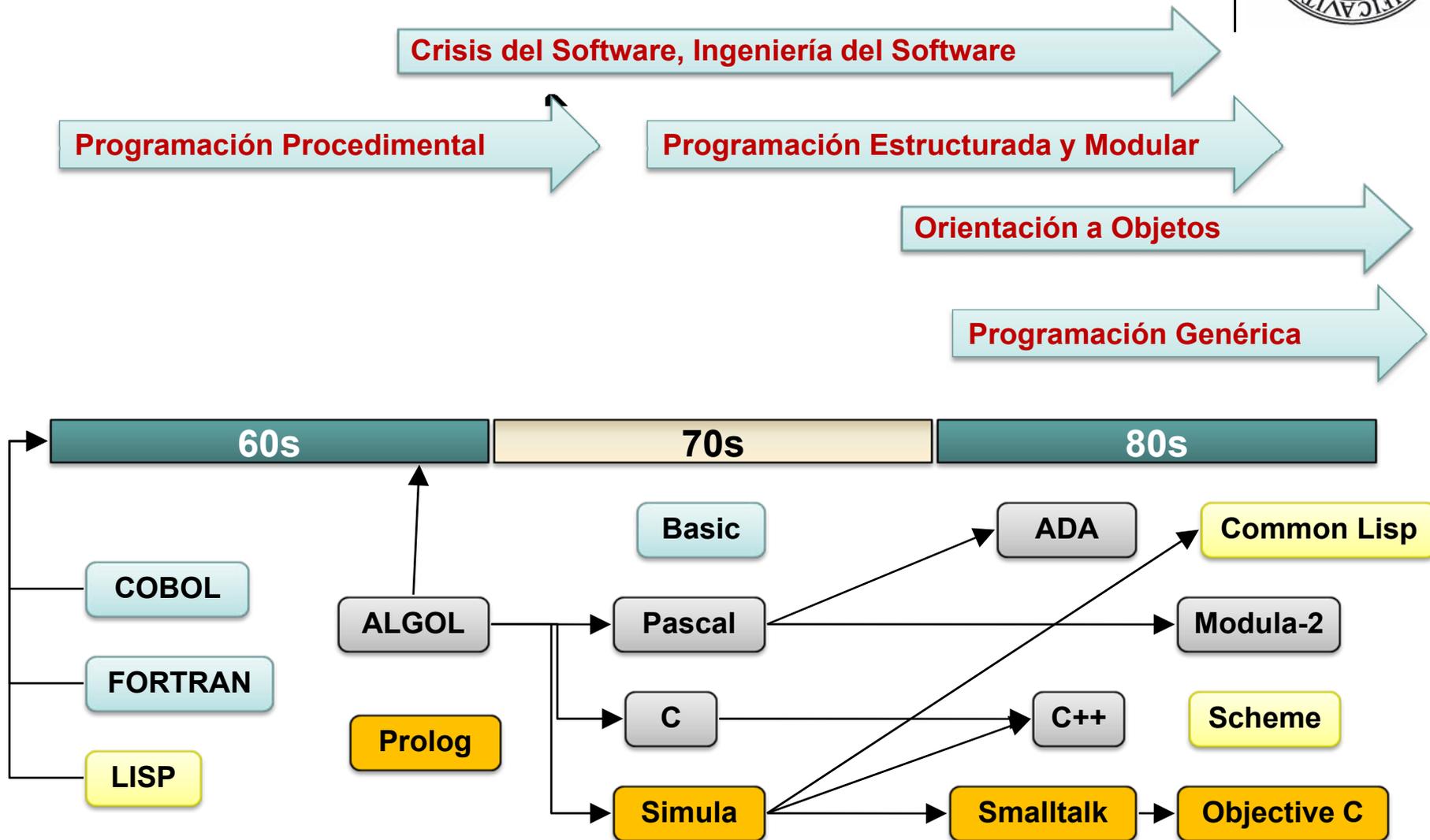
Generaciones



Generación	Lenguajes	Hardware	Movimientos
Primera (1945-55)	Código Máquina	Relés, Válvulas de vacío	
Segunda (1955-68)	FORTRAN COBOL LISP	Transistores, Memorias de ferrita	Prog. Estructurada y Modular Proceso por Lotes
Tercera (1968-1980)	ALGOL PASCAL C BASIC ADA	Circuitos integrados, Memorias de transistores	Ingeniería de Software Orientación a Objetos Bases de Datos
Cuarta (1980-)	C++ JAVA HASKELL PYTHON	VLSI MultiCore Flash	Comp. Distribuida Interfaces Gráficas Multimedia Internet

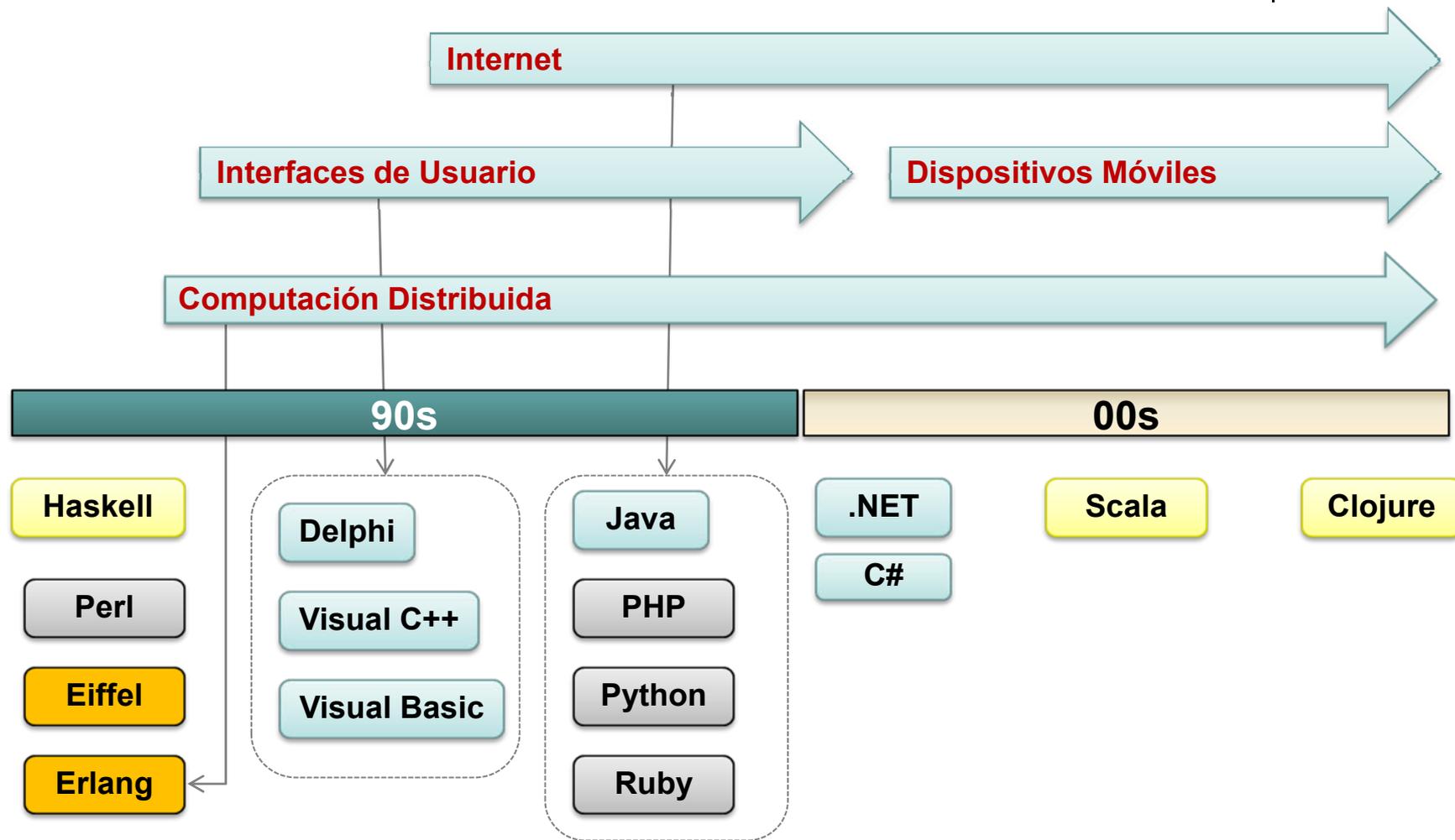


Linea del Tiempo



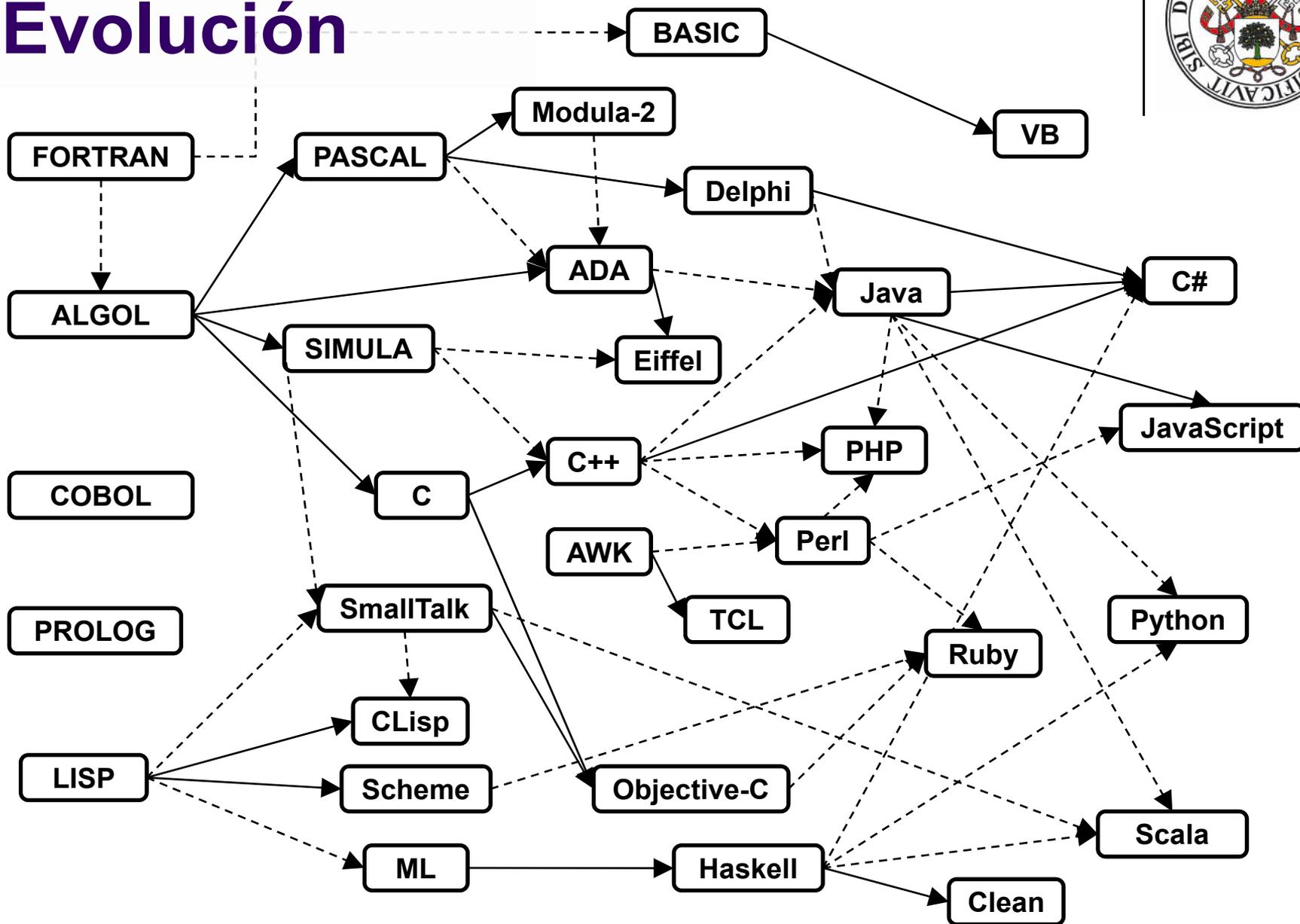


Linea del Tiempo



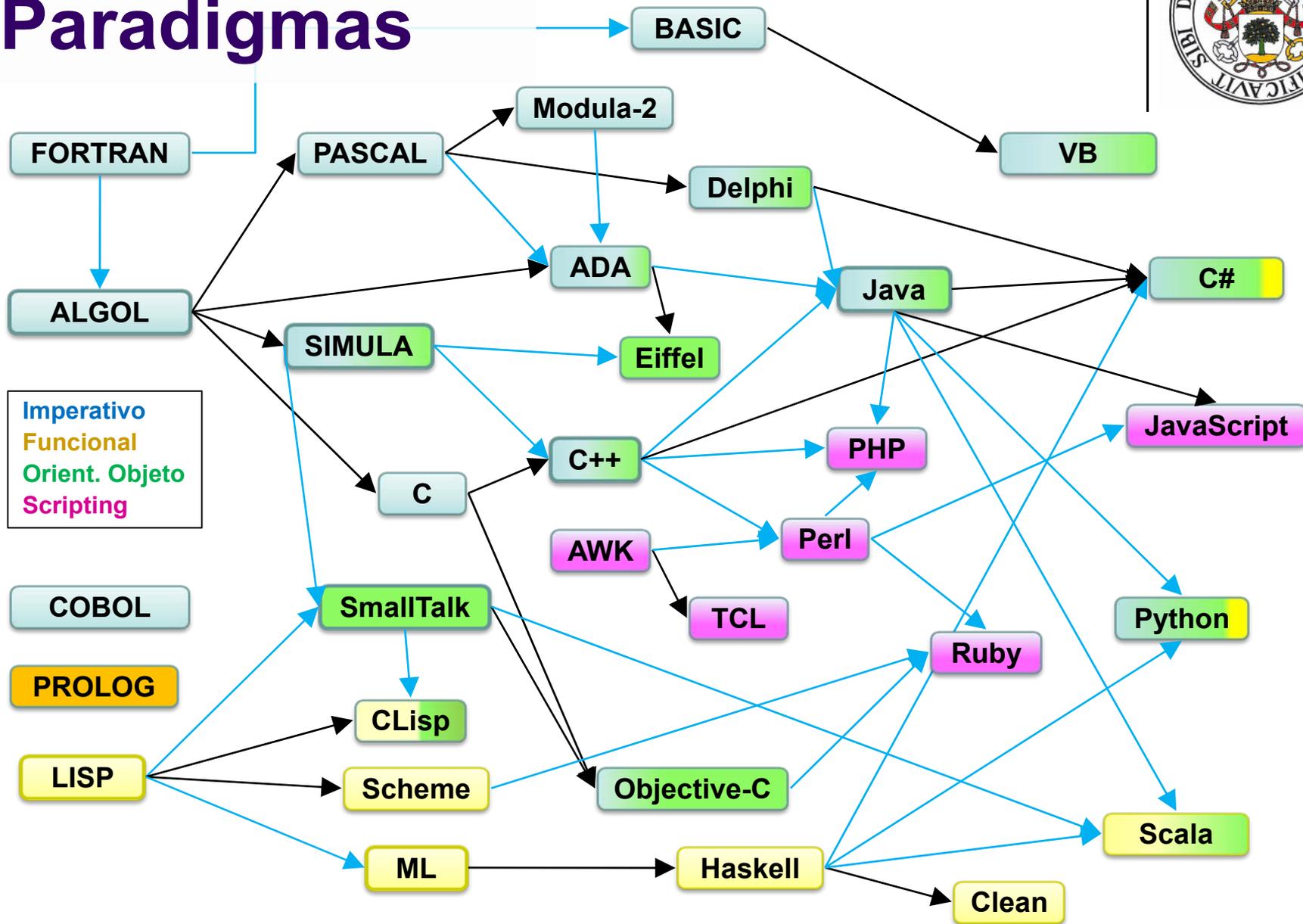


Evolución





Paradigmas





Evolución Histórica

- FORTRAN (Formula Translating) 1957
 - Orientado al cálculo científico
 - Proceso de arrays
 - GOTO asignado
 - Versiones II, III, IV, 66 (subrutinas), 77, 90 (array slicing, recursividad, modularidad, sobrecarga, TADs), 96-03-08 (paralelismo, orientación a objeto)
- COBOL (Common Business Oriented Lenguaje) 1959:
 - Orientado al mundo empresarial
 - Sintaxis basada en lenguaje natural: 400 palabras reservadas, con verbos, nombres, modificadores, etc.
 - Código auto-modificable: ALTER X TO PROCEED TO Y
 - Especificación detallada de valores numéricos (PIC)
 - Modularidad mediante Copybooks



Familias y Evolución Histórica

- LISP (List Processing) 1958, McCarthy
 - Orientado a la investigación (Inteligencia Artificial)
 - Basado en *s-expresiones*
 - Código y datos intercambiables
 - Tipado débil
- Familia Lisp:
 - Common Lisp , 1984 (Generalización, Orientación a Objeto)
 - Scheme, 1975 (Simplificación, Closures)
 - Clojure, 2007
- Familia ML:
 - Sistema de tipado Hindler-Millner
 - Haskell, 1990
 - Clean (1987), Scala (2003)

Ejemplo programa COBOL



IDENTIFICATION DIVISION.

PROGRAM-ID. PerformFormat4.

AUTHOR. Michael Coughlan.

- * An example program using the PERFORM..VARYING format.
- * Pay particular attention to the values produced by the
- * WITH TEST BEFORE and WITH TEST AFTER loops.
- * Note that the PERFORM within a PERFORM produces the same
- * results as the PERFORM..VARYING..AFTER

DATA DIVISION.

WORKING-STORAGE SECTION.

01 LoopCount PIC 9 VALUE ZEROS.

01 LoopCount2 PIC S9 VALUE ZEROS.

PROCEDURE DIVISION.

Begin.

DISPLAY "Start WHILE Iteration of LoopBody"

PERFORM LoopBody WITH TEST BEFORE

VARYING LoopCount FROM 1 BY 2

UNTIL LoopCount GREATER THAN 5.

DISPLAY "Finished WHILE iteration. LoopCount = " LoopCount.

...

Ejemplo programa LISP



```
(defun simplify (expression)
  (simplify-2 (rules) expression))

(defun simplify-2 (rules expression)
  (cond
   ((null rules) expression)
   (T (simplify-2 (cdr rules) (apply-rule (car rules) expression)))))

(defun apply-rule (rule expression)
  (substitute (car rule) (cadr rule) expression))

(defun substitute (pattern replacement expression)
  (cond
   ((null expression) ())
   ((occurs-at-front pattern expression)
    (substitute-at-front pattern replacement expression))
   (T (cons (car expression)
             (substitute pattern replacement (cdr expression))))))

(defun occurs-at-front (pattern expression)
  (cond ((null pattern) T) ((null expression) nil)
        ((matches (car pattern) (car expression))
         (occurs-at-front (cdr pattern) (cdr expression)))
        (T nil)))
```

Familias y Evolución Histórica



- ALGOL (Algorithmic Lenguaje) 1958/60/68 N.Wirth
 - Familia de lenguajes, diseñados por un comité de expertos
 - Bloques de código, recursividad, funciones internas, paso de parámetros
 - Arrays dinámicos, paralelismo, definición de operadores
- Familia ALGOL
 - PASCAL, 73 → Modula-2, 80
 - Delphi/ Free-Pascal, 93
 - ADA, 83
 - C, 74 → C++, 80
 - Objective-C, 86
 - C#, 95
 - Java, 95



Orientación a objeto

- SIMULA, 67
- Ortodoxa:
 - SmallTalk, 80 → Objective-C, 86
 - Eiffel, 86
- Parcial
 - ADA, 83 (Genericidad)
 - C++, 80 (Templates)
 - Java, 95 (Interfaces)
 - C# , 2001 (.NET)
- Basada en prototipos
 - JavaScript, 96
 - Python, 91



Lenguajes de Scripting

- Cliente (navegador):
 - HTML / CSS
 - ACME Script: JavaScript, ActionScript (Flash), 1995
 - Java (applets), 1995
- Servidor
 - PHP, ASP, 1995
 - Java (servlets), 2000
 - Ruby, 1995
- Propósito general
 - Perl, 1987
 - Tcl/Tk, 1989
 - Python, 1991

Otros lenguajes



- Programación lógica:
 - PROLOG, 1972
- Concurrencia
 - Erlang, 1986
 - Oz , 1991
- Bases de Datos
 - SQL (No es Turing-completo)
- Minimalistas
 - Forth, 1970
 - APL (1964) → J (1990)
 - BrainFuck