

COMPILEDORES E INTÉPRETES

APUNTES 4 DE SETIEMBRE

SEBASTIÁN ROJAS LORA 2018104604

APUNTADOR

**Todas las clases habrá un apuntador oficial,
todos los estudiantes deben ser
apuntadores mínimo 1 vez,
dependiendo de la cantidad de
estudiantes.**

**Se deben subir los apuntes en formato PDF,
con nombre COM20MMDD-k.pdf**

Al thread de apuntes en el foro:

<http://ec.tec.ac.cr/index.php/foro>

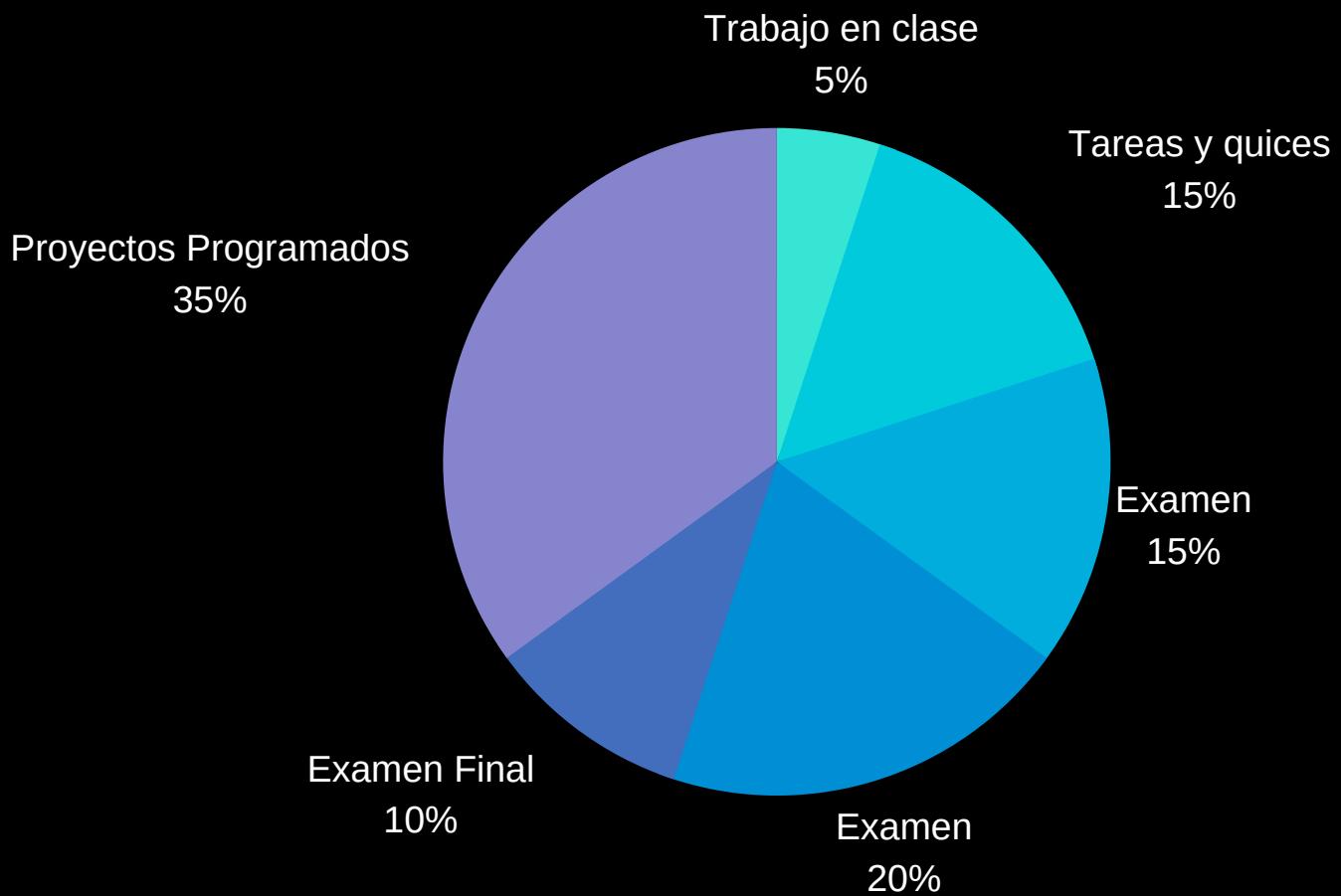
DATOS DEL PROFESOR

**Dr. Francisco J. Torres-Rojas
Email: torresrojas.cursos.05@gmail.com**

CONTENIDO DEL CURSO

- Introducción
- Análisis Léxico
- Análisis Sintáctico
- Análisis Semántico
- Generación de código
- Optimización
- Temas avanzados

EVALUACIÓN



Dependiendo de la cantidad de quices el profe eliminará los que tengan peor nota individualmente.
Además que el examen con la nota más alta tendrá el valor de 20% y el examen más bajo 15%

ASPECTOS IMPORTANTES

Tarea 0 -> Enviarla al asistente Riquelme, preferiblemente agregar el makefile.

Saber usar LateX y sino, aprenderlo :).

Los quices se tienen que enviar al asistente Riquelme.

Los grupos de proyecto quedaron para 3 personas, si quieren divorciarse hablar con el profesor.

Con júbilo y euforia los estudiantes aceptaron la evaluación.

No cometer fraude académico, cualquier intento de fraude será castigado de la forma más severa permitida por los reglamentos del ITCR.

LIBROS DE TEXTO

- **Compiler Construction - Kenneth C. Louden**
- **Compiler Principles, Techniques, and tool - Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman (Adicional)**
- **Crafting a Compiler with C - Charles N. Fischer, Richard J.LeBlanc. Jr (Adicional)**
- **The Language Instinct- Steven Pinker (Adicional, Ensayos)**

ENSAYOS CORTOS

Tareas Cortas (ingles y LateX):

- **Capítulo 1 - Ensayo Viernes 11 de Setiembre**
- **Capítulo 2 - Ensayo Viernes 18 de Setiembre**
- **Capítulo 3 - Ensayo Viernes 25 de Setiembre**
- **Capítulo 4 - Ensayo Miércoles 4 de Octubre**
- **Capítulo 5 - Ensayo Miércoles 14 de Octubre**

El ensayo consiste en un resumen del material y luego una opinión o lo que entendí + intentar relacionarlo con el curso.

La extensión del ensayo puede ser de 4-5 páginas.

**Enviar el ensayo al correo del profesor:
torresrojas.cursos.05@gmail.com**

**UN POCO DE
HISTORIA DEL
NACIMIENTO DE
LOS LENGUAJES
DE ALTO NIVEL**

**IF YOU DON'T COMPILE
FOR A WHILE,**



**IT WILL TAKE A WHILE
TO COMPILE**

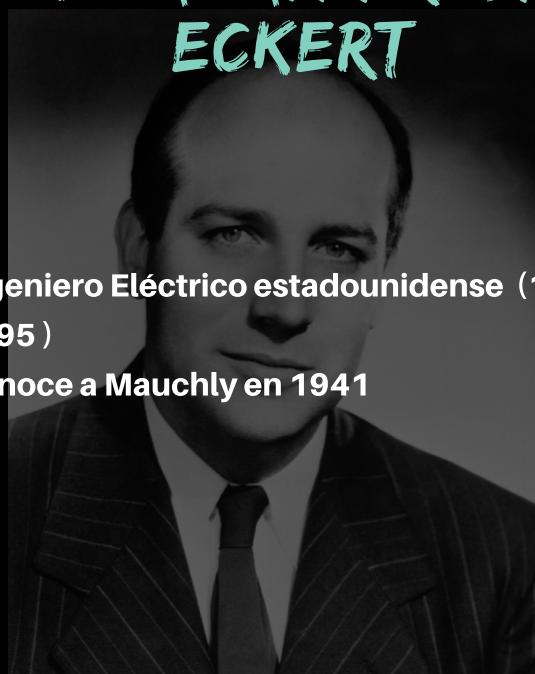
MONSTRUOS PELUDOS DE LA COMPUTACIÓN

JOHN WILLIAM MAUCHLY



Físico Estadounidense (1907 - 1980)
Ph. D. Fisica en Johns Hopkins University
Se interesó por la electrónica en 1941

JOHN ADAM PRESPER ECKERT



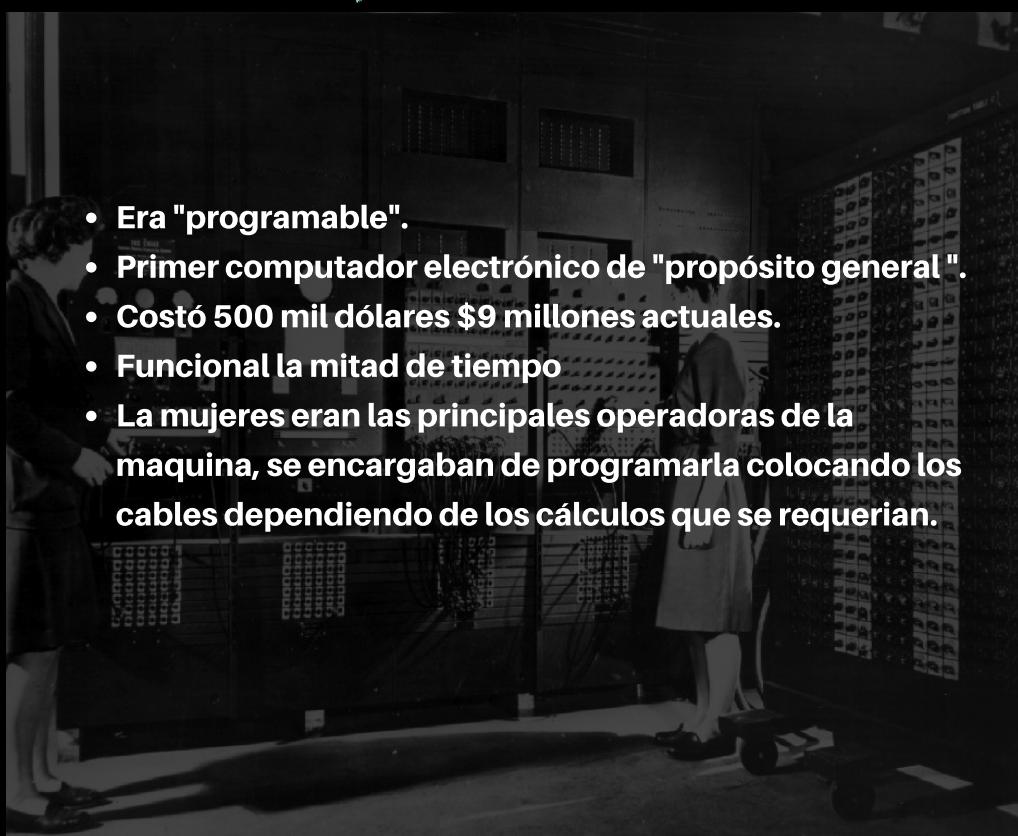
Ingeniero Eléctrico estadounidense (1919 - 1995)
Conoce a Mauchly en 1941



FABRICARON LA ELECTRONIC NUMERICAL INTEGRATOR AND COMPUTER

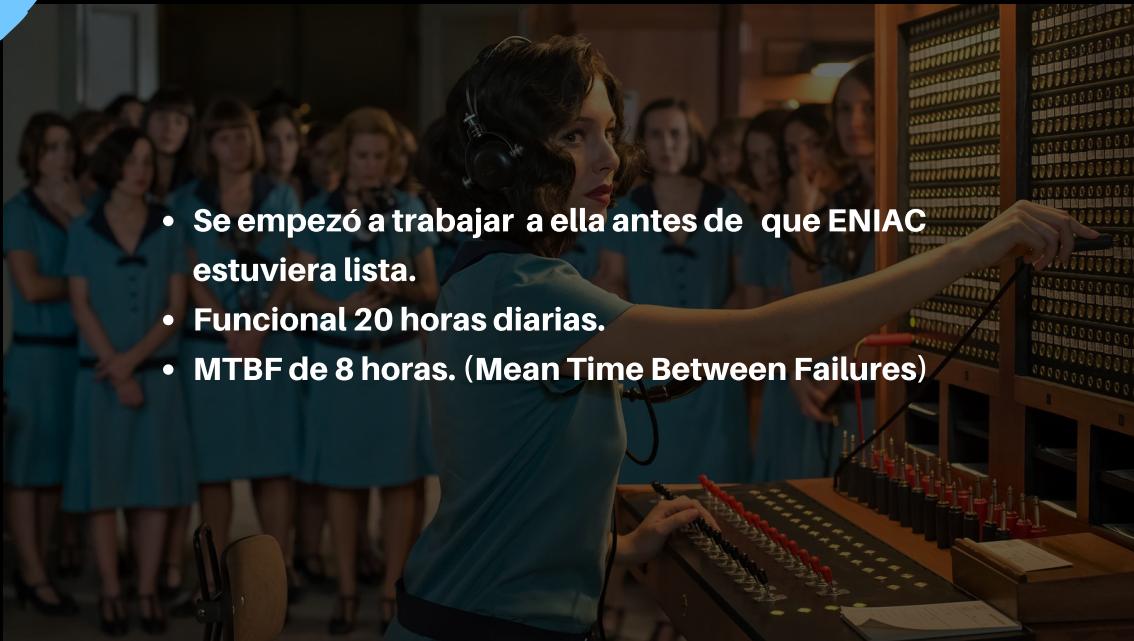
ENIAC 1946-1955

- Era "programable".
- Primer computador electrónico de "propósito general".
- Costó 500 mil dólares \$9 millones actuales.
- Funcionó la mitad de tiempo
- La mujeres eran las principales operadoras de la máquina, se encargaban de programarla colocando los cables dependiendo de los cálculos que se requerían.



FABRICARON LA ELECTRONIC DISCRETE VARIABLE AUTOMATIC COMPUTER

EDVAC 1949 - 1961



- Se empezó a trabajar a ella antes de que ENIAC estuviera lista.
- Funcional 20 horas diarias.
- MTBF de 8 horas. (Mean Time Between Failures)

JOHN VON NEUMANN 1903 - 1957

- Matemático húngaro
- Contribuyó en distintos campos tales como Arquitectura de computadores, Ciencias de la computación, IO, Teoría de juegos, Teoría de conjuntos.
- Parte del proyecto de Manhattan .



¿Primer fraude académico?

Crewmate

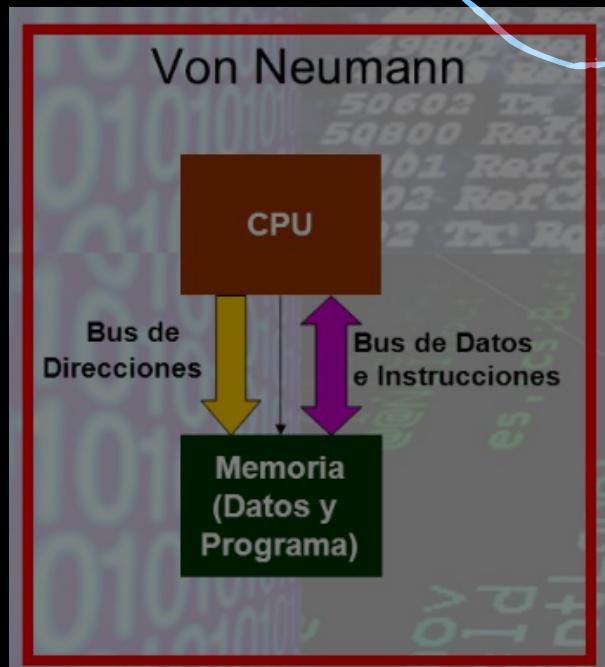
There is 1 Impostor among us



- Cuenta la leyenda que no era una persona muy agradable, además de publicar un paper acerca de las ideas de Mauchly y Eckert sin su consentimiento.

ARQUITECTURA DE VON NEUMANN

- Propuesto en 1945 por Von Neumann
- Programar era rediseñar el alambrado - antes era muy difícil, duraban hasta semanas.
- Nace el Software
 - Surgen ideas como El ciclo fetch, conjunto de instrucciones , programa que se lee de una memoria.



UNIDAD DE CONTROL

- Responsable del ciclo de fetch
- Establece el "lenguaje máquina" o arquitectura de la computadora.

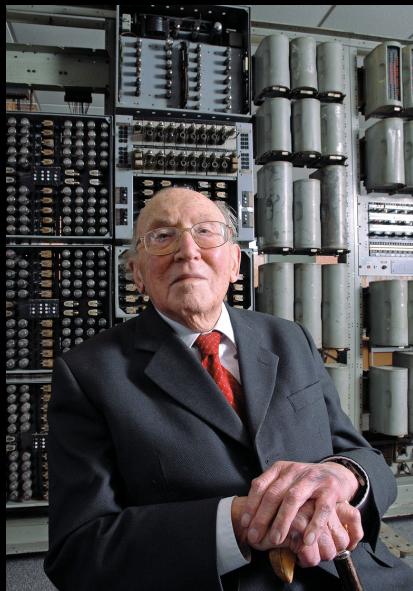
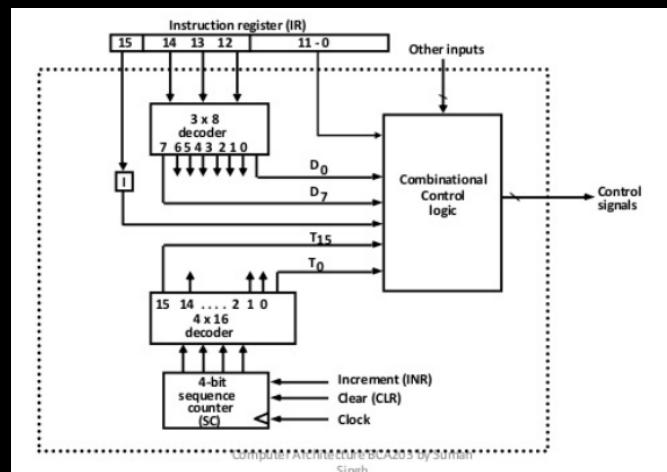
TIPOS DE DISEÑO DE CPU

Alambrada

Microprogramada

UNIDAD DE CONTROL ALAMBRADA

- Se implementan con compuertas lógicas
- Cada instrucción de lenguaje máquina tiene sus circuitos correspondientes
- Arquitectura fija, para cambiar el conjunto de instrucciones hay que rediseñar la UC.
- Son más rápidas y eficientes que las UC microprogramadas
- Poco o nada flexibles



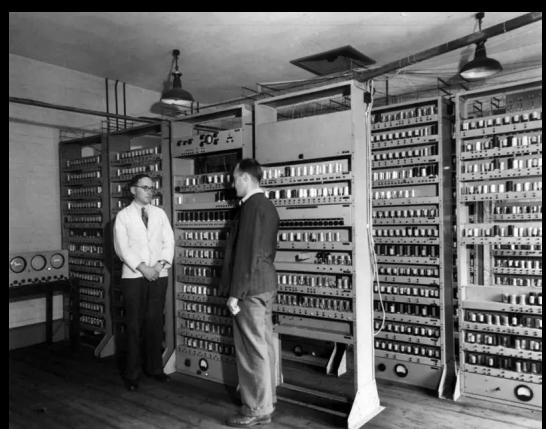
SIR MAURICE VINCENT WILKES 1913 - 2010

- Físico inglés.
- Uno de los fundadores de la Ciencia de la computación e IO
- En 1951 inventa el concepto de microprogramación
- Contribuye con el concepto de subrutinas para FORTRAN
- Gana premio Turing en 1967

INVENTA Y CONSTRUYE

EDSAC 1946 - 1958

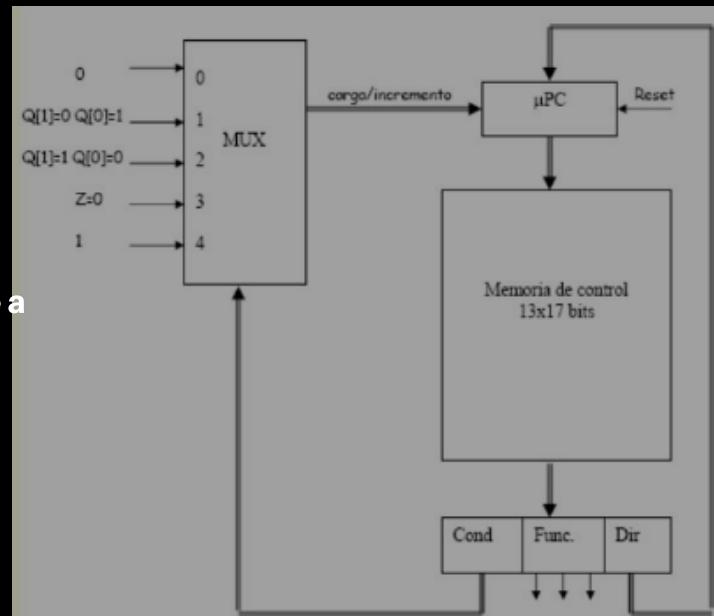
- Computadora electrónica creada en UK.
- Sus siglas significan Electronic Delay Storage Automatic Computer



UNIDAD DE CONTROL MICROPROGRAMADA

- Existe una organización de computadora subyacente
- La arquitectura se crea sobre esta organización
- El conjunto de instrucciones se define sobre la organización
- Una instrucción de lenguaje máquina se construye a base de múltiples microinstrucciones, tomada de una micromemoria
- Sacrifica rendimiento pero aporta la flexibilidad

○ Tip: Al hablar sobre microprogramación agregarle a todo "micro", parecerá que sabemos de lo que hablamos



UC MICROPROGRAMABLE

- Podríamos crear un sistema que haga la contabilidad pero... sería una tortura
- Microprogramar es muy difícil
- Aunque si lo lográramos no habría sistema más rápido, sería lo mejor de lo mejor

- Lo único real es el hardware
- Se rodea de microprogramación para darle un comportamiento
- Teóricamente podríamos crear cualquiera cosa



Comportamiento

Micromprogramación

Hardware

PRODUCTIVIDAD VS. EFICIENCIA

Productividad: líneas de código sin errores por unidad de tiempo al desarrollar un sistema

Eficiencia: cantidad de recursos(tiempo, espacio, etc.) que un sistema en funcionamiento requiere por cada resultado

- Al ser muy difícil producir código de microprogramación la productividad va ser muy baja pero lo que produzcamos será muy eficiente

Microprogramación

Hardware

P

E

LENGUAJE MÁQUINA

- La microprogramación define el lenguaje máquina
- Mucho más fácil que microprogramar
- El programador tiene la responsabilidad de colocar programa en Ram, códigos de operación, modos de direccionamiento, formato de argumentos
- Inherently no portátil

- El lenguaje máquina resulta tener una brecha semántica más corta que aumenta la productividad aunque se disminuye la eficiencia del sistema.

Lenguaje máquina

Microprogramación

Hardware

P

E

LENGUAJE ENSAMBLADOR

- Programar en lenguaje máquina es tedioso y propenso a errores
- Inventado en los 50's
- Programa que recibe **nemónicos y nombres simbólicos con cierta sintaxis simple y genera lenguaje máquina**
- Cálcula desplazamientos
- Mucho más fácil - no portátil

- Mucho más fácil que el lenguaje máquina, aumentó la productividad de manera significativa
- Disminuye la eficiencia entre más nivel

Ensamblador

Lenguaje máquina

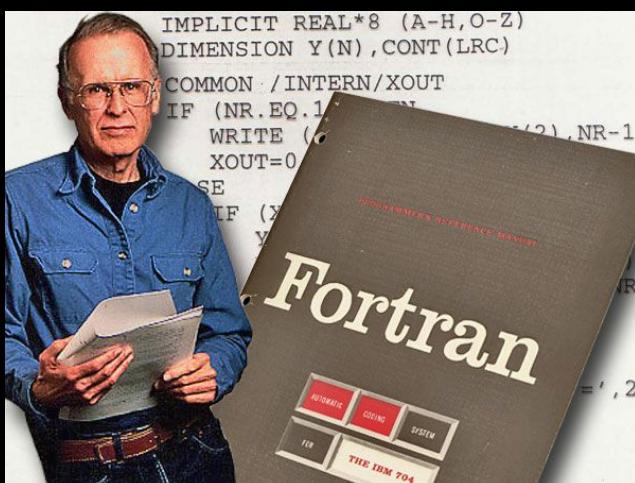
Microprogramación

Hardware

P

E

JOHN BACKUS 1924 - 2007



- Matemático y Científico de la Computación
- M. Sc. en Matemáticas - Columbia University
- Trabajó con IBM desde 1950 programando en lenguaje máquina y ensamblador.
- El primer proyecto fue un programa para calcular las posiciones de la Luna en lenguaje máquina.
- En 1956 crea FORTRAN, primer lenguaje de alto nivel de uso extendido.
- En 1977 inventa FP lenguaje de programación funcional.

Apuntes del 04/09/2020

Andrei Leon Salas

8 de septiembre de 2020

Parte I Datos administrativos

Las clases son los miercoles y viernes de 7:30 am a 9:20 am. Se va a utilizar teams y zoom para las clases, y la entrega de proyectos, tareas y apuntes por medio del foro <http://ec.tec.ac.cr/index.php/foro/>. Si necesita acceso puede solicitarlo a la asistente del profesor Silvia silmeli97@gmail.com. Se tiene que postear en el foro en 4 dias naturales o un poco mas para que quede bien con el siguiente formato: COM200904-k.pdf siendo k 1 si lo entrego de primero o 2 si lo posteо de segundo. El sentido de ser apuntador es aprender bien la materia ya que se tiene que poner mucha atencion en el momento que el profe habla para hacer buenos apuntes, y adicionalmente extender la informacion que dio el profe (con graficos, imagenes o informacion extra). Estos apuntes son revisados por el profe y tiene que venir bien su informacion para identificarse.

Formas de contacto

- Correo electronico Torresrojas.cursos.05@gmail.com: este correo es exclusivo para este curso, cualquier tarea o proyecto entregado a otro correo se considera un cero
- Foro: <http://ec.tec.ac.cr/index.php/foro/>: es el medio oficial y se recomienda hacer las consultas aqui, para que tambien otros companeros puedan verlo y asi aclarar sus propias dudas

Evaluacion

- Tareas y quices [15 %]: Se establecio que se haran los miercoles y no se reponen. Son individuales a menos que el profesor mencione lo contrario, y se utiliza el correo electronico como medio de entrega oficial. Todo trabajo escrito se debe utilizar LATEX. Los quices mas bajos se eliminaran al final del curso, para ayudar al estudiante, y esta es la razon por la que no se

reponen los quices. Se hacen a MANO, hay que tomarle foto y enviarlo a Riquel driquelme2699@gmail.com (NO enviar copia del correo al profesor). La entrega tardia se le quitaran 3^{k+1} puntos donde k es cada dia de atraso. Las tareas que sean programadas se haran sobre C en Linux e individuales o en grupos de 3 dependiendo del enunciado. Si hay una revision y no se presenta nadie, es tambien un 0 automatico (en grupo tiene que estar aunque sea uno). Los ensayos tienen las siguientes fechas, tienen que ser en ingles y en LATEX, un resumen del material y una opinion personal o conclusion de lo que trataba el material. extension de 4 o 5 paginas. Se entregan directo al profesor Torresrojas.cursos.05@gmail.com

- Ensayo 1 Capitulo 1 Viernes 11 de septiembre
 - Ensayo 2 Capitulo 2 Viernes 18 de septiembre
 - Ensayo 3 Capitulo 3 Viernes 25 de septiembre
 - Ensayo 4 Capitulo 4 Miercoles 4 de octubre
 - Ensayo 5 Capitulo 5 Miercoles 14 de octubre
- Trabajo en clase [5 %]
 - Examen [15 %]
 - Examen [20 %]
 - Examen final [10 %]
 - Proyectos programadis [35 %]: todos son hechos en C

Libros de texto

- Compiler Construction – Kenneth C. Louden
- Compiler Principles, Techniques, and tool – Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman (extra)
- Crafting a Compiler with C – Charles N. Fischer, Richard J.LeBlanc. Jr(extra)
- The Language Instinct- Steven Pinker(extra)

Fraude academico

Sera fuertemente penalizado cualquier tipo de fraude academico. El profe enfatizo mucho en esto, y que la escuela esta notando un incremento de esta mala practica. Hay que usar el sentido comun para saber si uno esta incurriendo en fraude pero ante la duda el profe dijo que con gusto el respondia correos con cualquier duda. Algunos ejemplos de fraude son:

- Usar el trabajo de otro como propio
- Conseguir de alguna forma adelantada los examenes o pruebas
- Usar recursos o cualquier otra cosa que adulteren la evaluacion de los examenes, proyectos, tareas, etc
- NO es plagio utilizar informacion del foro de cursos pasados

Parte II

Historia

Personas

Nombre	Nacimiento	Universidad
John William Mauchly	USA 1907-1980	Johns Hopkins University
John Adam Presper Eckert	USA 1919-1995	Universidad de Pensilvania
John von Neumann	Hungría 1903-1957	Universidad de Budapest
Sir Maurice Vincent Wilkes	Británico 1913-2010	Newcastle University
John Backus	USA 1924-2007	Columbia University

Nombre	Contribución	Premio Turing o tasa de perro
John William Mauchly	Co-creador de ENIAC	N/A
John Adam Presper Eckert	Co-creador ENIAC	N/A
John von Neumann	Arquitectura de computadores, proyecto manhattan, First Draft (Plagio?)	N/A
Sir Maurice Vincent Wilkes	EDSAC, microprogramación, subrutinas	1967
John Backus	FORTRAN, primer lenguaje de alto nivel, BNF, FP	1977

Maquinas

ENIAC

Programable por medio de cables, era la primera computadora de propósito general, que se podía cambiar su función sin tener que crear la máquina desde cero. Costó medio millón de dólares (9 millones con la inflación actual). Tenía un tiempo de funcionamiento del 50 %, y eran mujeres quienes principalmente se encargaban de toda la programación de esta. Se la decían «Giant Brain» según la prensa, e igual hubo historias de terror de que las computadoras dominarían el mundo (no tan alejado de la realidad poniéndonos filosóficos).

EDVAC

Programable pero en vez de ser cables, se hacia en memoria, muy innovador en la época, tanto que aun ahora se utiliza esta misma idea. Se empezó a trabajar desde antes que ENIAC estuviera lista, y fue funcional 20 horas diarias con un MTBF (MEAN TIME BETWEEN FAILURES) de 8 horas. Costo para su momento medio millón de dólares (6643615 millones de dólares actuales)

Contribuciones

First draft

Fue un ensayo sobre el funcionamiento de EDVAC que circuló en 1945. Se habla mucho de que fue un fraude académico de John von Neumann hacia Mauchly y Presper, junto con este documento se propuso la arquitectura von Neumann

Arquitectura von Neumann

Propuesta en 1945 por von Neumann, se comenta que tuvo influencia de Turing y que pudo tener ideas robadas de otros académicos. Basicamente es la primera arquitectura que integraba la idea del software. Usando un programa que se lee en la memoria, y ejecutandolo indefinidamente con un ciclo de fetch. Esencialmente el software

Unidad de control

Es el responsable de ejecutar el ciclo de fetch, tiene de forma simple los pasos definidos de estar recuperando en memoria las instrucciones, ejecutarlas, y volver al primer paso indefinidamente, siendo estas instrucciones posibles de cambiar el estado de la máquina y ejecutar de forma diferente. No existía antes del EDVAC

Diseño de la unidad de control

Existen dos tipos de diseños, el alambrado que trata de hacer todas las instrucciones por medios físicos (compuertas lógicas por ejemplo) y el microprogramado que es tener un pequeño programa con esas instrucciones pero por software, no por hardware. La primera es más eficiente pero la segunda más flexible, ya que en la primera no se pueden agregar instrucciones sin tener que recrear todo, y en la segunda si podemos agregar sin tener que construir de nuevo el hardware. Cabe destacar que microprogramada es que ya el set de instrucciones viene hecho y no se puede cambiar, la microprogramable es la que se puede cambiar

Niveles de eficiencia y dificultad

Como vimos antes, las instrucciones hechas en hardware son mas rapidas pero menos eficientes. Esta regla se cumple generalmente entre mas bajo se programa mas eficiente va a ser. Un sistema hecho a partir de microprogramas va a ser mucho mas veloz que en java, pero va a ser muy lento crear ese codigo. A esto se le llama brecha semantica. un sistema asi seria muy eficiente pero poco productivo.

- Eficiencia: cantidad de recursos que un sistema en funcionamiento requiere (el programa que uno hace es eficiente, no la persona)
- Productividad: la cantidad de lineas de codigo sin errores por unidad de tiempo al desarrollar un sistema (la persona es quien es productivo, no el programa hecho)

En microprogramacion la eficiencia es alta pero la productividad baja. En lenguaje maquina la eficiencia baja pero la productividad sube. Pero en lenguaje ensamblador para una excepcion, la productividad sube pero la eficiencia no baja.

Lenguaje ensamblador

Por no ser tan semanticos, es muy tedioso y propenso a errores, por lo que en 1950 se crea ensamblador, que crea nemonicos y nombres simbolicos para poder generar lenguaje maquina. Como es equivalente lo generado por ensamblador a lenguaje maquina es igual de eficiente pero mucho mas facil de utilizar

Quiz 01

- 1. Defina detalladamente los siguientes conceptos:
 - a) ENIAC
 - b) John Mauchly
 - c) “First Draft” de von Neumann
 - d) Maurice Wilkes
 - e) EDVAC
 - f) Ciclo de fetch
 - g) Productividad
 - h) EDSAC
 - i) Eficiencia
 - j) Unidad de Control alambrada

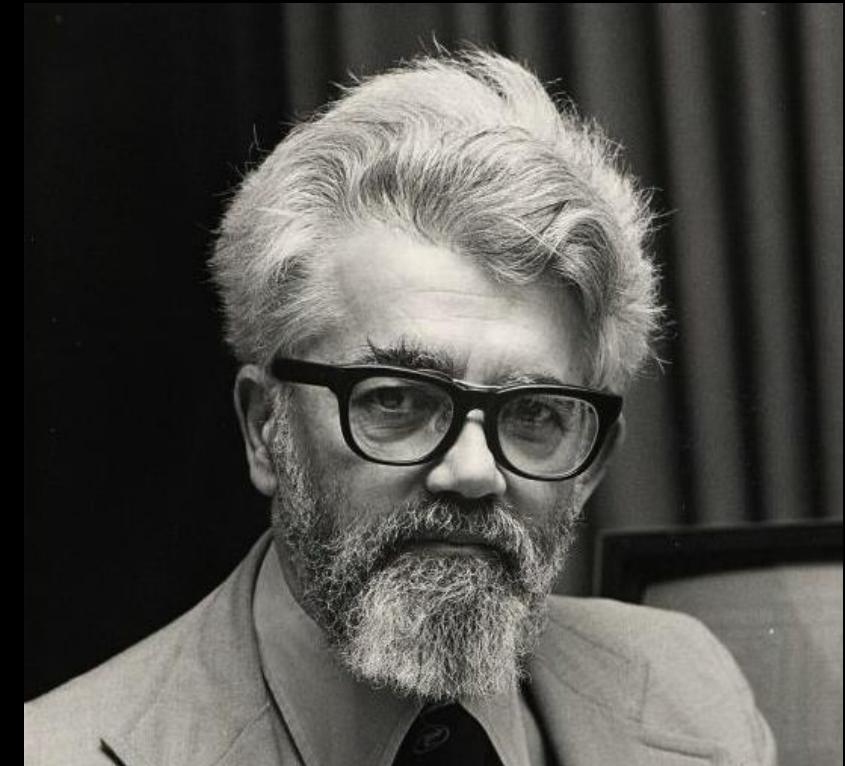
John Backus

- En 1977 gana un Turing Award por FORTRAN y su influencia en el desarrollo de lenguajes de alto nivel



John McCarthy 1927-2011

- Ph, D. en Matemáticas, Princeton University
- Inventó **LISP** y trabajó en la definición de ALGOL
- Pionero de la inteligencia artificial
- Turing Award 1971 por su contribución en el campo de la inteligencia artificial
- Inventa el concepto de “garbage collection”
- Influenciado por el trabajo de von Neumann
- Fue profesor en el MIT, Stanford, Princeton



Grace Murray Hopper 1906-1992



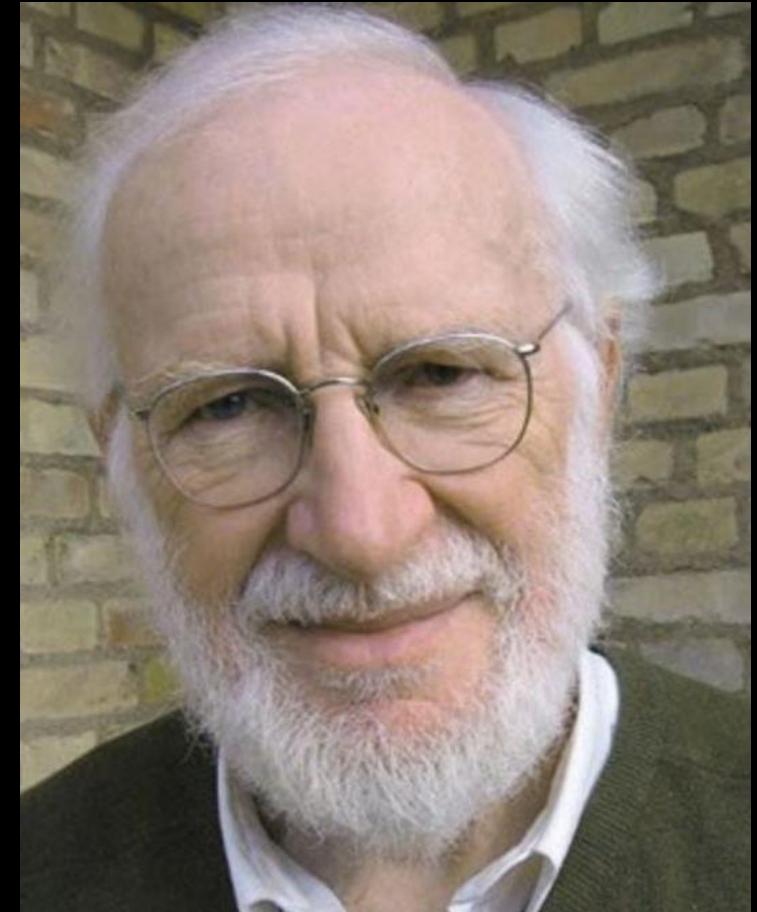
- Ph. D. en Matemáticas, Yale University
- Almirante de la Marina USA
- Primer programadora de la historia
- Inventó COBOL
- Primer compilador? Desarrolla A-0 que traducía código matemático a código maquina que era mas cercano a un Linker o Loader pero que originalmente se le llamo compilador, por eso la confusión
- No recibió Turing Award 😞
- Inventa los términos “compilador” y “debugging”

USS Hopper, nombrado por
Grace Hopper

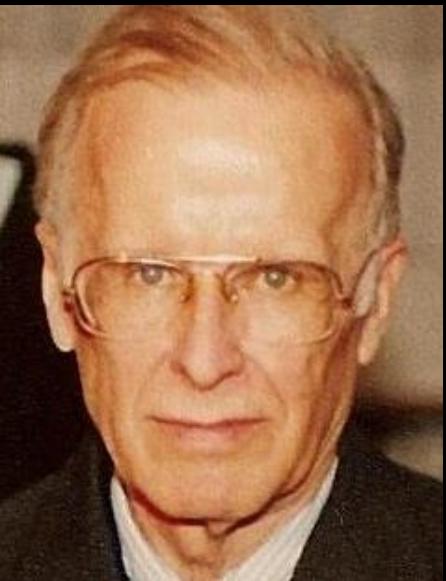


Peter Naur 1928-2016

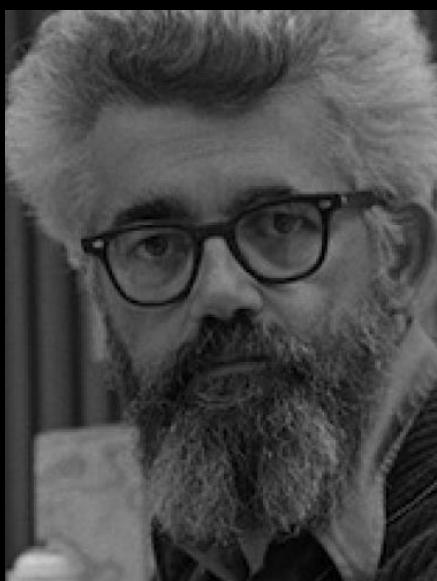
- Ph, D. en astrofísica, Universidad de Copenhague
- Líder del comité que creo ALGOL
- Mejora el BNF (Backus Normal Form), pasa a llamarse Backus Naur Form
- Gana el Turing Award en el 2005, por sus aportes en ALGOL 60 y diseño de compiladores



4 Grandes



FORTRAN



LISP



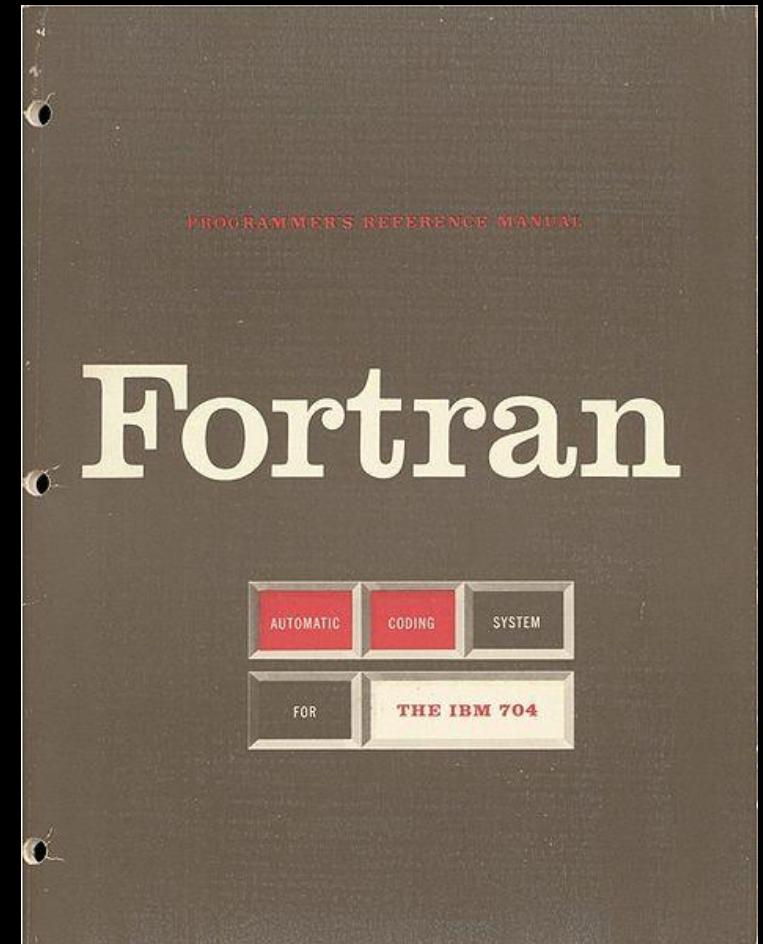
COBOL



ALGOL

FORTRAN – Formula Translation

- Desarrollado de por Backus en IBM (1950s)
- Fue el primer LAN de uso generalizado
- Ha evolucionado con el tiempo:
 - FORTRAN
 - FORTRAN II, III, IV
 - FORTRAN 66, 77, 90, 95, 2003, 2008, 2018
- Aun se utiliza
- Favorito de los físicos y se utiliza mucho en High Performance Computing (HPC). Se usa una versión moderna de FORTRAN
- Se escribió mucho código en este lenguaje y es mas fácil mantenerlo que hacer versiones nuevas con lenguajes modernos



C FOR COMMENT	RE SERV ING	FORTRAN STATEMENT				IDENTI FICATION
		STATEMENT NUMBER	72	73	80	
C		PROGRAM FOR FINDING THE LARGEST VALUE				
C	X	ATTAINED BY A SET OF NUMBERS				
		DIMENSION A(999)				
		FREQUENCY 30(I1,10), 5(100)				
1		READ 1, N,, (A(I), I= 1,N)				
		FORMAT (I3/(12F6.2))				
		BIGA = A(1)				
5		DO 20 I = 2,N				
30		IF (BIGA-A(I)) 10,20,20				
10		BIGA = A(I)				
20		CONTINUE				
		PRINT 2, N, BIGA				
2		FORMAT (22H1THE LARGEST OF THESE 13, 12H NUMBERS IS F7.2)				
		STOP 77777				

```

C COMPUTE POSITIVE ROOT OF A
C QUADRATIC EQUATION
  READ INPUT TAPE 3, 201, A, B, C
201 FORMAT (3I5)
  IF (A) 300,400,400
300 STOP 1
400 R=-B+SQRT(B*B-4*A*C) / (2*A)
  WRITE OUTPUT TAPE 4, 501, R
501 FORMAT (F5.3)
  STOP
END

```

- En las primeras versiones de FORTRAN todo debía ser escrito en mayúsculas
- Las columnas son muy importantes
- C en la primer columna denota comentarios
- Columnas 2 – 5 etiquetas para jumps
- Columna 6 denota continuación de línea anterior
- Columna 7 - 72 es código
- Utiliza if aritmético
 - If (expresión aritmética) negativo, cero, positivo
- Arreglos se declaran y se usan con paréntesis redondos, podía crear confusiones

LISP – List Processor

- Desarrollado por McCarthy (MIT), 1950s
- Utiliza muchos paréntesis y notación prefija
 - notación infija ($k + j$), notación prefija ($+ k j$)
- Orientado a IA
- Precursor de la programación funcional
 - Todo son listas
 - Se construyen funciones y se pasan como datos
- Es interpretado
- Derivados: Scheme, Common Lisp, Clojure



```
(defun factorial (N)
  "Compute the factorial of N."
  (if (= N 1)
      1
      (* N (factorial (- N 1)))))
```

Define la función

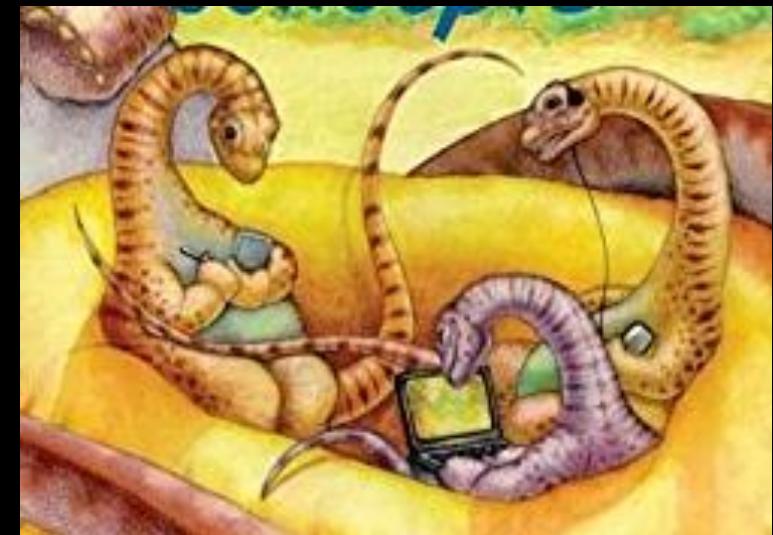
then, devuelve 1

else
Multiplica el N * el factorial de N-1

Función que calcula factorial de forma recursiva

COBOL – Common Business-Oriented Language

- Desarrollado por Grace Hopper en 1959
- Ella hizo un lenguaje que cualquier persona entendiera que era lo que hacia
- Ventajas – fácil de programarlo, el propio programa es la documentación
 - MOVE Y TO X.
 - IF X IS GREATER THAN Z MOVE 1 TO Y.
- Desventajas – El código es muy grande. Después de un rato las instrucciones quedan ilegibles. Muchas palabras reservadas.
- La industria amaba COBOL a diferencia de la academia



Programadores de COBOL

COBOL – cont.

- Aun es muy utilizado
- hay mucho Legacy code
- 1997 => 80% estaba escrito en COBOL
- 2012 => 60% estaba escrito en COBOL

Datos actuales

- COBOL es la base del 43% de los sistemas bancarios [1]
- COBOL maneja 95% de las transacciones en los cajeros [1]
- COBOL hace el 80% de las transacciones de tarjetas de crédito en persona [1]
- 70% de todas las transacciones de negocios mundiales dependen de COBOL [2]
- Y aún continua creciendo porque estos sistemas son “modernizados” pero no reemplazados

(1) <https://www.howtogeek.com/667596/what-is-cobol-and-why-do-so-many-institutions-rely-on-it/>

(2) <https://blog.hackerrank.com/the-inevitable-return-of-cobol/>

ALGOL – Algorithmic Language

- Diseñado por el comité ACM/GAMM (USA/Europa) en 1958. Dirigido por Peter Naur
- Querían resolver los defectos de FORTRAN
- Participaron personajes importantes de aquella época
- Primero en ser descrito en BNF



De izquierda a derecha
Arriba: John McCarthy, Fritz Bauer, Joe Wegstein
Abajo: John Backus, Peter Naur, Alan Perlis

ALGOL – cont.

- Se introducen cosas que utilizamos todos los días:
 - Bloques begin-end
 - Llamadas por valor y referencia
 - Variables locales
 - then/else, reemplazando el if aritmético
 - etc.
- Era el anti-COBOL, la academia lo amo pero la industria no lo acogió tanto como a el COBOL
- Existen muchas versiones: ALGOL 58, 60, 68, 73
- Ya no se utiliza sin embargo su legado continua
- Nace la familia de “Algol-like languages”:
 - BCPL, B, Pascal, PL/1, Simula, C, C++, Java, etc.

```
begin
procedure movedisk(n, f, t);
integer n;
integer f;
integer t;

begin
outstring (1, "move ");
outinteger(1, f);
outstring (1, " --> ");
outinteger(1, t);
outstring (1, "\n");

end;

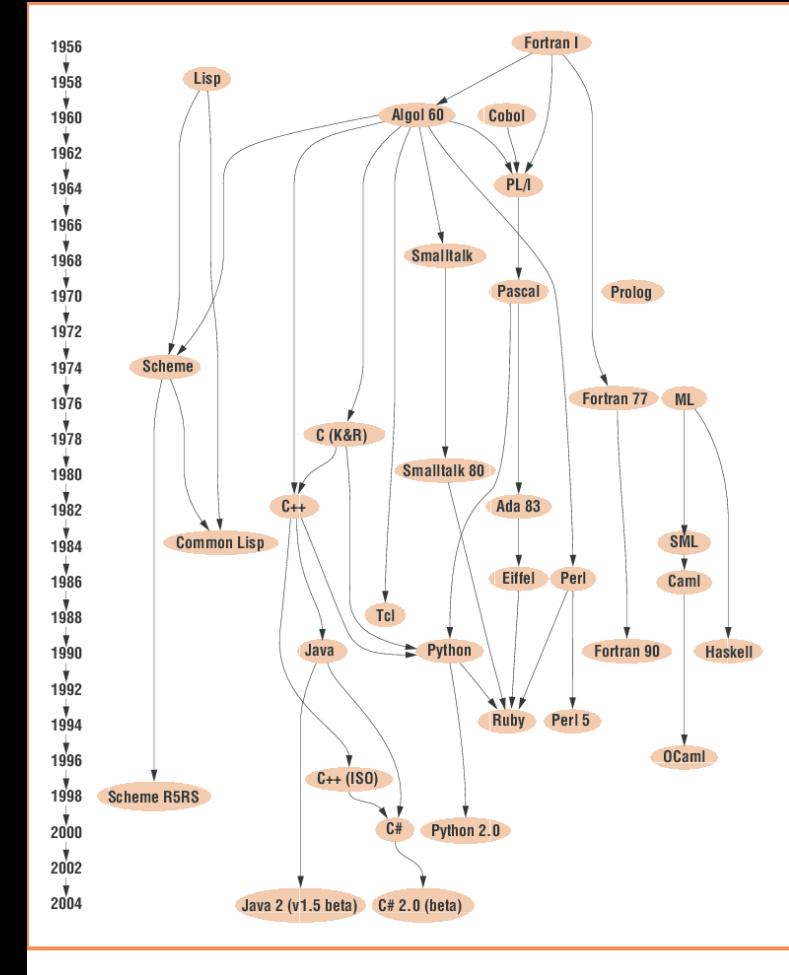
procedure dohanoi(n, f, t, u);
integer n;
integer f;
integer t;
integer u;

begin
if n < 2 then
    movedisk(1, f, t)
else
begin
    dohanoi(n - 1, f, u, t);
    movedisk(1, f, t);
    dohanoi(n - 1, u, t, f);
end;
end;

dohanoi(3, 1, 3, 2);
```

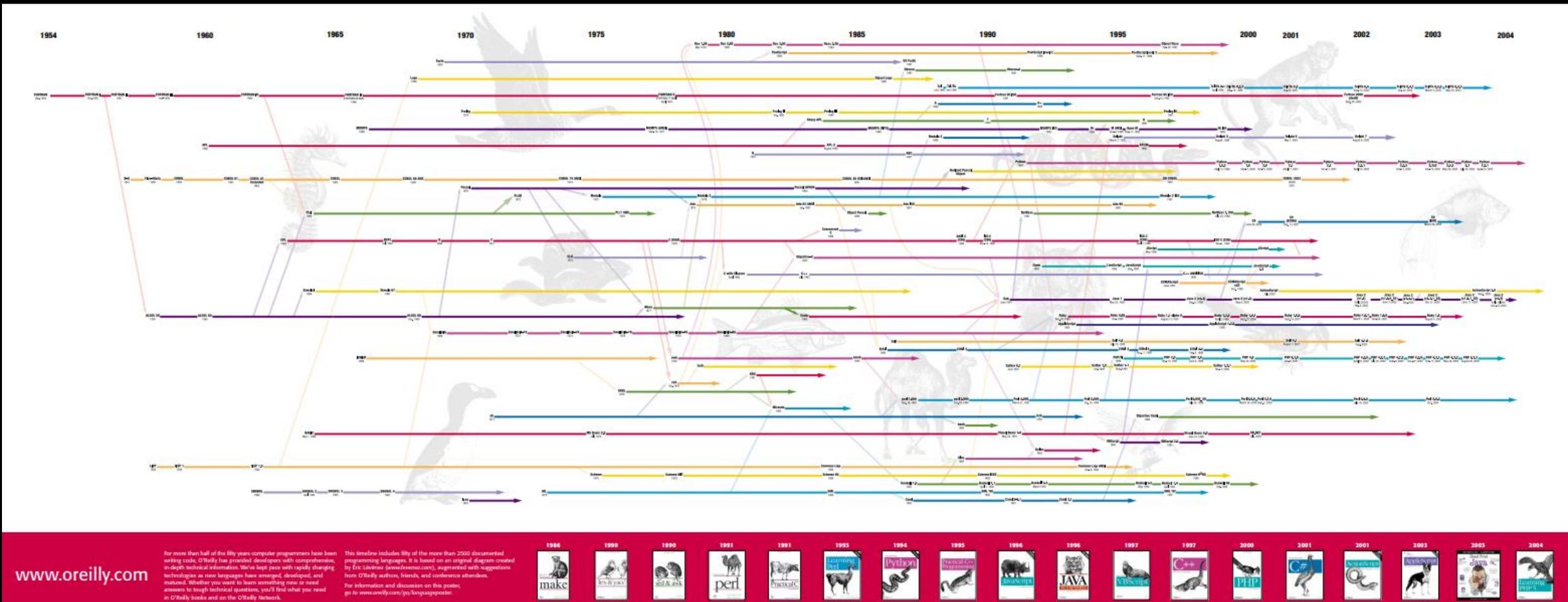
Lenguajes de Alto Nivel

- Existen desde 1956
- Es mas fácil programar en alto nivel
- La productividad es enorme
- Define la industria
- Aparecen lenguajes nuevos por la brecha semántica, hacer uno cuya brecha sea pequeña con el problema para resolverlo mas fácilmente



History of Programming Languages

O'REILLY®



Ver en grande: https://omohundro.files.wordpress.com/2010/01/prog_lang_poster-1.pdf

Lenguajes de Alto Nivel – cont.

- Definen una nueva capa
- Son independientes del hardware
- Portabilidad, solo se necesita un compilador que traduzca a otra máquina
- Abstracción de programación, se piensa en el lenguaje de alto nivel, no en la máquina.
- El hardware se vuelve “irrelevante”



Productividad vs Eficiencia

Lenguajes de Alto Nivel

P: 

E: 

Ensamblador

P: 

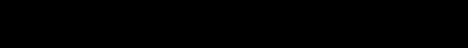
E: 

Lenguaje Máquina

P: 

E: 

Microprogramación

P: 

E: 

Hardware

- Productividad máxima
- Las maquinas cada vez son mas rápidas por lo que se pierda de eficiencia puede que no sea tan notable
- Al final es preferible sacrificar la eficiencia por la productividad

- Ensamblador y lenguaje máquina mantienen el mismo nivel de eficiencia
- Cualquier cosa que puedo hacer en máquina la puedo hacer en ensamblador

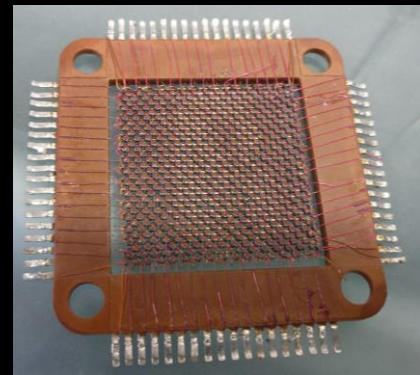
IBM 701

- Primera computadora comercial – Abril 1952
- IBM dominaba en el mundo de la computación
- Solo se fabricaron 19
- Configuración:
 - 4096 palabras de 36 bits
 - 150 mil inst/seg
 - 150 tarjetas/seg (I)
 - 100 tarjetas/seg (O)
- IBM alquilaba las maquinas a \$23,750 (\$232,945.38 actuales)
- IBM ofrecía el mantenimiento, cursos de entrenamiento, accesorios, repuestos, etc.
- MTBF 30 min (Tiempo promedio entre fallas)



IBM 704

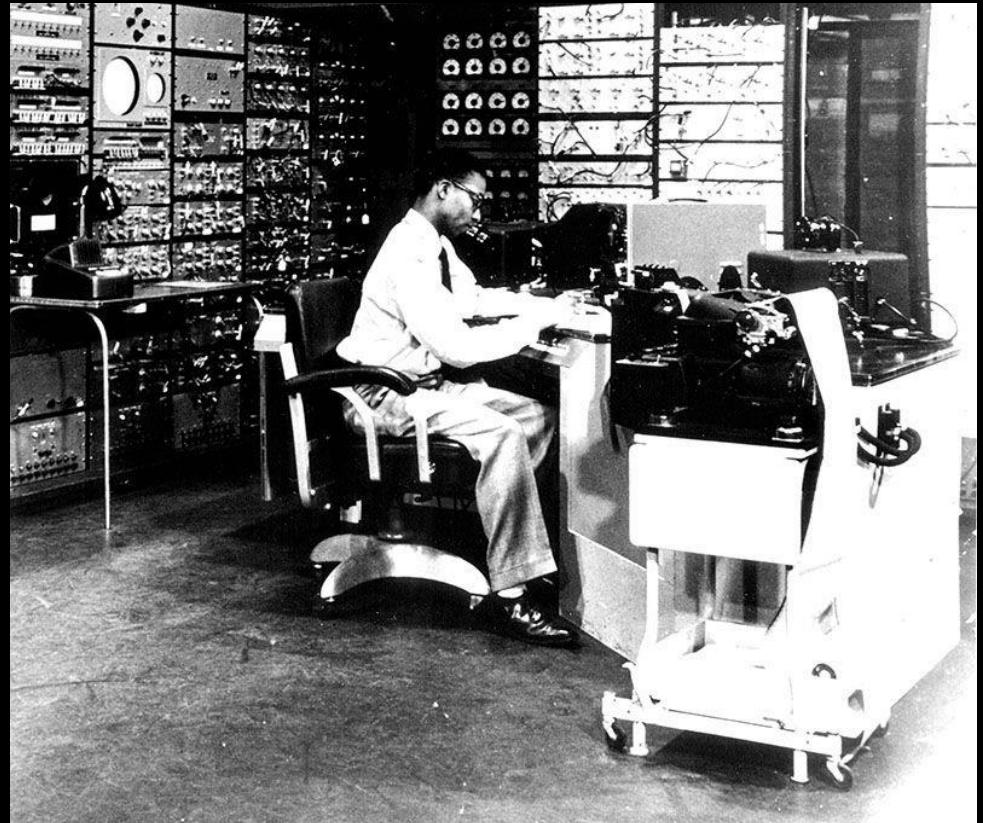
- 1954-1960
- Primera con punto flotante
- 200 maquinas. Alquiler a \$35,550 por mes (\$343,497.58 actuales)
- Doble de rápida que el 701, pero incompatible con el código de la anterior, había que convertir todo
- Tenía dispositivos offline para copiar tarjetas a cintas
- Usa memoria de núcleos de ferrita
- MTBF 8 horas
- Traía FORTRAN disponible, por ello fue un éxito



Ejemplo de una memoria de nucleo de ferrita

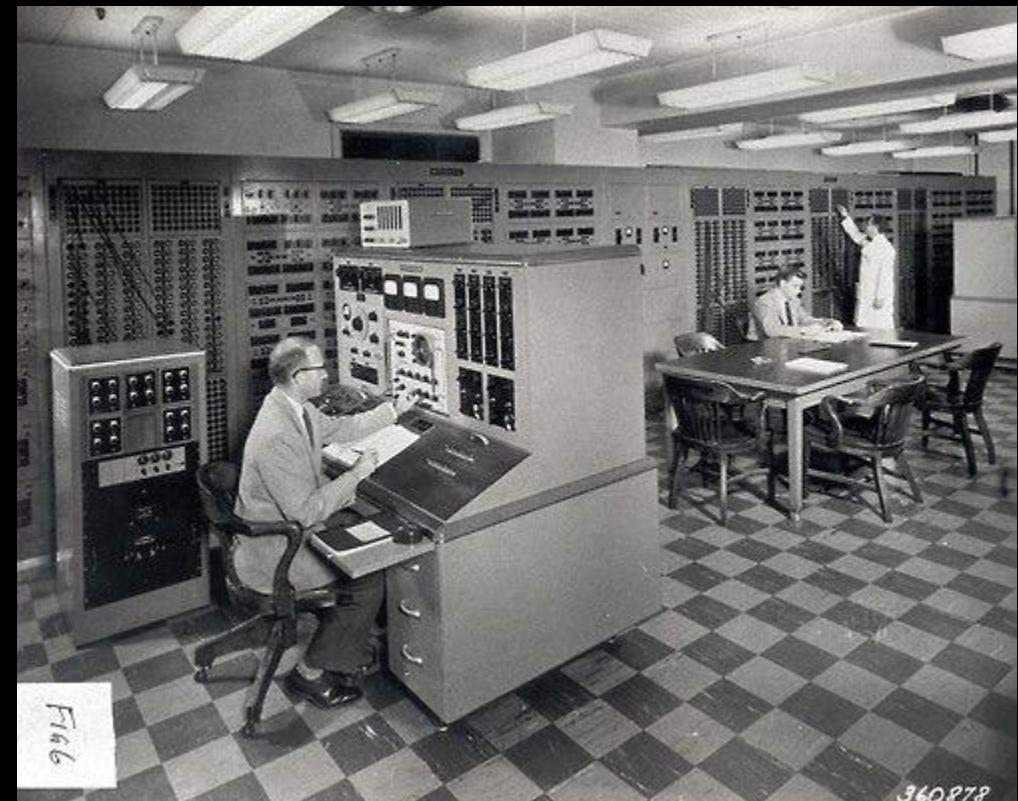
Máquina Dedicada

- Las computadoras eran caras y muy pocas personas las sabían usar
- En los 50s se reservaban “Horas de máquina”
- El que usaba la compu era responsable de hacer todo
- La secuencia de boot era manual, lenta (podía durar unos 10-15 minutos) y era propensa a errores



Máquina Dedicada – Uso típico en el caso ideal

1. Cada persona debe configurar los tableros de control
2. Colocar FORTRAN en la lectora de tarjetas.
Bootear la computadora – lee FORTRAN
3. Colocar el programa de FORTRAN en la lectora.
Leer, Cuantas pasadas? Unas 2 ó 3 pasadas. Colocar de nuevo las tarjetas por cada pasada
4. Si no hay errores el compilador perfora un programa equivalente en ensamblador
5. Colocar ensamblador en la lectora de tarjetas. Boot a la computadora
6. Colocar programa generado por el compilador de FORTRAN en la lectora de tarjetas. Lo leer, varias pasadas?
7. Ensamblador genera un programa equivalente en lenguaje máquina.
Se usaban muchas tarjetas en esa época e IBM las vendía.
8. Se coloca el programa generado en la lectora de tarjetas. Se bootea la computadora. El programa corre
9. Recoger los datos generados de la impresora
10. Llega otra persona y repite todo el proceso de nuevo



Analizando la Máquina Dedicada



- Ventajas:
 - Muy emocionante
 - Control absolute del usuario
 - Pocos expertos
 - Era facil encontrar quien estaba usando la máquina cuando algo fallaba o se perdía
- Desventajas:
 - Trabajo repetido
 - No hay estandares, cada quien lo hacia a su manera
 - Los usuarios ser expertos en hardware para saber por que estaba fallando en el momento
 - Uso ineficiente de recursos

Mejorando la Máquina Dedicada

- Debido a la subutilización de recursos se inventa al operador de computadoras.
 - El programador llegaba dejaba las tarjetas y el operador se encargaba del resto.
- Se recolectan los trabajos y se factorizan pasos para hacerlo mas eficiente.
 - Ejemplo: se recogen todos los programas de FORTRAN, se pone a correr el compilador y se le pasan los programas. Mas rápido que estar corriendo el compilador cada vez que se va a pasar un programa.
- Nacen las configuraciones estándar
- Aparece el procesamiento en lotes. Procesamiento batch
- Mejora el rendimiento del centro de cómputo.
Throughput (cantidad de productos terminados por unidad de tiempo)
- Antes unas 5-6 personas usaban la computadora, ahora cambia a 100-200 por día



APUNTES

09/09/2021

COMPILADORES E INTÉPRETES

APUNTADOR:

Dennis Johel Angulo Fuentes 2018319143

QUIZ 01

1. Defina detalladamente los siguientes conceptos:

- a) ENIAC
- b) John Mauchly
- c) "First Draft" de von Neuman
- d) Maurice Wilkes
- e) EDVAC
- f) Ciclo fetch
- g) Productividad
- h) EDSAC
- i) Eficiencia
- j) Unidad de Control Alambrada

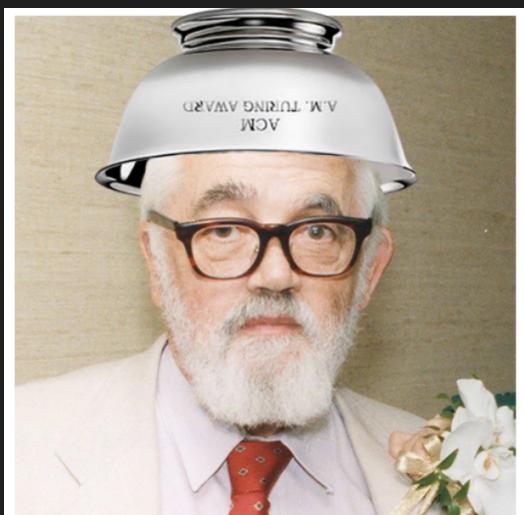
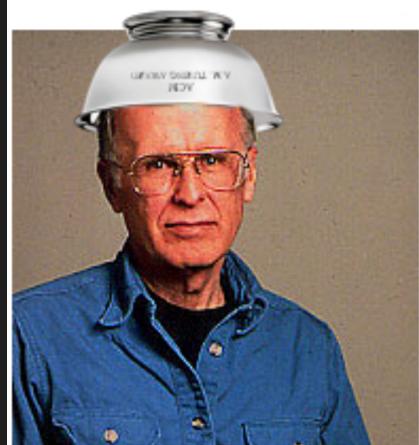


Hola Dios, soy yo de nuevo



John Backus (1924 - 2007)

- Matemático y Científico de la Computación, U.S.A
- M. Sc en Matemáticas - Columbia University
- En 1956 crea **FORTRAN**, primer lenguaje de alto nivel
- Ganó un Turing Award en 1977
- Intenta la **BNF (Backus Normal Form)**



John McCarthy(1927 - 2011)

- Matemático y Científico de la Computación, U.S.A.
- Ph. D. en Matemáticas - Princeton University
- Profesor en MIT, Stanford, Princeton
- Muy influenciado por von Neumann
- Pionero de la IA. En sus investigaciones de IA creía que los lenguajes que existían no eran apropiados
- Inventó **LISP**
- Inventó el concepto **garbage collection (Dejar a los objetos existir y luego barrer los que no se ocupen)**
- Trabajó en la definición de **ALGOL**
- Ganó un Turing Award en 1971

Grace Murray Hopper (1906 - 1992)

- Matemático y Científico de la Computación, U.S.A
- Ph. D. en Matemáticas - Yale University
- Contraalmirante de la Marina U.S.A
- Ganó un Turing Award en 1977
- Posiblemente la primera programadora de la historia
- ¿Primer compilador?
- Inventó **COBOL**
- Inventó el término **compilador y debugging**

No obtuvo un Turing Award ¿Misoginia? ¿Sus contribuciones no fueron gustadas por los académicos?
Doña Grace tuvo su venganza.



Peter Naur (1928 - 2016)

- Astrónomo y Científico de la Computación, Dinamarca
- Ph. D. en Astrofísica, usó computadoras para su tesis de doctorado y le terminaron encantado
- Después de graduarse se dedicó a la computación (¿Y qué con el Ph. D. en Astrofísica?)
- Hizo muchas contribuciones a la **BNF** y se cambió el nombre a la **Backus-Naur Form**
- Líder del Comité de **ALGOL 60**
- Hizo grandes contribuciones en lenguajes de programación, compiladores, algoritmos e Ingenería de Software
- Ganó un Turing Award en 2005



Curiosidades Del Turing Award

- Es un premio anual entregado por Association for Computing Machinery (ACM) a aquellos individuos cuyas contribuciones realizadas a la comunidad informática son relevantes en carácter técnico.
- Se le considera el Premio Nobel de la computación.
- Puede haber más de un ganador al año (generalmente porque trabajan juntos).

Los 4 Grandes



- Entre 1956 - 1960
- Muy importantes históricamente:
 - Primeros en hacer lenguajes de alto nivel.
 - Estos lenguajes fueron exitosos
 - Influencia enorme en el desarrollo de lenguajes de alto nivel, compiladores, técnicas de desarrollo de software

FORTRAN

COBOL

LISP

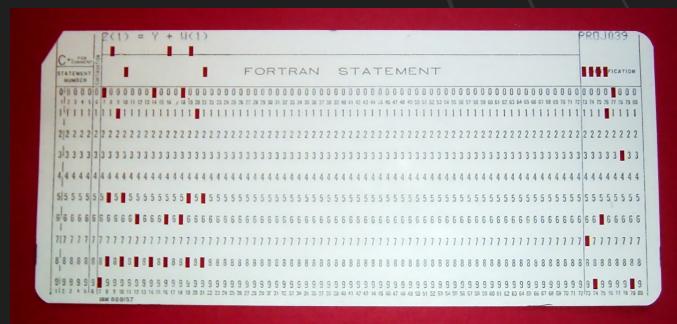
ALGOL

FORTRAN

- Backus, con toda la flojera del mundo, quería dejar de convertir fórmulas matemáticas a ensamblador, así creó **FORmula TRANslation**

- Utilizaba tarjetas perforadas
- Creado en los 50s en IBM por Backus
- Primer L.A.N de uso generalizado
- Ha evolucionado:
 - FORTRAN
 - FORTRAN II, FORTRAN III, FORTRAN IV
 - FORTRAN 66, FORTRAN 77, FORTRAN 90, FORTRAN 95
 - FORTRAN 2003, FORTRAN 2008, FORTRAN 2018

- Todavía se usa bastante.
- Favorito en HPC (Debido a que muchos de los clientes de HPC son físicos cuyos trabajos estaban hechos en FORTRAN).
- Las últimas versiones de FORTRAN incluyen POO.



Dato interesante: Se indexaba con (), ej: ARR(5)



En esta clase todos nos volvimos monstruos peludos en FORTRAN



LISP



- **LISP Processor**
- Desarrollado por McCarthy (M.I.T) en los 50s
- Usa notación prefija y muchos paréntesis, demasiados
- Es interpretado
- Orientado a IA
- Lenguaje precursor en programación funcional
- Todo son listas, tanto los datos como el código.
- Derivados de LISP: Scheme, Common Lisp, Clojure
- Influencia en el desarrollo de:
 - Python
 - Lua
 - Ruby
 - Haskell
 - Smalltalk



Dato interesante: Los comentarios se agragaban entre """

En esta clase todos nos volvimos monstruos peludos en notación prefija y LISP

COBOL

- **Common Business-Oriented Language**
- Durante la presente pandemia se generó mucho trabajo para programadores de COBOL
- Desarrollado por Doña Grace en 1959
- Doña Grace quería un lenguaje donde básicamente se escribiera en inglés, así los programadores no tendrían que estudiar para saber COBOL.
- Ventajas:
 - Fácil de programar
 - El propio código es la documentación
- COBOL es muy querido por las industrias, pero no por la academia
- Los programas en COBOL eran muy grandes
- Muchas de las palabras en inglés estaban reservadas
- **Legacy code.**



Datos interesantes: Porcentaje de programas escritos en COBOL mundialmente.

- 1997 => 80%
- 2005 => 75%
- 2012 => 60%

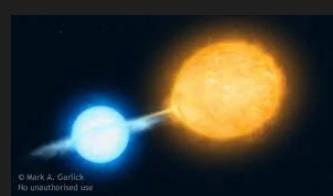
Esta fue la venganza de Doña Grace

En esta clase todos nos volvimos monstruos peludos en COBOL

```
begin
    real x;
    integer i,j;
    for i := 2 until Z do begin
        x := Y[i];
        for j := i-1 step -1 until 1 do
            if x >= A[j] then begin
                A[j+1] := x; goto Found;
            end else
                A[j+1] := A[j];
        A[1] := x;
    Found:
        end
    end
end Sort
```

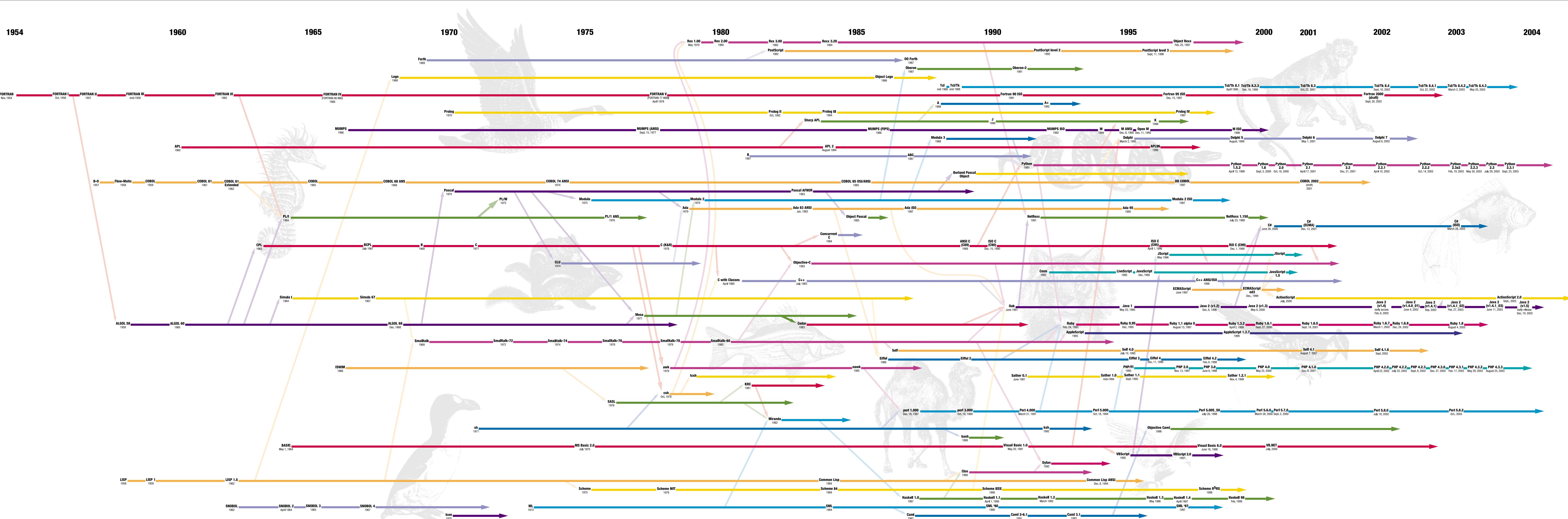
- **ALGOrithmic Language**
- Diseñado por un comité encabezado por Peter Naur en 1958
- Inspirados en FORTRAN, quisieron resolver los defectos
- Primer lenguaje descrito en BNF
- Se crearon conceptos que repercuten en la programación actual:
 - bloques **begin - end**
 - Llamadas por valor y referencia
 - variables locales
 - **then / else**
- El anti-COBOL (amado por académicos, ignorado por la industria)
 - ALGOL 58, 60, 68, 73
- De cierta manera todos programamos en ALGOL. Familia "ALGOL-like languages":
 - BCPL, B, Pascal, PL/1, Simula, C, C++, Java, etc.

Dato interesante: Hay una estrella que lleva el mismo nombre



History of Programming Languages

O'REILLY®



www.oreilly.com

For more than half of the fifty years computer programmers have been writing code, O'Reilly has provided developers with comprehensive, in-depth technical information. We've kept pace with rapidly changing technologies as new languages have emerged, developed, and matured. Whether you want to learn something new or need answers to tough technical questions, you'll find what you need in O'Reilly books and on the O'Reilly Network.

meline includes fifty of the more than 2500 documented
amming languages. It is based on an original diagram created
c Lévénez (www.levenez.com), augmented with suggestions
O'Reilly authors, friends, and conference attendees.

formation and discussion on this poster,
www.oreilly.com/go/languageposter.



Compiladores e Intérpretes

Introducción

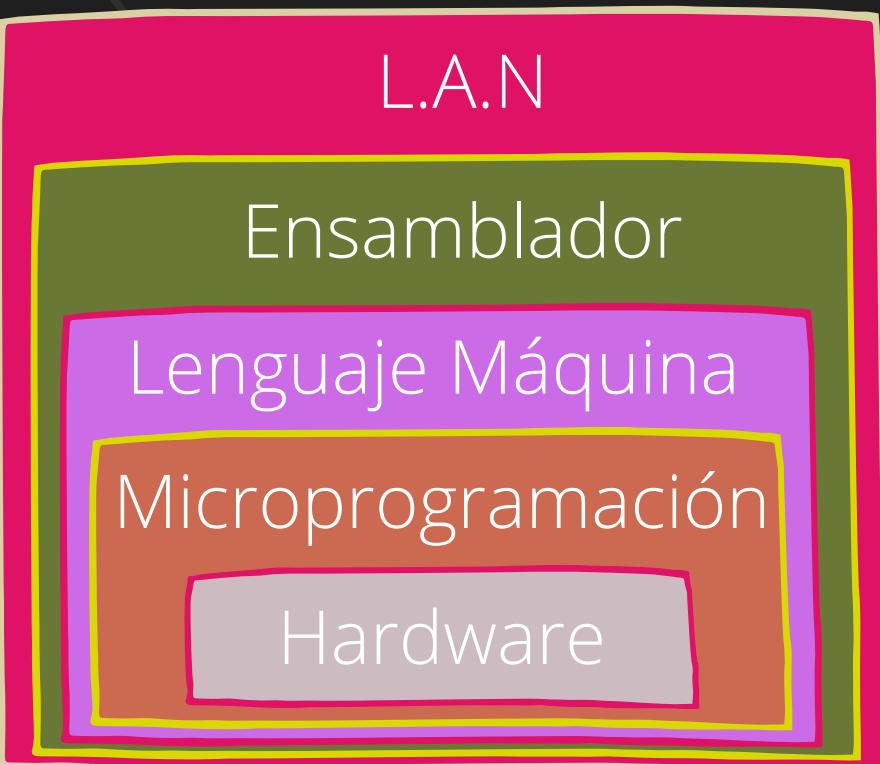
Niveles de Máquinas

L.A.N

- Definen una nueva capa
- Son independientes del hardware
- Ya no se piensan en la capa de abajo, el hardware inferior se vuelve irrelevante
- Portabilidad
- Abstracción de programación, ya no se piensa en ensamblador, ahora se piensa en el lenguaje que se trabaja (C, C++, Java, Python)



Productividad vs Eficiencia



IBM 701

- Primera computadora comercial (Abril 1952)
- IBM dominaba el mundo computacional. Solo se fabricaron 19
- Parte de las configuraciones eran:
 - 4096 palabras de 36 bits
 - 150 mil inst./misisipi
 - 150 tarjetas/misisipi (I)
 - 100 tarjetas/misisipi (O)
- Se alquilaba en \$23,750 al mes (\$227,728 actuales)
- MTBF: 30 minutos (Dinerito pa IBM)

1 misisipi = 1 seg

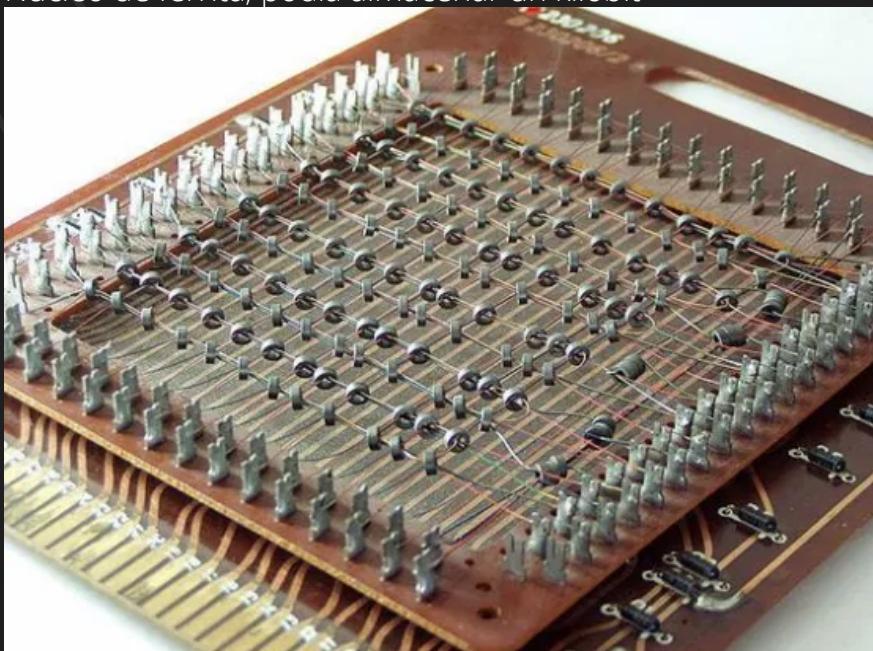


IBM 704



- Primera computadora con punto flotante (1954 - 1960)
- Se vendieron o alquilaron unas 200 máquinas.
- Se alquilaba en \$35,550 al mes (\$340,874 actuales)
 - Era el doble de rápida que la 701 pero incompatible
- Dispositivos para copiar tarjetas a cinta off-line
- Memoria de núcleos de ferrita
- MTBF: 8 horas
- Venía con FORTRAN

Núcleo de ferrita, podía almacenar un kilobit



Máquina Dedicada

- Si usted quería usar la máquina, debía reservarla por "horas de máquina"
- El programador / científico / ingeniero era responsable de hacer todo, ellos hacían lo que quería con la computadora
- El booteo era manual, lento y propensa a errores

Uso Típico (caso ideal)

1. Configurar todos los tableros de control.
2. Colocar FORTRAN en lectora de tarjetas. Dar boot al computador.
3. Colocar fuente FORTRAN en la lectora. Leer. ¿Varias pasadas?
4. Si no hay errores, el compilador perfora un programa equivalente en ensamblador.
5. Colocar Ensamblador en lectora de tarjetas. Dar boot al computador.
6. Colocar ensamblador generado por compilador en lectora de tarjetas. Leer. ¿Varias pasadas?
7. Ensamblador perfora programa equivalente en lenguaje máquina.
8. Colocar lenguaje máquina en lectora de tarjetas. Dar boot al computador. Nuestro programa corre.
9. Recolectar listados de impresora.
10. Siguiente usuario, GO TO 1.

Analizando la Máquina Dedicada

- Ventajas:
 - Absoluto control del programador
 - Pocos expertos al inicio
 - Fácil establecer responsabilidades y controles
- Desventajas:
 - Trabajo repetido entre un usuario y otro
 - No hay estándares de configuración
 - Programadores deben ser expertos en hardware
 - Uso muy ineficiente de recursos



Mejorando la Máquina Dedicada



- Idea administrativa:
 - Se crea el operador de computadoras.
¡Fuera programadores!
- Se recolectan trabajos y se factorizan pasos (Se corren todos los programas que comparten características juntos)
- Configuraciones estándar
- Procesamiento en lotes. **Procesamiento batch**
- El rendimiento del centro de computación aumentó (**Throughput**, cantidad de productos terminados por unidad de tiempo)

To Be Continued

APUNTES

11-9-2020

Operadores
Sistemas Operativos
Intro. Compiladores
DiagramasT





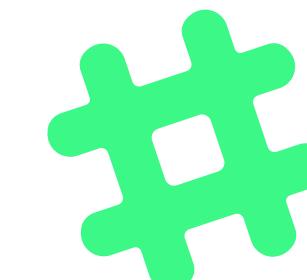
- 1 Medida administrativa
- 2 Factorización de pasos
- 3 Procesamiento batch (lotes)
- 4 Inicio de Config. Estándar

Máquina dedicada

Batch con Operador



- ✓ Solución al trabajo repetitivo
- ✓ Surgen más compiladores
- ✓ Reducción de errores de hardware
- ✓ El software selecciona el compilador

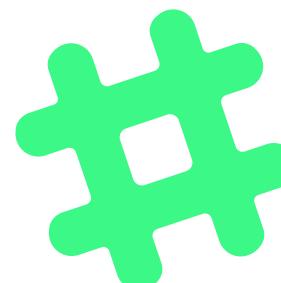
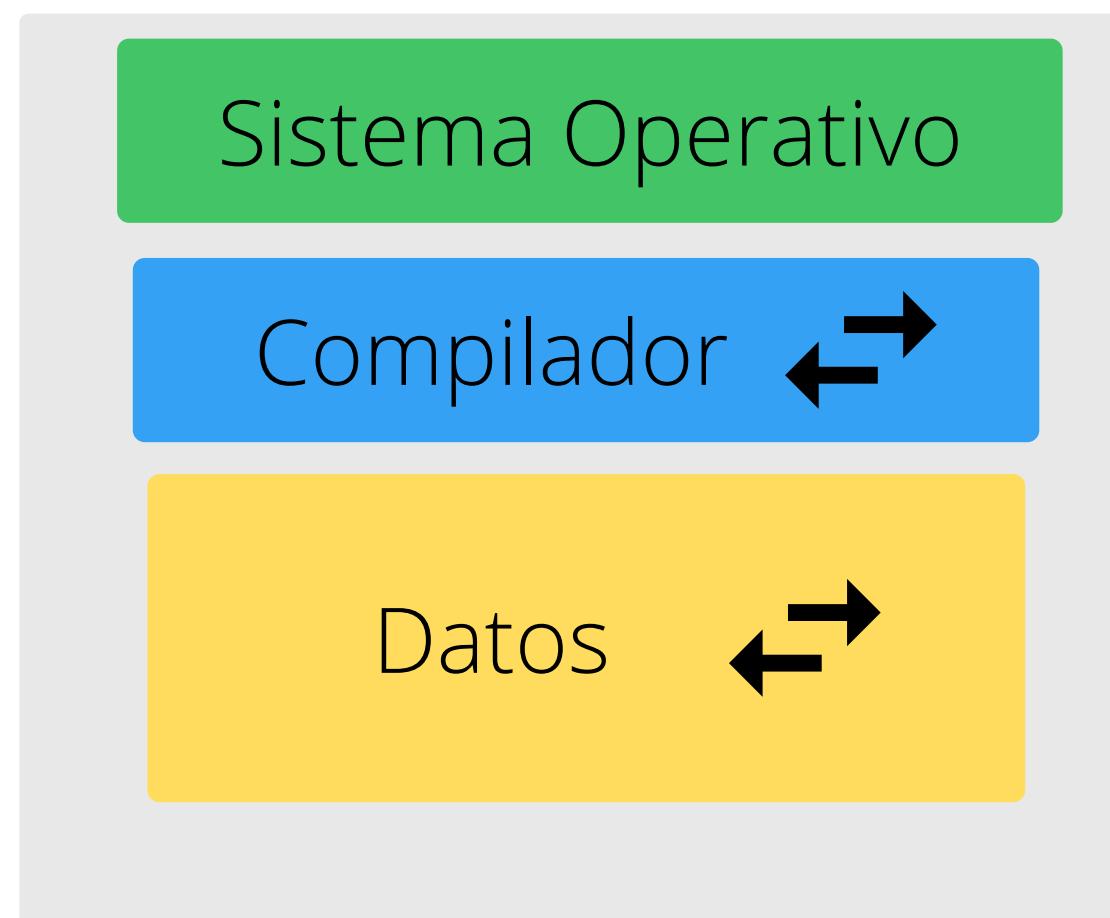


Cuando llegan los S.O. ya existían las máquinas y lenguajes de alto nivel

SISTEMAS OPERATIVOS



Distribución de la Memoria



Las tarjetas especificaban el compilador donde corría el programa, así como el inicio y el fin del mismo

Job Control Language(JCL)

- | | | |
|---|--|--|
| 1 | 2 | 3 |
| Primer SO
Interfaz de Usuario | Aún con tarjetas
Intercaladas | Antecesor de los
"Shell Languages" |
| 4 | 5 | 6 |
| Era "super" dedicado
y muy cambiante | Era propiedad de IBM,
ellos \$\$capacitaban\$\$ | Ocupaba operador
pero menos trabajo |

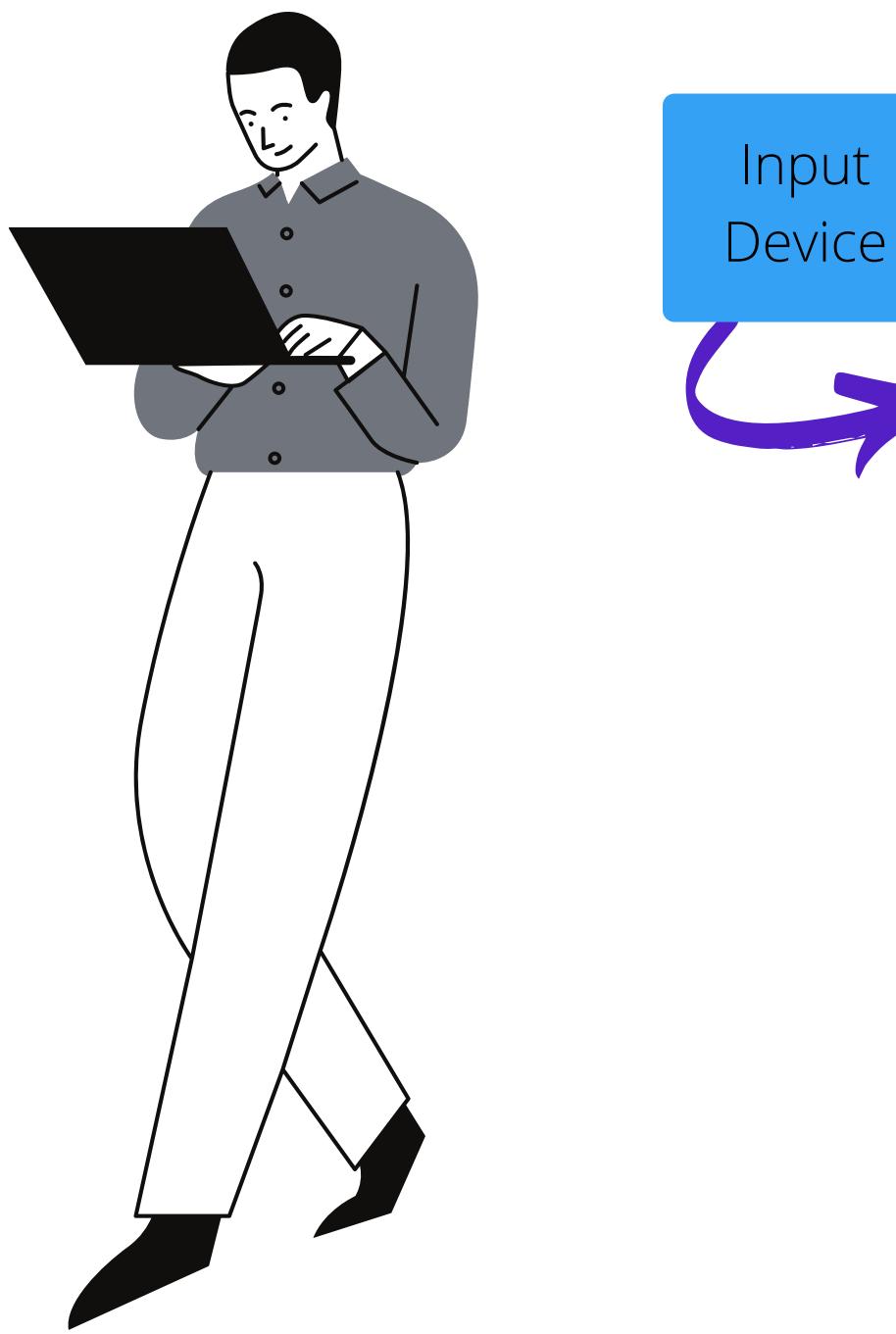
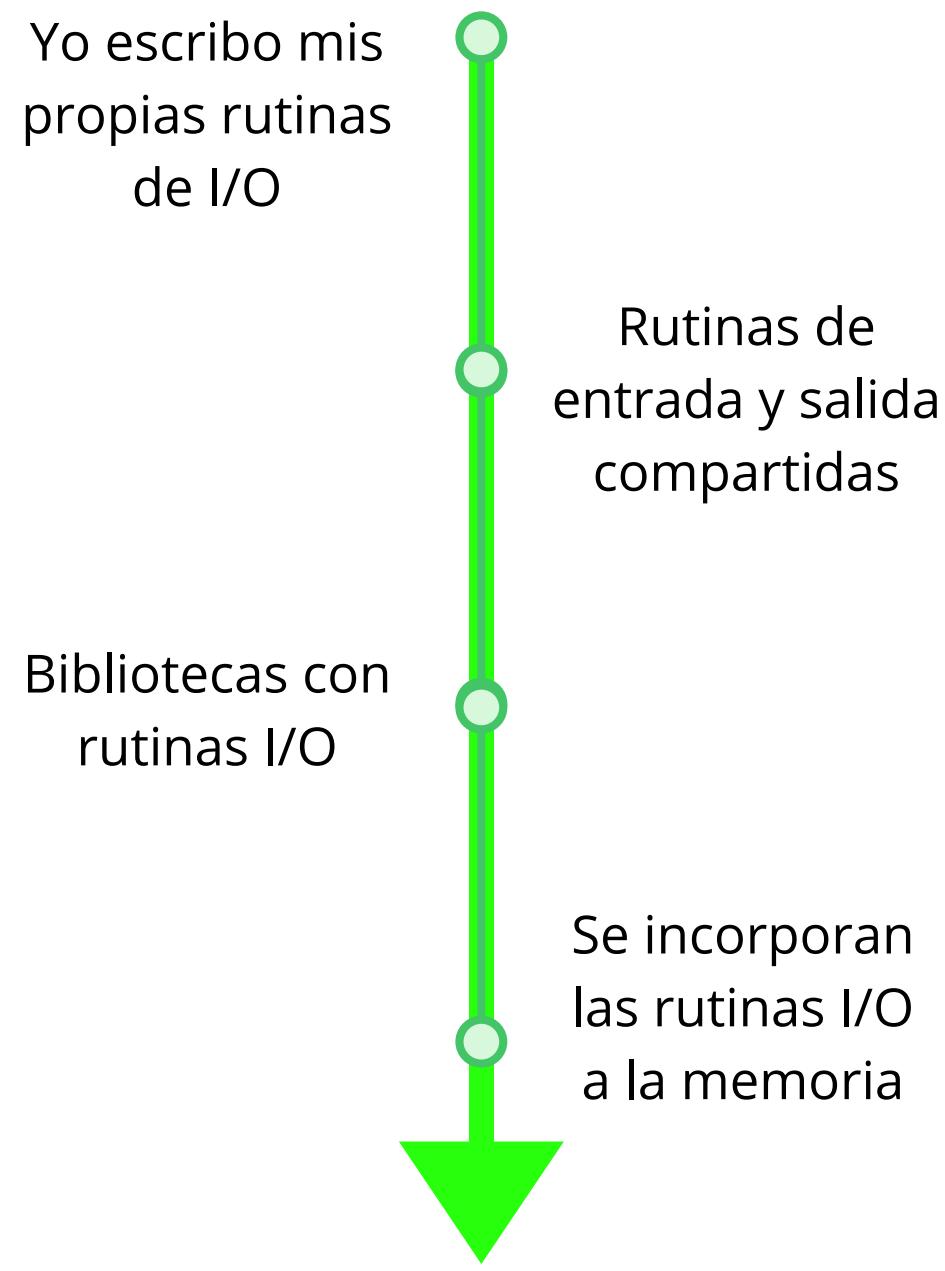
TH
RO
UG
HP
UT

- Máquina dedicada
- Batch con Operador
- Batch con SO

SISTEMAS OPERATIVOS

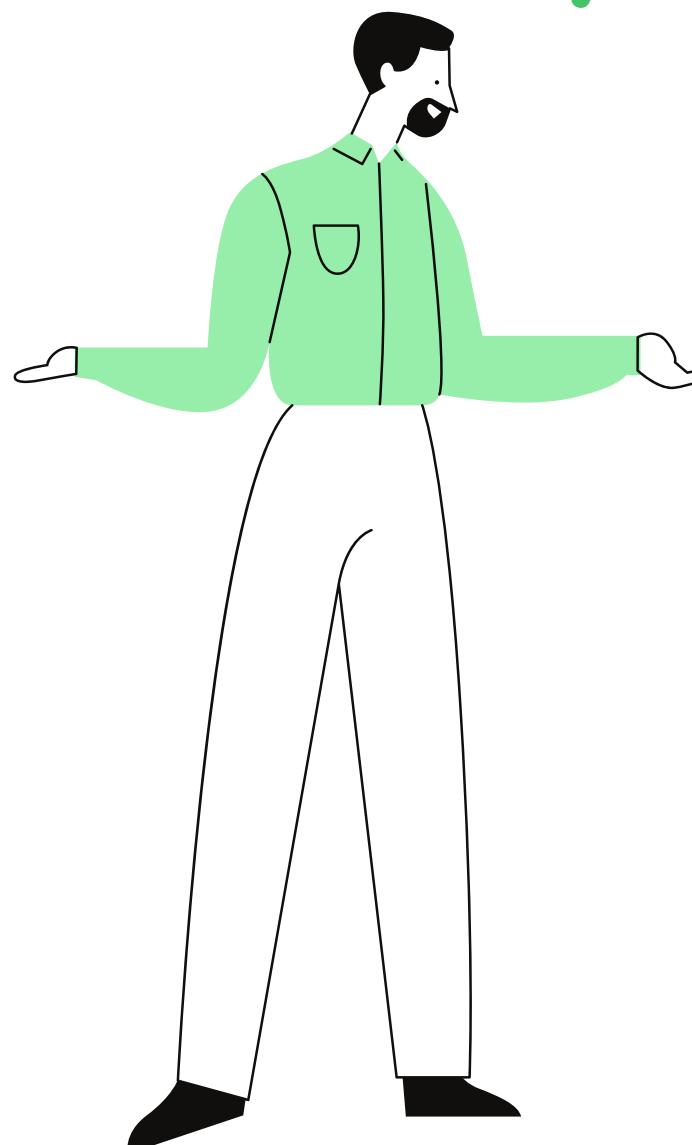


Input Output Timeline



Von Neumann sugiere un espacio de direcciones para E/S => Instrucciones Leng. Máquina

**Por qué no tener
I/O siempre en
Memoria?**



Cómo?

Cuando un programa corre, le pide al sistema operativo (en memoria) que le preste operaciones de entrada y salida



Operaciones I/O
optimizadas para el SO



Vínculo entre programa
y Sistema Operativo

Device Drivers!

Rutinas que son parte del sistema operativo que sabe usar varios dispositivos de hardware.

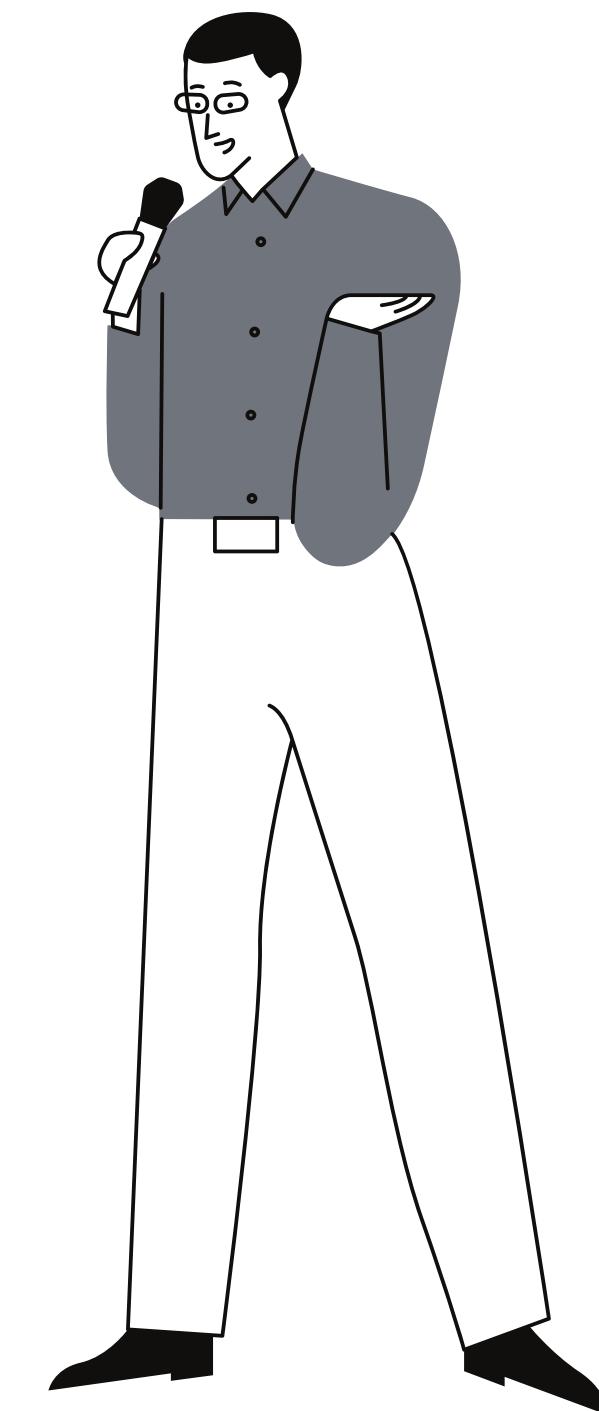
Solo entrada y salida?

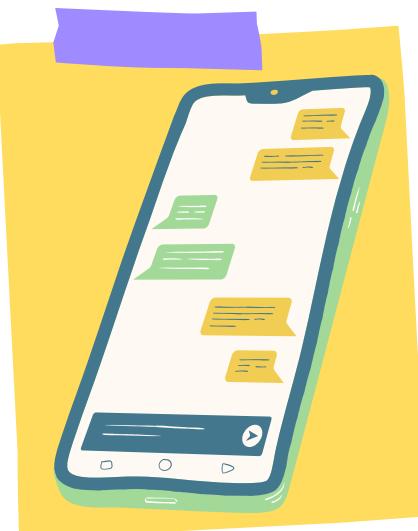
Disponibles
a los programas
iSystem Calls!
(INSTRUCCIONES
VIRTUALES)

Vinculo entre Sistema Operativo y Programa

Ocasionaron un crecimiento del sistema Operativo

Menos interacción con el hardware!!





Hardware

Celular funcionando como teléfono en una llamada.
Simulación

Proceso de envío de un
mensaje de texto desde
un celular hasta una
torre de comunicaciones

El sistema operativo se
encarga del envío de un
mensaje de Whatsapp

Real

Lo que **se ve** y
existe

Virtual

Lo que **se ve**,
pero **no existe**

Transparente

No se ve, pero
existe

Memoria virtual: no existe,
solo se simula, incluso se
puede usar más de lo que se
tiene disponible
físicamente

Ing. en
Computación

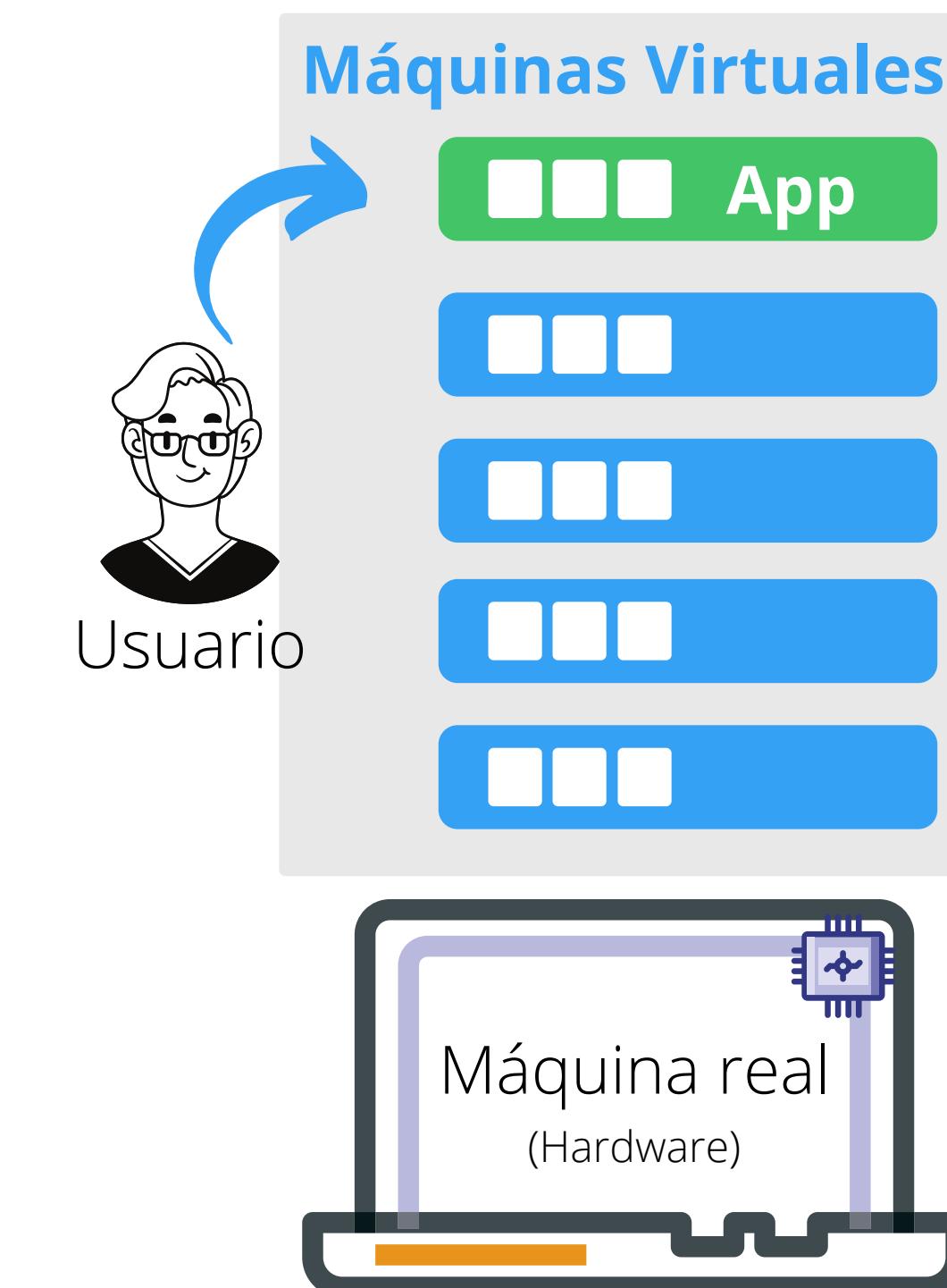
Es un sistema que
graba todo en disco y
solo saca lo que ocupa.

S. Operativo como Maquina Virtual

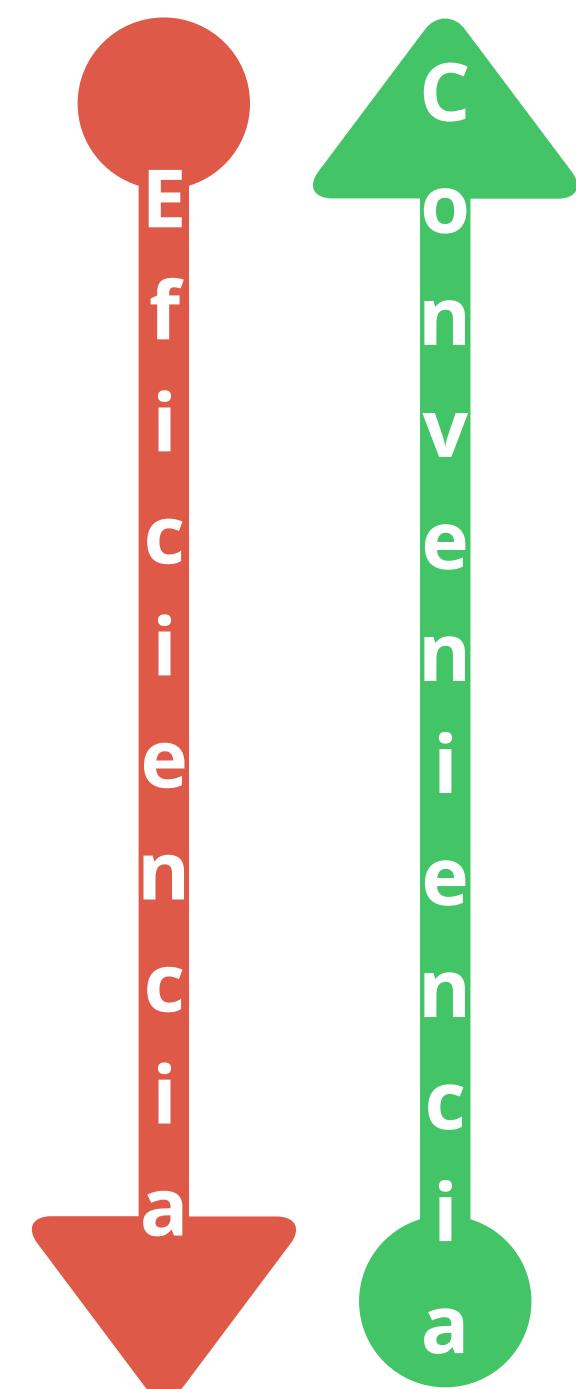


Máquina
Dispositivo(real, virtual o transparente) que realiza una tarea computacional

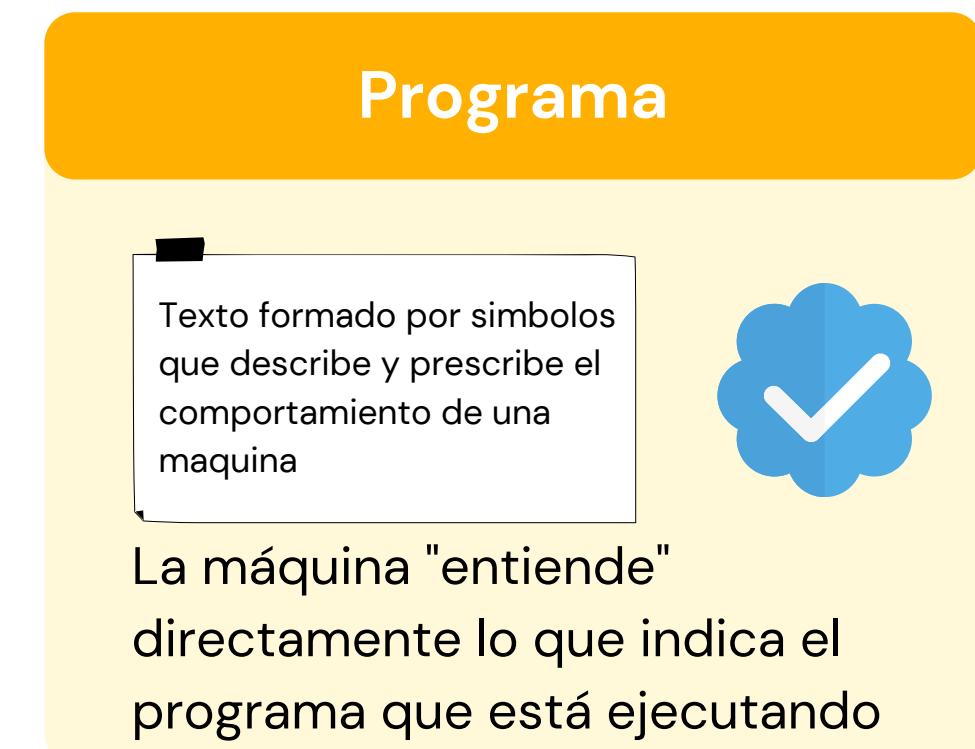
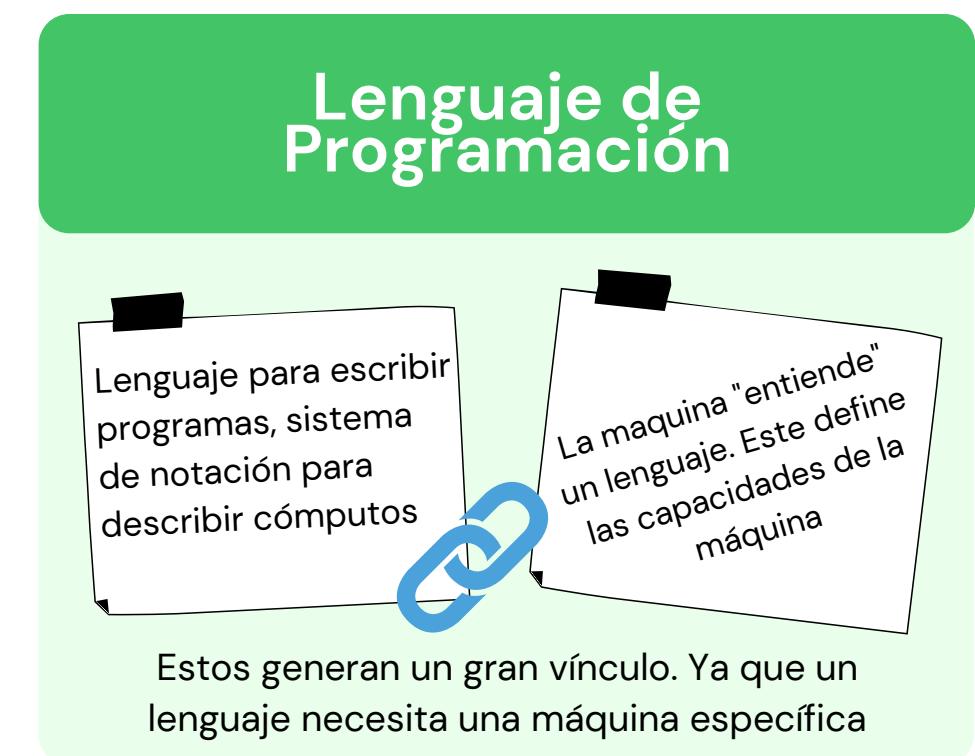
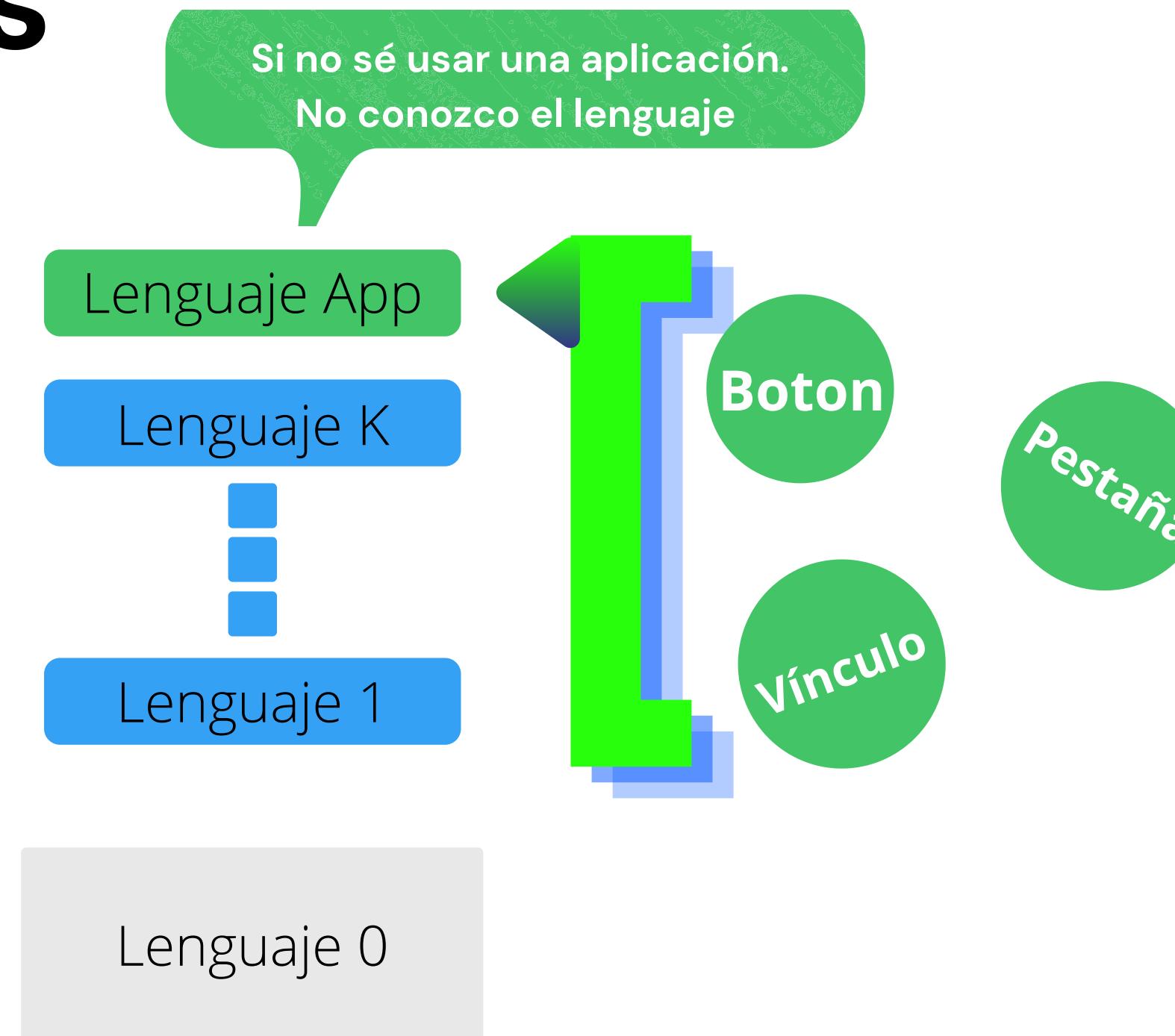
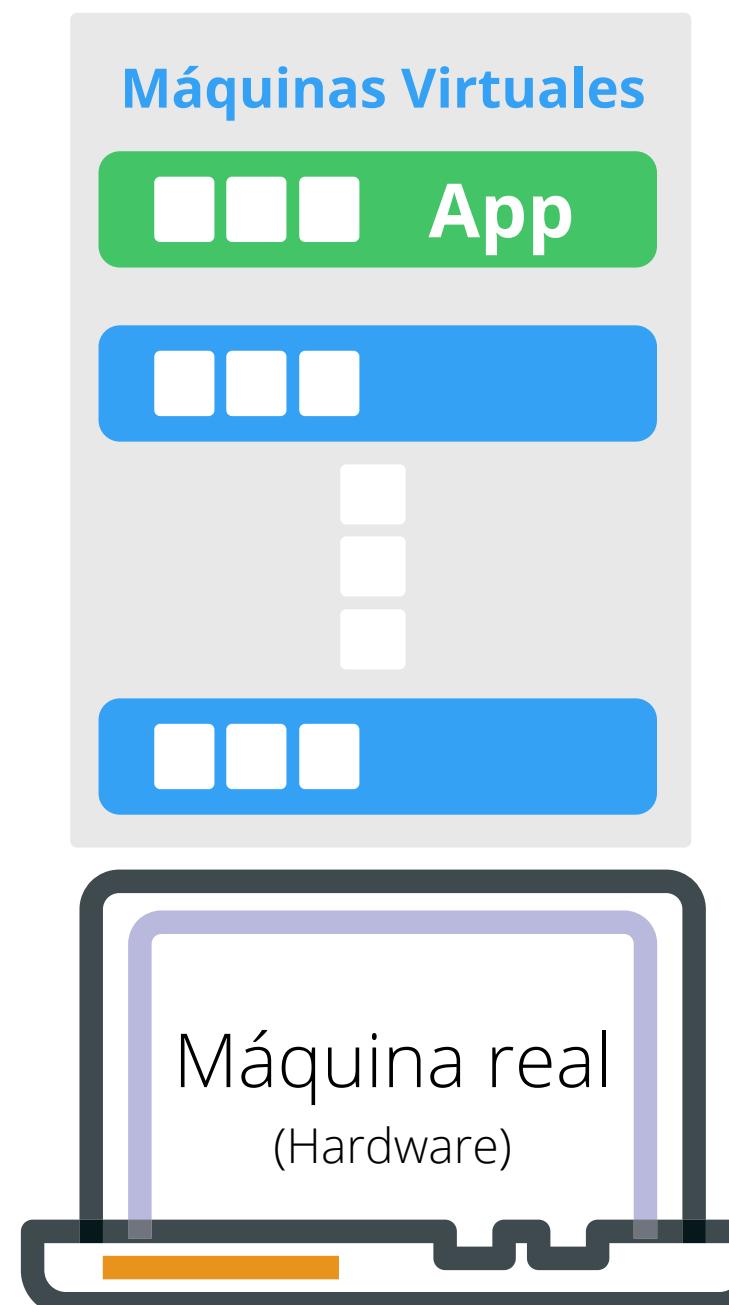
Dos SO distintos no pueden correr el mismo programa. Ya que los System Calls que utilizan son distintos.



Generalización



Máquinas y lenguajes



Desarrollo "Virtual"



Máquina Virtual



Diseño del lenguaje que la define.

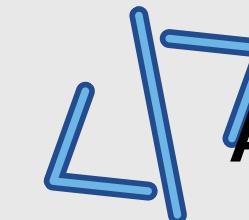


Se crean sobre la anterior y supone que esa es real.



Lograr que el lenguaje sea entendido por otra máquina.

Los lenguajes tienen la desventaja de que no pueden correr directamente en el de más abajo.



Aplicaciones



Diseño del lenguaje de aplicación



Entendimiento de este lenguaje por una máquina

Mapeo de Lenguajes

1

Traducción

- Convierte L en Li-1
- Traducción Completa
- Ejecución Rápida
- Traduce 1 vez
Usa N veces

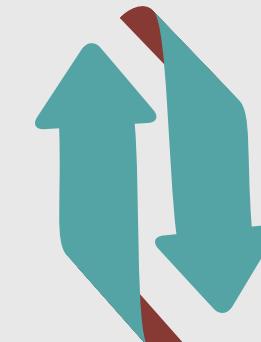
2

Interpretación

- Instrucción por Instrucción
- Simula ejecución en Mi-1
- No hay traducción
- Traduce N veces

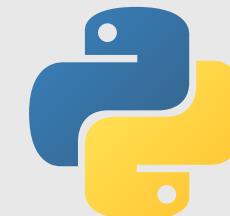
Compilador

Software traductor, toma lenguaje de una VM y lo saca en el lenguaje de una distinta. Produce errores!!!



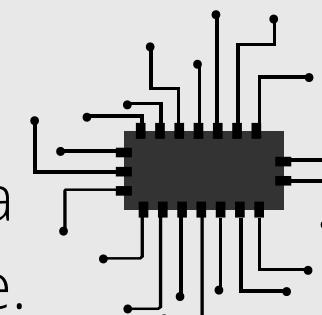
Intérprete

Software simulador, toma lenguaje de una VM y simula su ejecución en otra Maquina Virtual.



Computadora

Intérprete implementado en hardware. Probablemente cuenta con circuitos que lo hacen posible.



Siempre existen 3 lenguajes asociados a un compilador, para exemplificar como y donde se aplica cada uno se utilizan los diagramas T.

GCC

- 1 Es el estándar para GNU, Linux, etc.
- 2 GNU C Compiler
Luego → GNU Compiler Collection
- 3 Recibe lenguaje C, y lo traduce a x86 por medio de x86



Diagramas T

S

Lenguaje fuente:
(source) entrada que
debe ser traducida

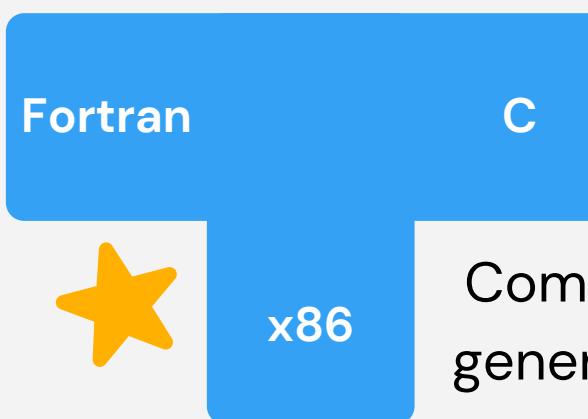
T

Lenguaje objeto
(target): lenguaje al
que se debe traducir

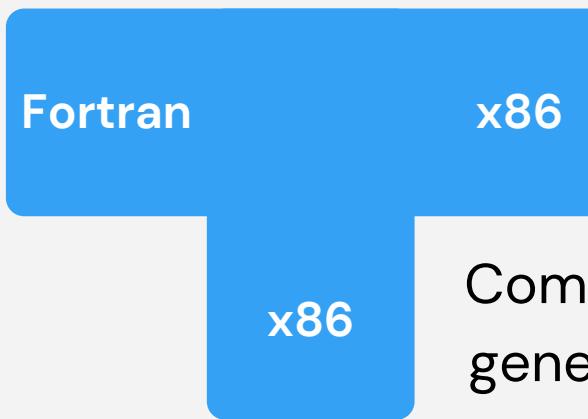
H

Lenguaje de
implementación
(host): lenguaje en el
que está escrito el
compilador

Ejemplos

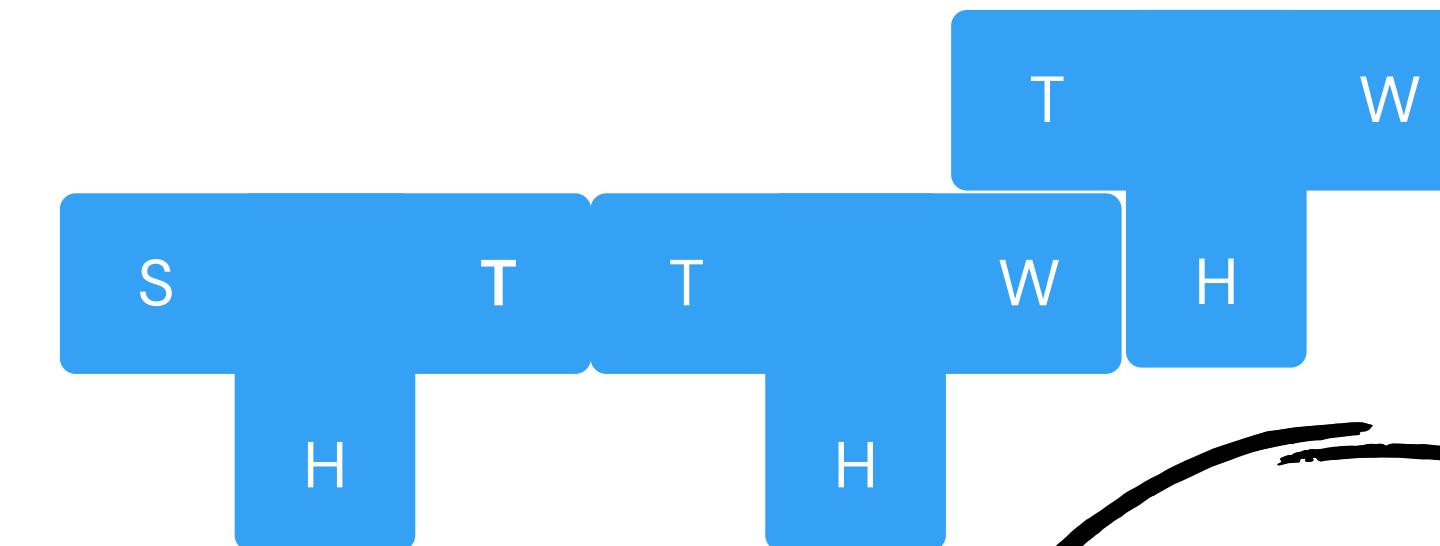


Compi. de Fortran que genera C o C++, escrito en x86



Compi. de Fortran que genera x86, escrito en x86

Notación y Operaciones

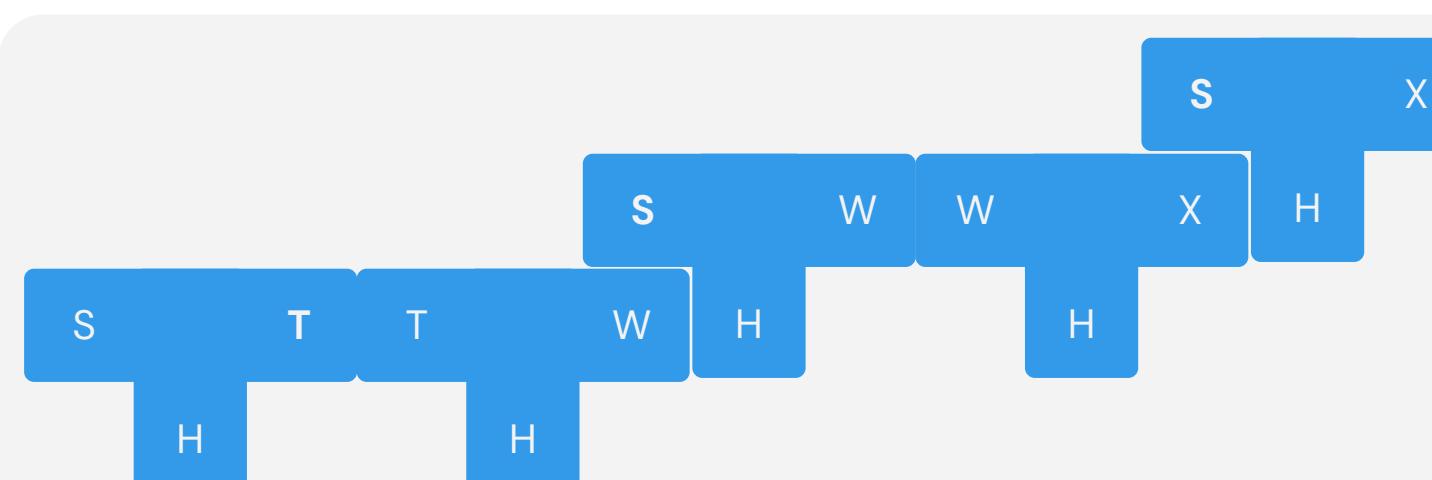
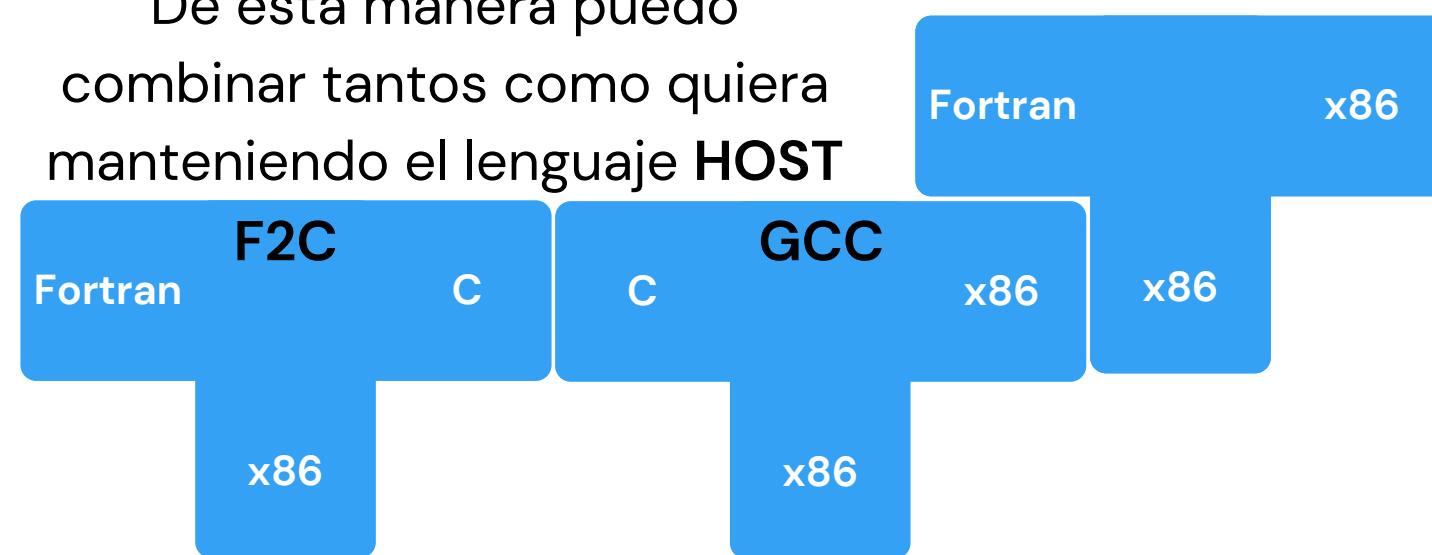


La unión de los compiladores 1 y 2 producen

Compilador que recibe T, genera W y está escrito en H

Notación y Operaciones

De esta manera puedo combinar tantos como quiera manteniendo el lenguaje **HOST**

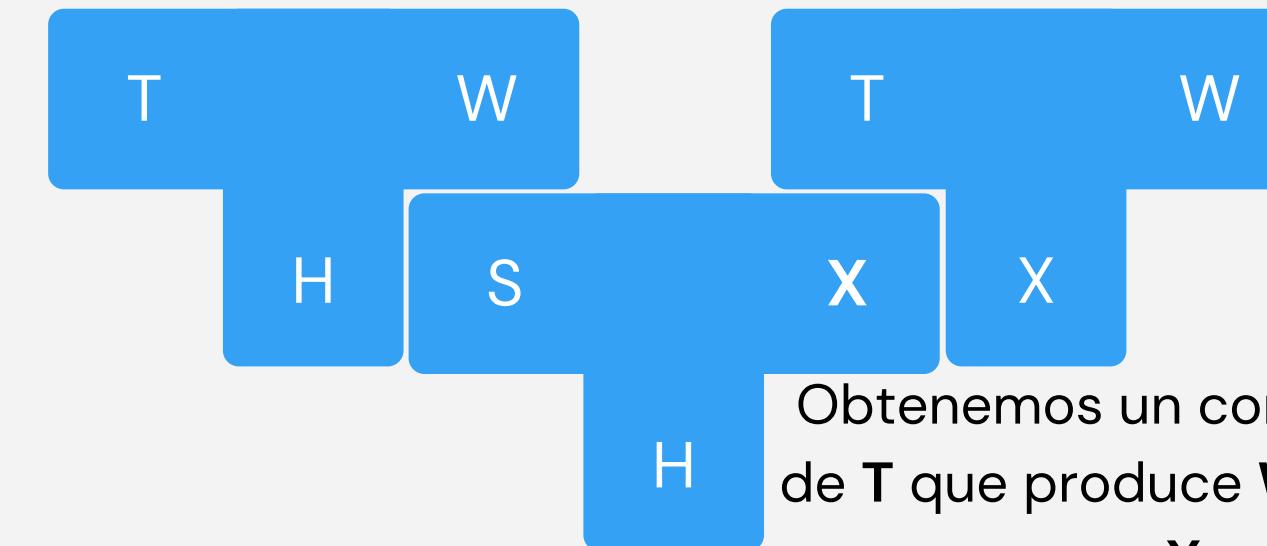


De manera que obtendría un compilador de **S** que está escrito en **H** y produce **X**

Cambio de host



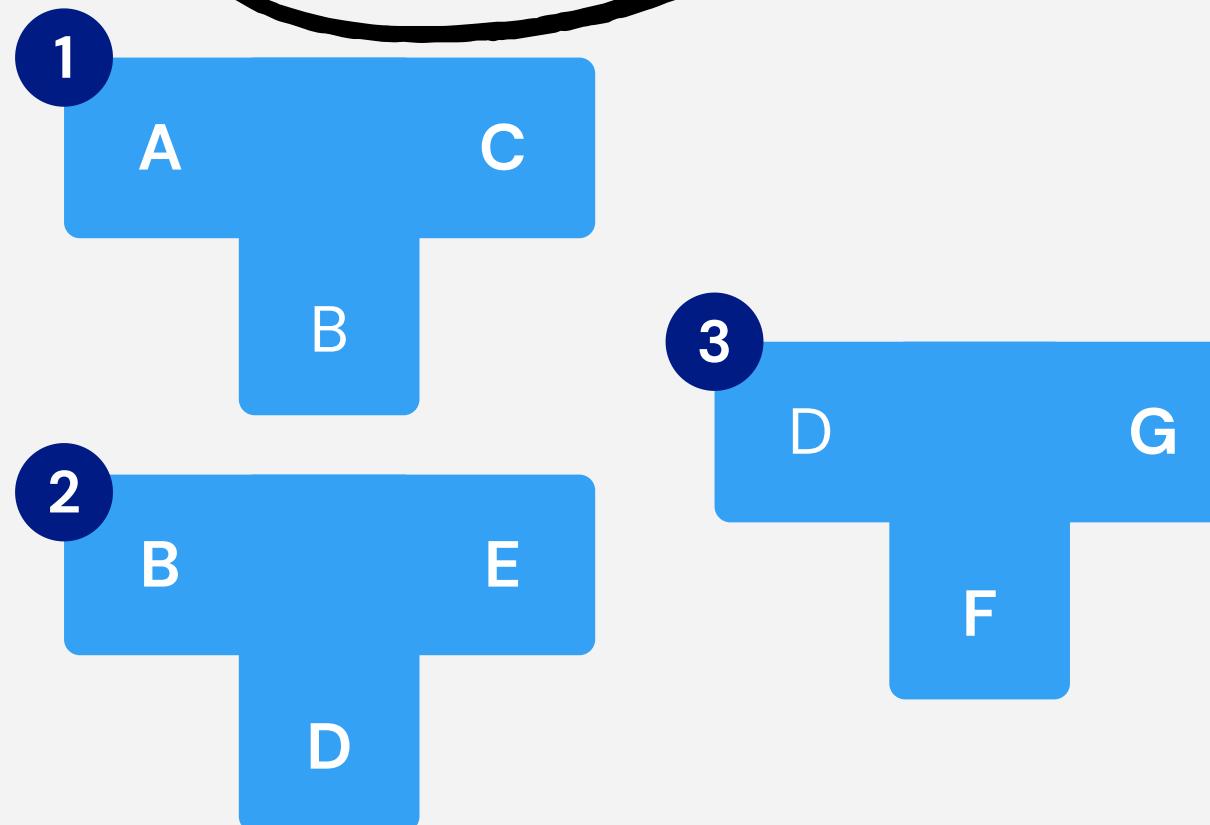
“ En este caso el compilador 1, ve un cambio en el lenguaje de implementación. Esto se denota de la siguiente manera:



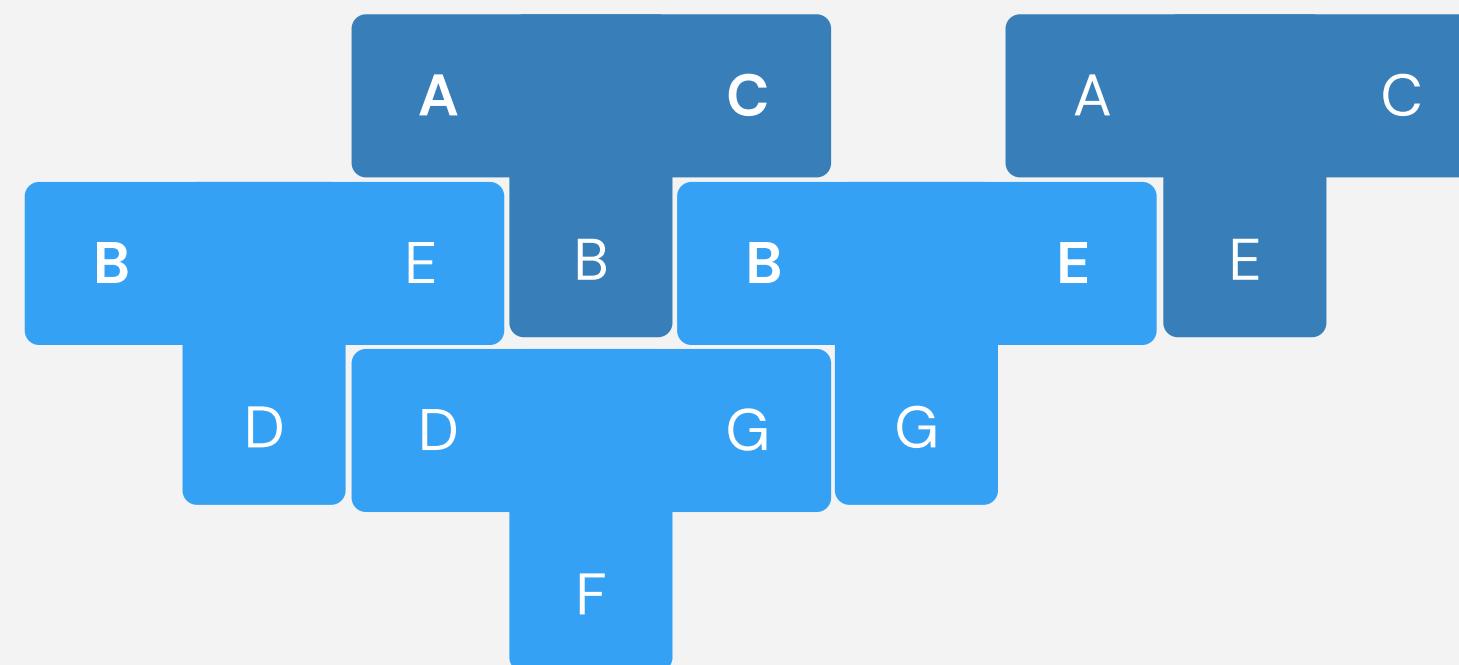
Obtenemos un compilador de **T** que produce **W** escrito en **X**

Notación y Operaciones

Utilizando los métodos mencionados anteriormente puedo realizar operaciones como la siguiente:

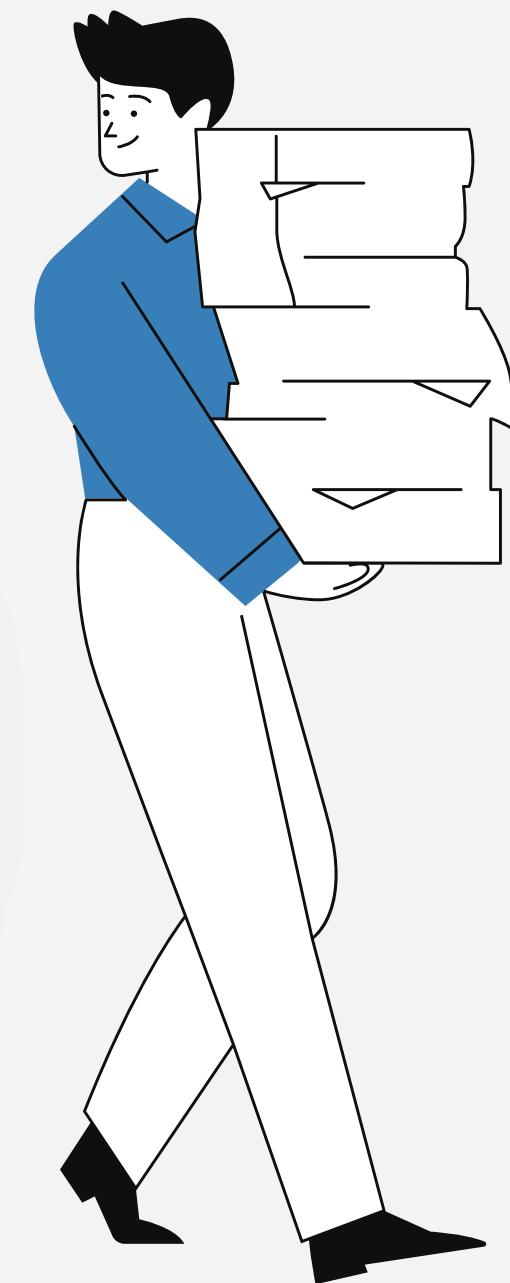


“En la parte superior tenemos al compilador 1, que ve su lenguaje de implementación encadenado a otro traductor. Esto genera un **Compilador de A** que produce **C** escrito en **E**.



“En la parte inferior encontramos al compilador encargado de traducir el lenguaje de implementación superior. Este es una modificación del **2** mediante el **3**, un **Compilador de B** que produce **E** escrito en **F**.

QUIZ EL MIÉRCOLES



Apunte del 11/09/2020

COMPILEDORES E INTÉPRETES
PROF. FRACISCO TORRES

REALIZADO POR

Marian Soza Hidalgo

SOFTWARE

REEMPLAZA AL OPERADOR

Sistemas operativos primitivos



- Software para reemplazar al operador.
- Menos fallas de hardware.
- Termina el primer programa y pone el siguiente.
- Tiene que haber algo que hable con el SO para saber cómo separar los programas (JCL).



- 1) Seleccionar compilador
- 2) Compilar
- 3) Ejecutar
- 4) Guardar resultados



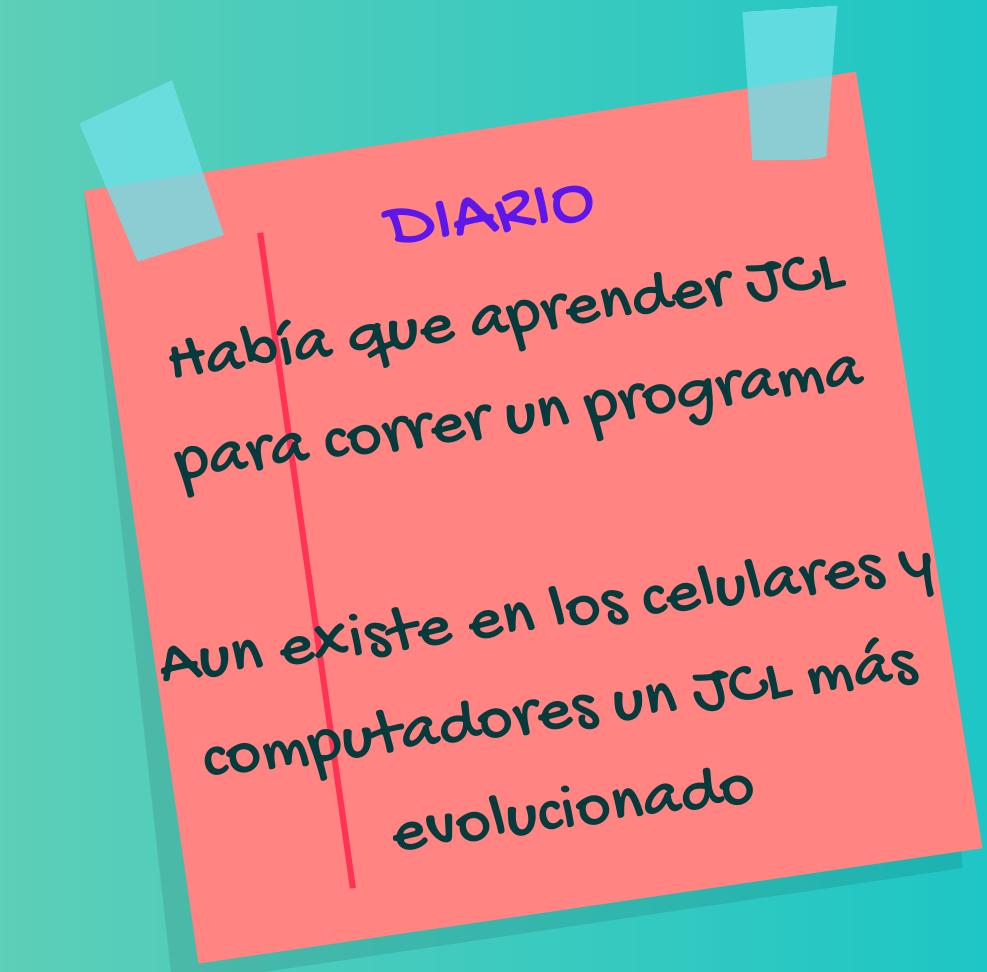
Los lenguajes de alto nivel llegaron
antes que los SO primitivos.
(cobol)



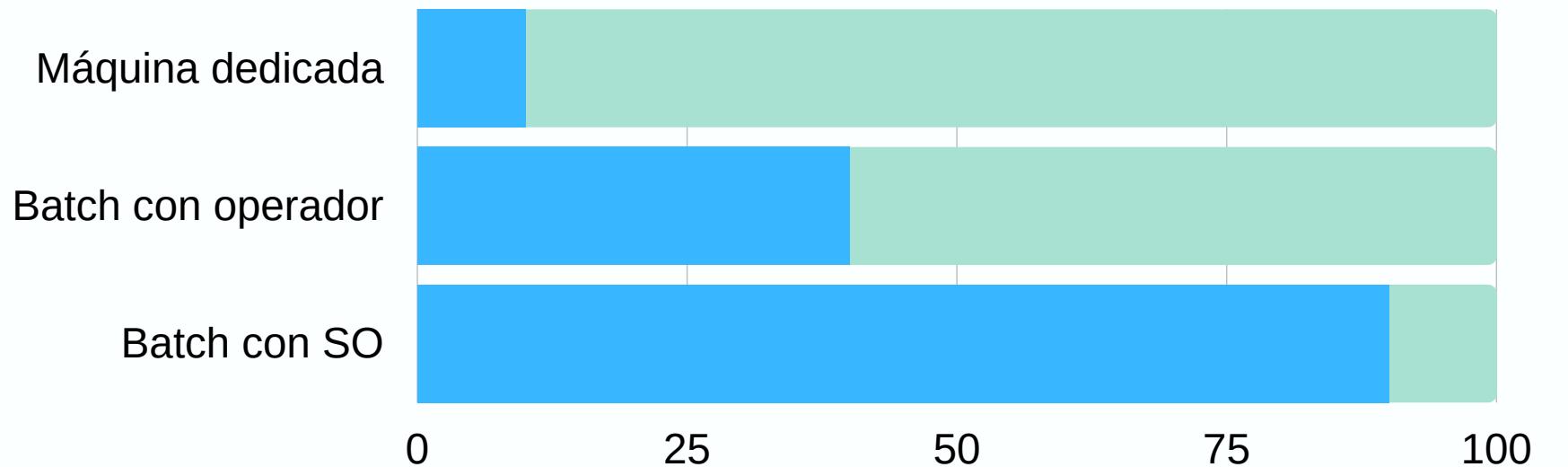
JCL

JOB CONTROL LENGUAJE

- Primer interfaz usuario
- Primer sistema operativo
- Antecesor de interfaces gráfica y lenguajes "shell"
- Sintaxis primitiva



PRODUCTIVIDAD: CENTRO DE CÓMPUTO



Batch: conjunto de trabajos de características similares.

Throughput: Cantidad de trabajos terminados por unidad de tiempo.

Entrada/Salida

+ SO

Primitivo

- Arquitectura Von Neuman
- Instrucciones en lenguaje máquina
- Había que ponerlo en **TODOS** los programas
- Se comunicaba con el SO para pedir scan/print
- Era responsabilidad del programador añadir todos los detalles.
- Cada dispositivo nuevo era más complejo que otro.
- Necesitábamos algo más flexible.

- Si siempre se necesita entonces que esté siempre en memoria.

VENTAJAS:

- Sube productividad, subrutinas E/S probadas y confiables, no hay que estarlas cambiando segun el dispositivo.
- Operaciones de E/S en el SO se llaman **Device Drivers**

Device Drivers se generalizaron, no solo para E/S. Pertenecen a los System Calls.

System Calls: Servicios que proporciona el SO a los programas.

DEFINICIONES

MÁQUINA

Dispositivo (real, virtual o transparente) que lleva a cabo una tarea computacional.

REAL

Se ve y existe

TRANSPARENTE

No se ve pero existe

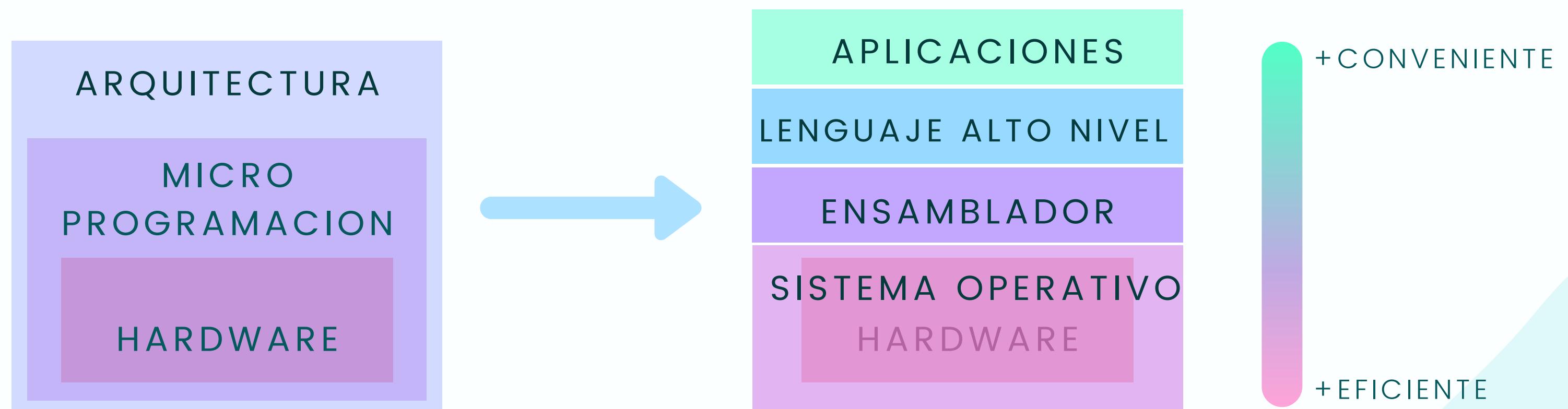
VIRTUAL

Se ve pero no existe

SISTEMA OPERATIVO COMO MÁQUINA VIRTUAL

Máquina que vemos pero no existe.

Idea tan exitosa que causó una redefinición de capas:



NOTA: El SO quita el hardware pero nos da conveniencia y servicios

LENGUAJE DE PROGRAMACION

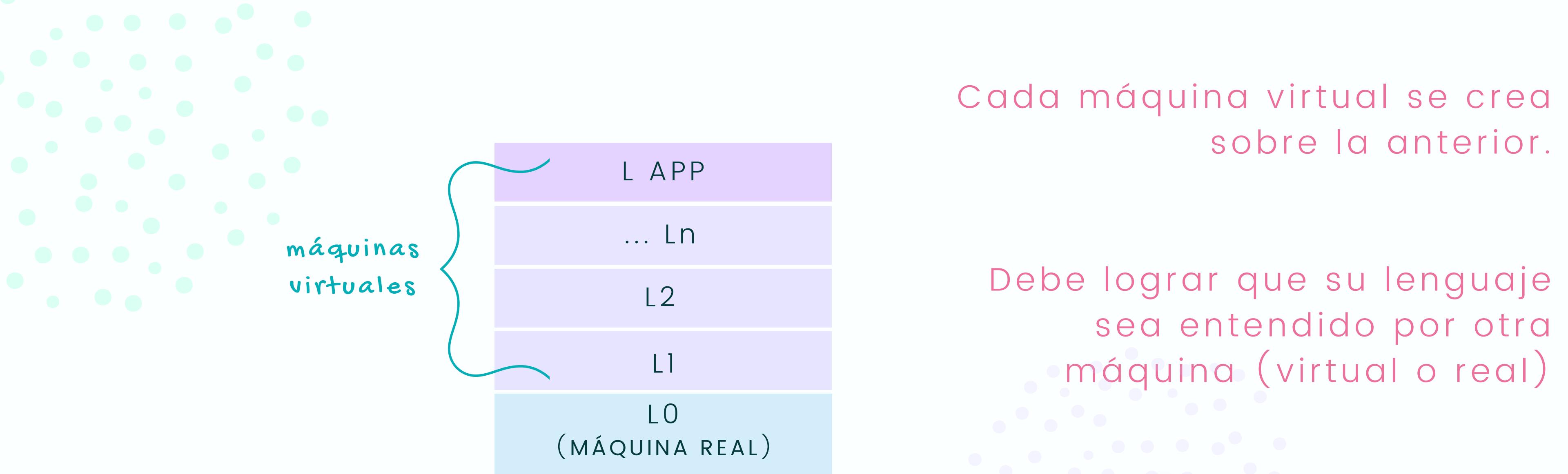
Lenguaje apropiado para escribir un programa.

Sistema de notación que permite describir computos.

En nuestro caso un lenguaje es equivalente a una máquina

PROGRAMA

Texto formado por símbolos. Describe y prescribe el comportamiento de una máquina.



Máquinas virtuales

Cada máquina virtual se crea sobre la anterior.

Debe lograr que su lenguaje sea entendido por otra máquina (virtual o real)



Lenguajes Mapeados

COMPUTADORA
Intérprete implementado en hardware.

COMPILEADOR



Software que traduce un programa del lenguaje de una máquina virtual al lenguaje de otra.

TRADUCCIÓN (Compilación)

Convierte el programa total a otro lenguaje. Se hace una sola vez y es más rápido.

INTERPRETACIÓN

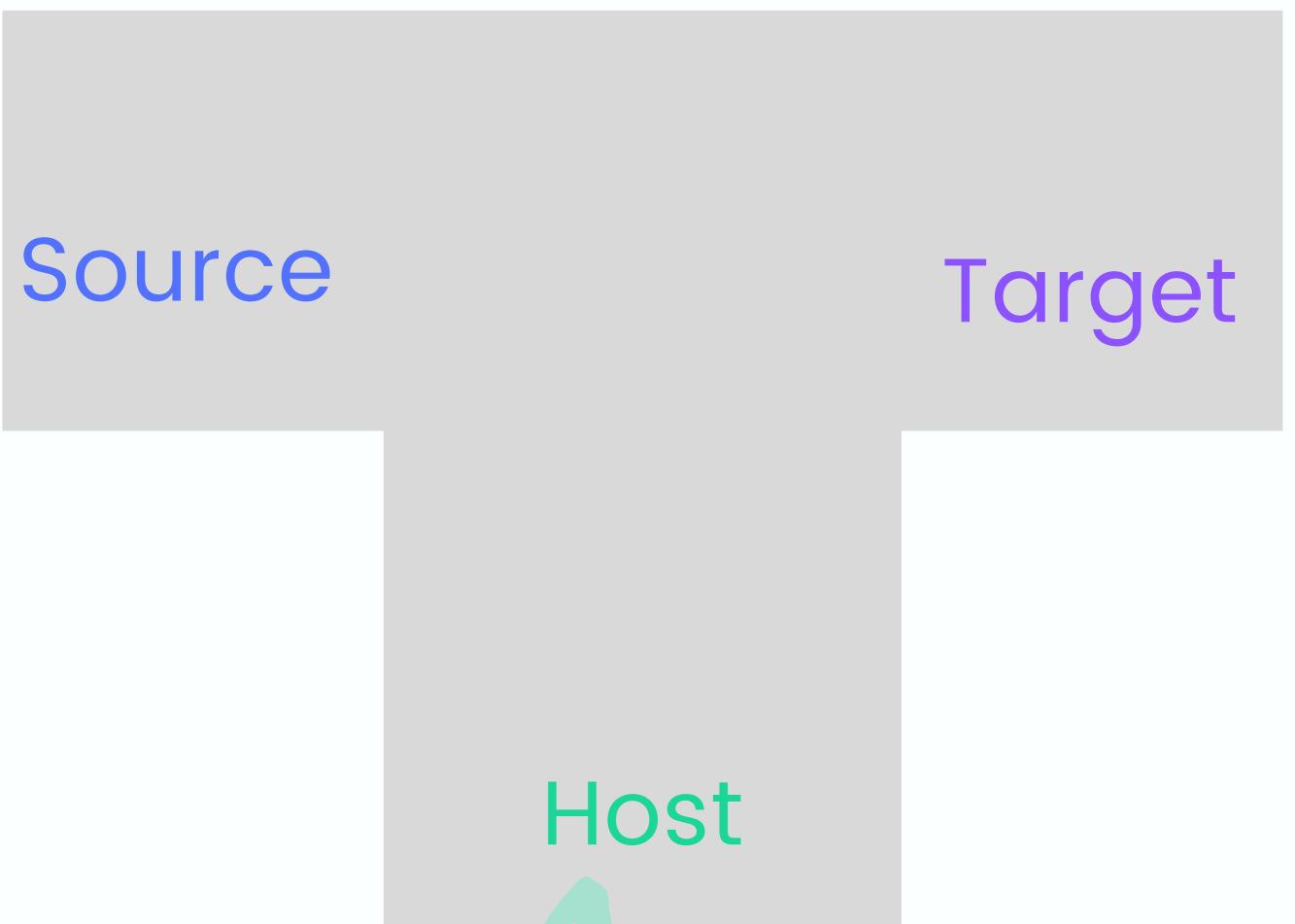
Interpreta instrucción por instrucción. Siempre realiza todo el proceso por lo que es más lento de ejecutar.

INTÉRPRETE

Software que ejecuta instrucciones del lenguaje de una máquina virtual simulándolas en el lenguaje de otra MV.



Lenguaje fuente,
entrada, por traducir.



Host
Lenguaje de
implementación. En lo que
está escrito el compilador

Lenguaje objeto.
Salida



compiladores

C

x86

x86

GCC

Compilador estándar GNU, Linux.

Creado por Richard Stallman:
programador de USA
(SOFTWARE LIBRE)



FORTRAN

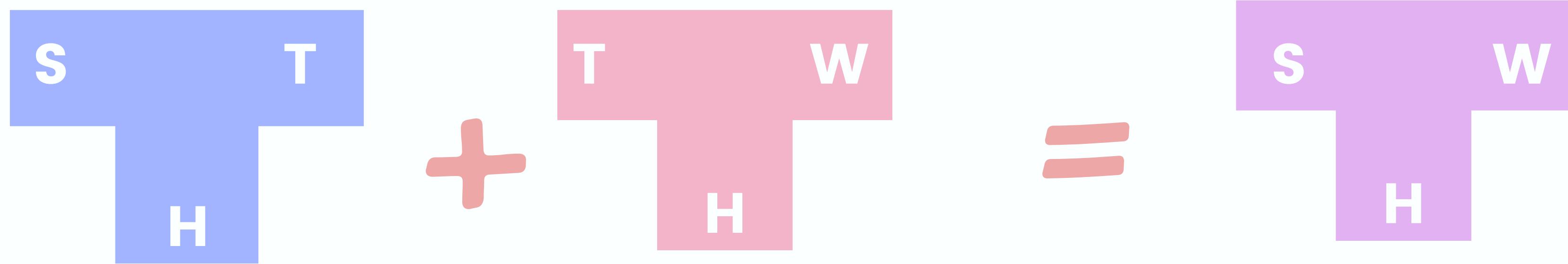
C

x86

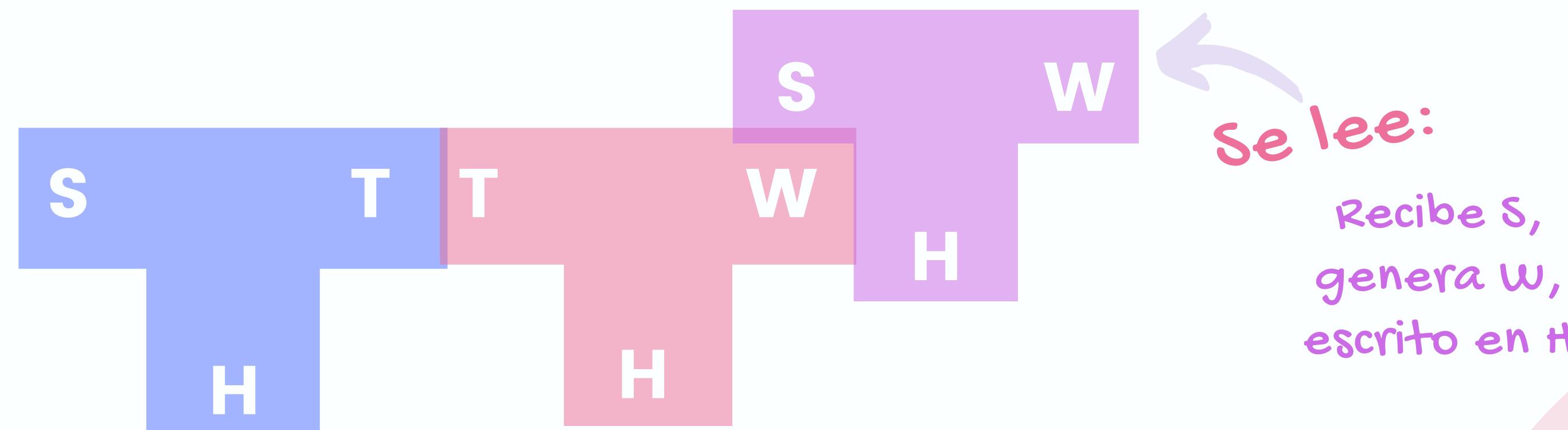
F2C

Convierte Fortran a C porque es más
conocido.

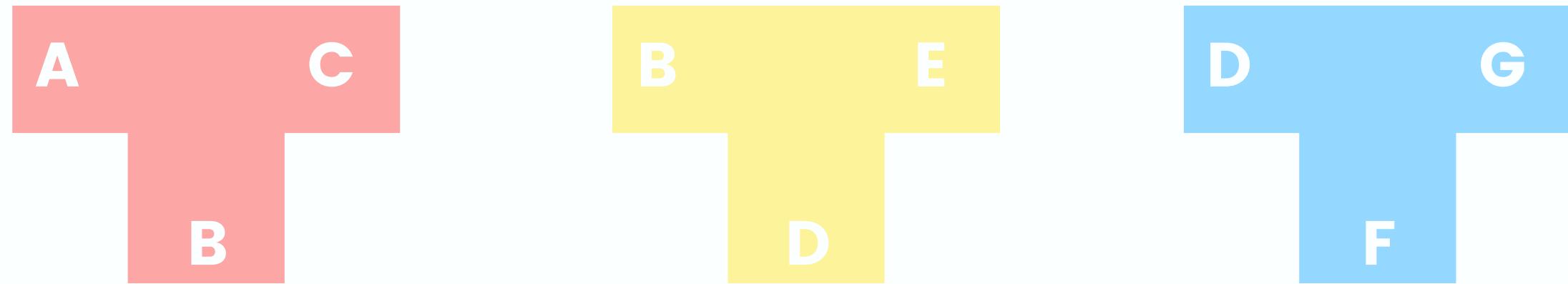
Operaciones con diagramas



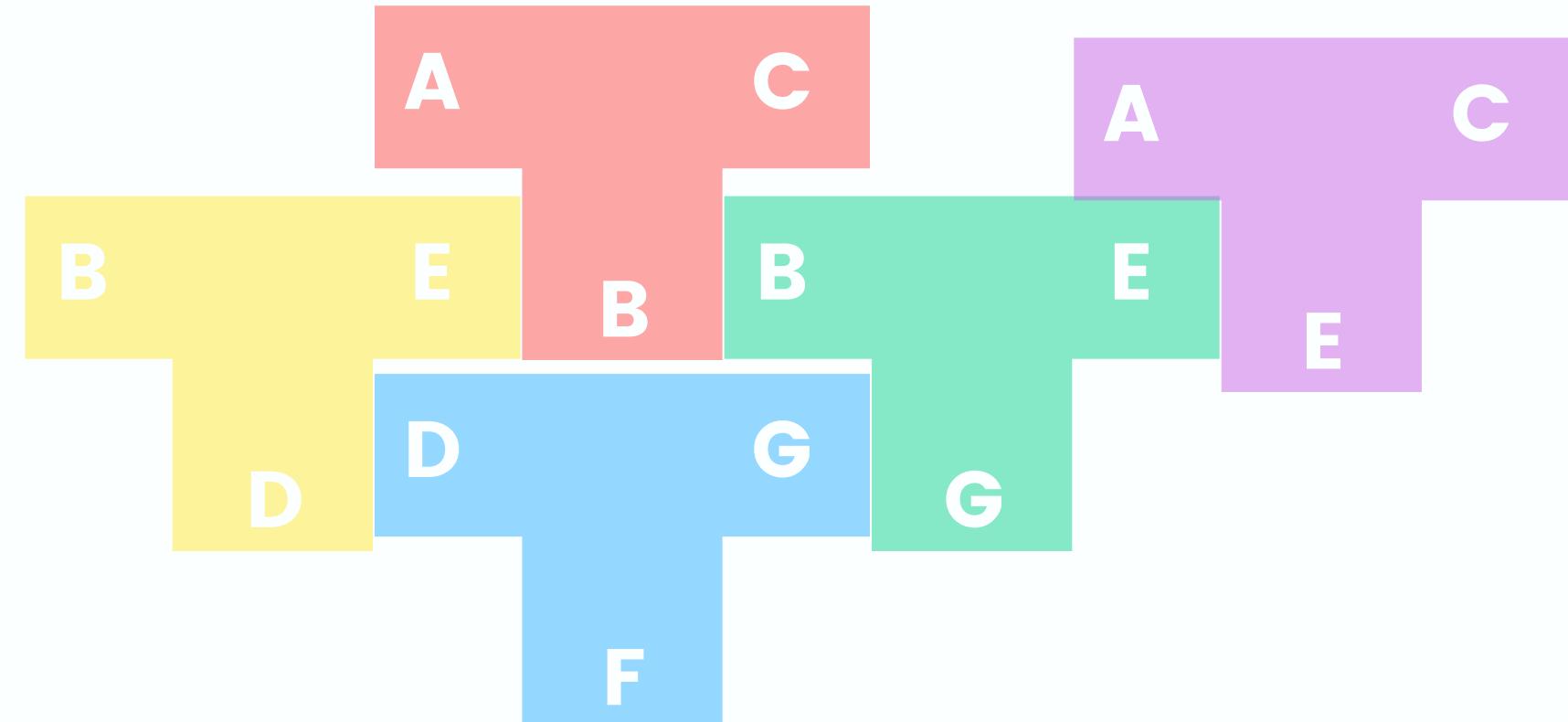
notación:



EJEMPLO



RESUELTO:



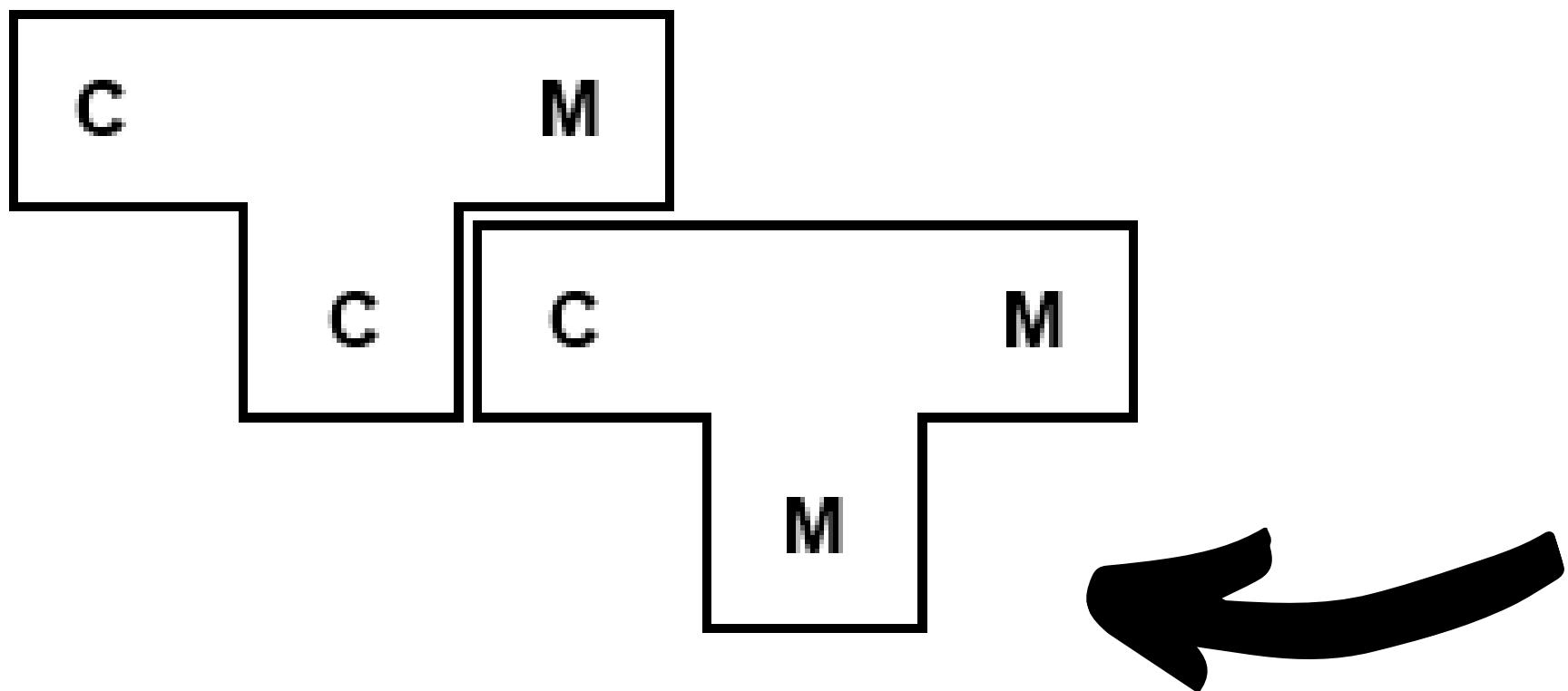


i excited!

2020

APUNTES DEL 9/16

COMPILADORES
E INTERPRETES



APUNTADORA: NAKISHA DIXON

TEMAS VISTOS

EN LA
AGENDA DE
HOY

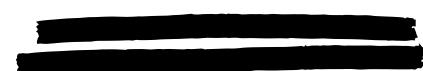


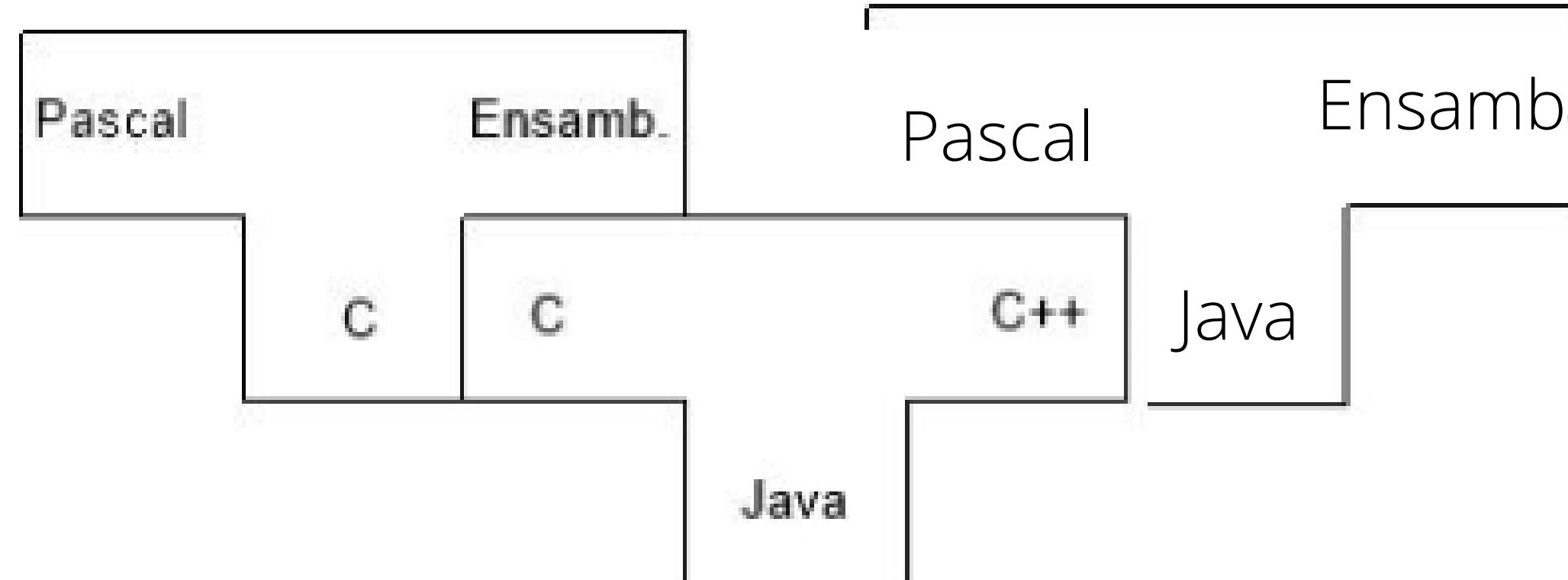
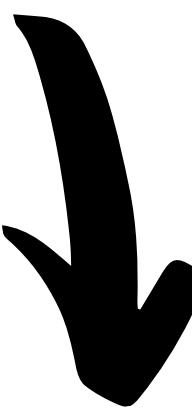
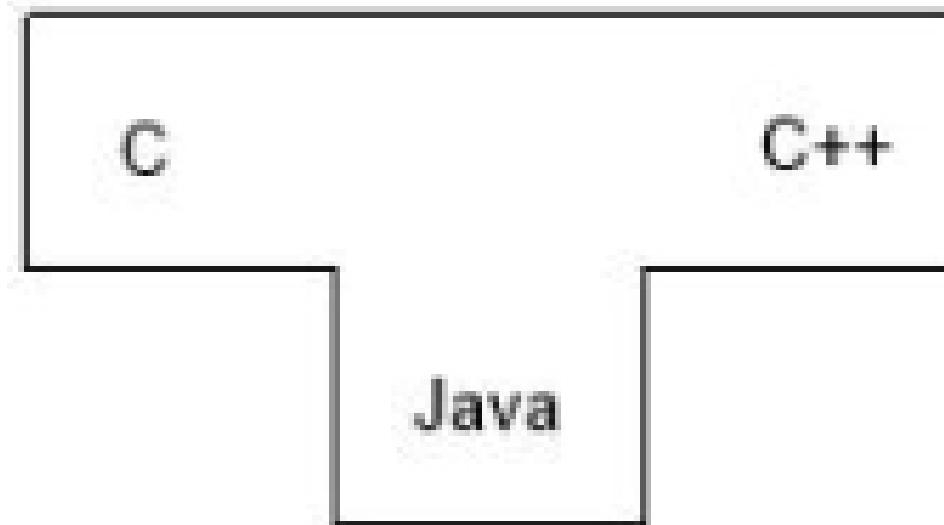
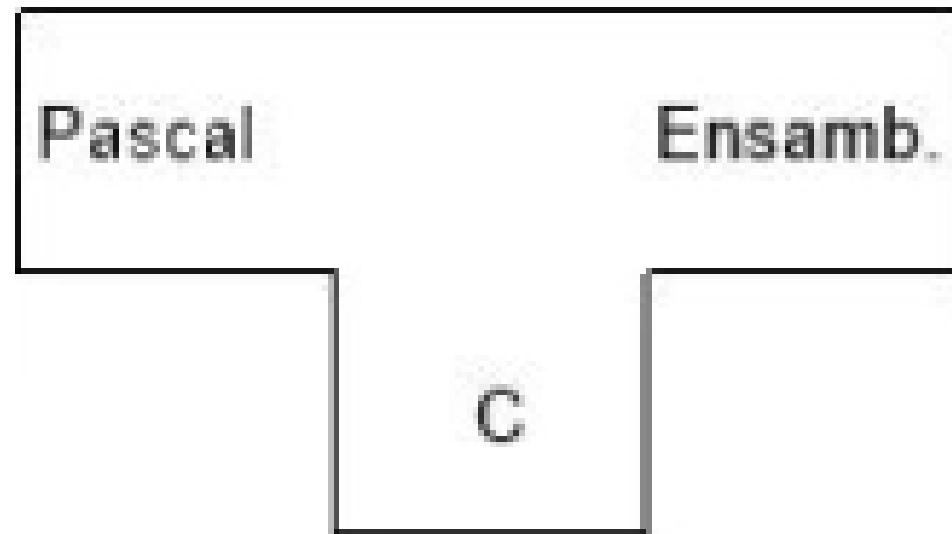
- 1 OPERACIONES CON DIAGRAMAS T
- 2 ARQUITECTURA ARM
- 3 CROSS COMPILER
- 4 BOOTSTRAPPING DE PASCAL
- 5 LENGUAJE OBJETO
- 6 LENGUAJE IMPLEMENTACION

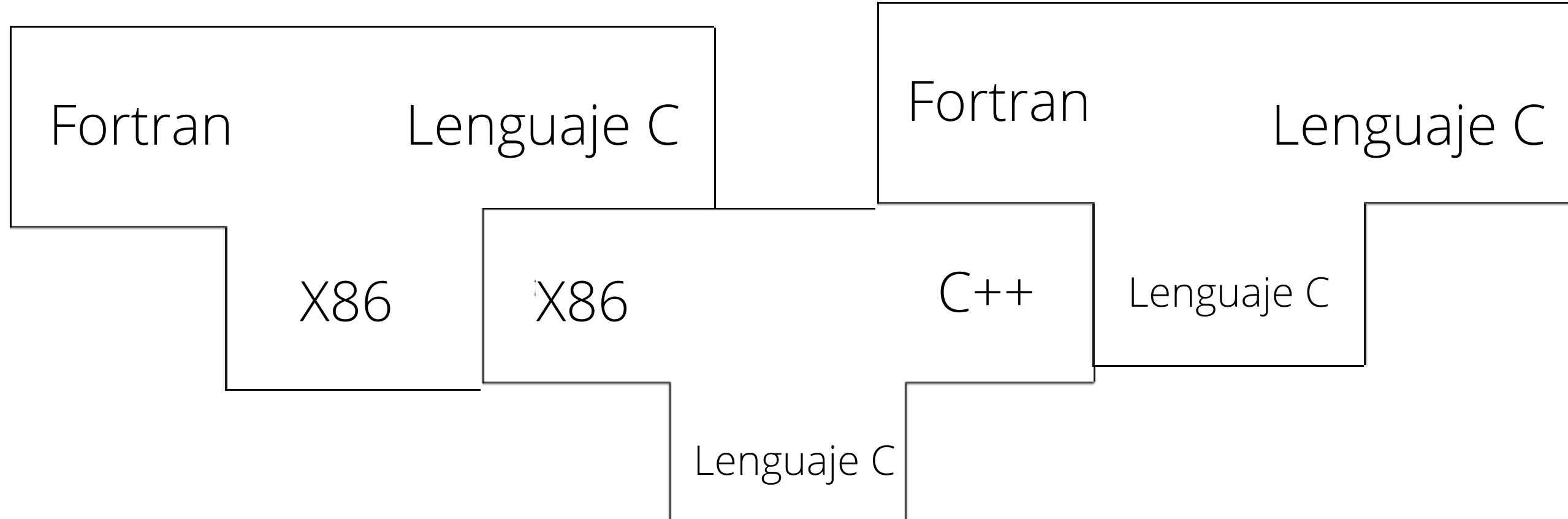
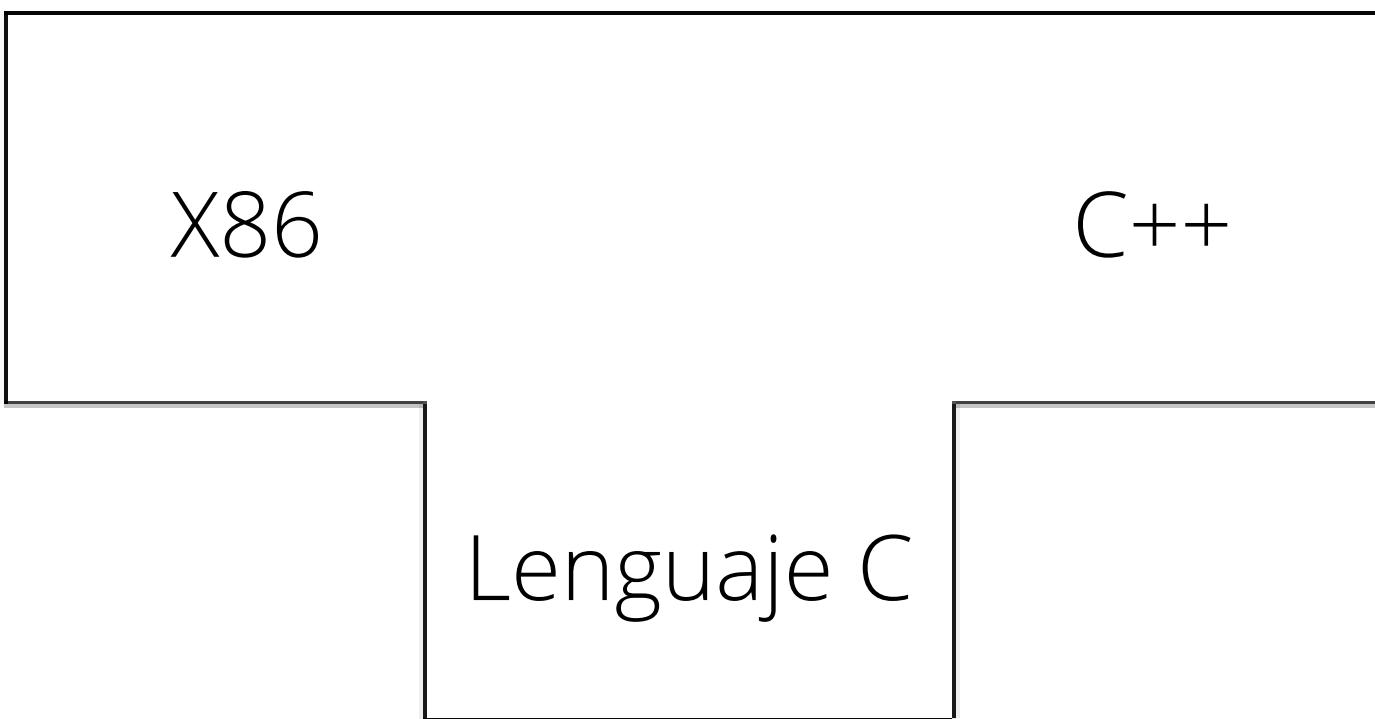
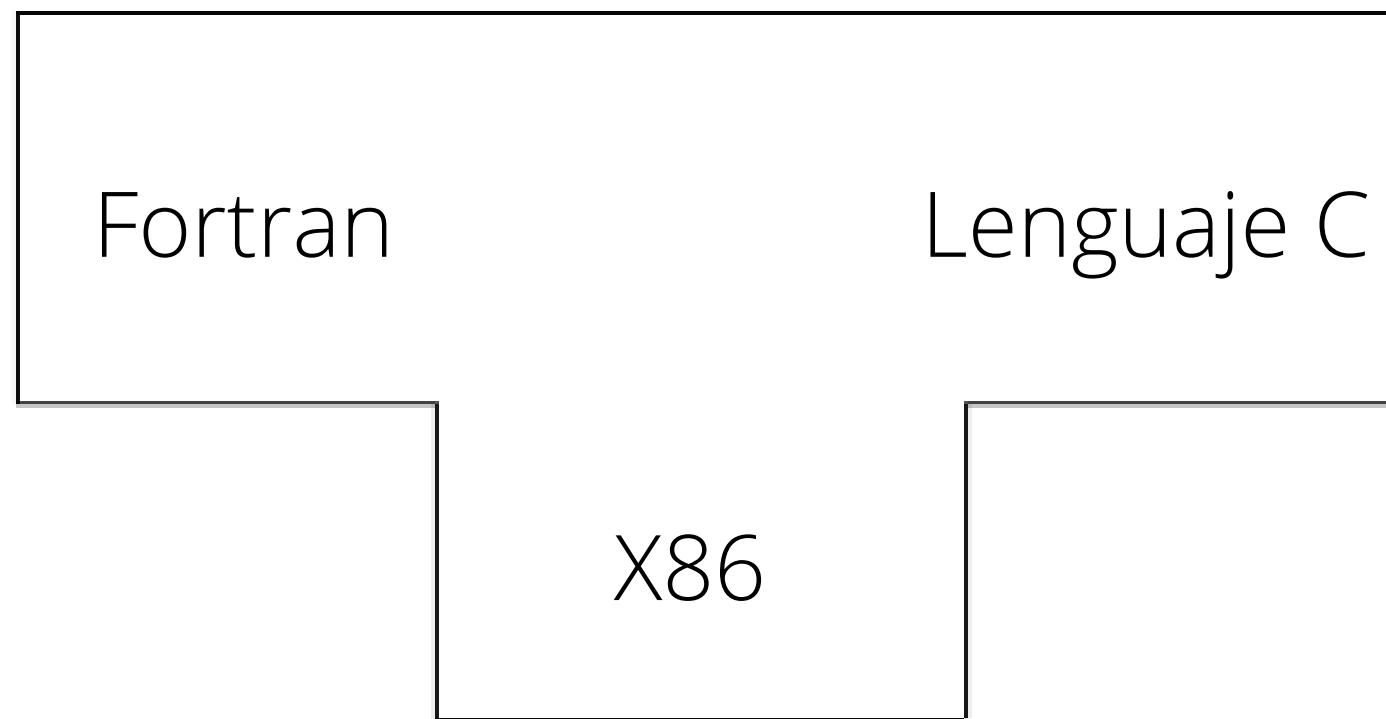
¡NO HUBO QUIZ!

OPERACIONES CON DIAGRAMAS 1

Asociados al compilador hay tres lenguajes. El lenguaje Fuente (Source -S-), el lenguaje Objeto (Target -T-) y el lenguaje implementacion (Host -H-)

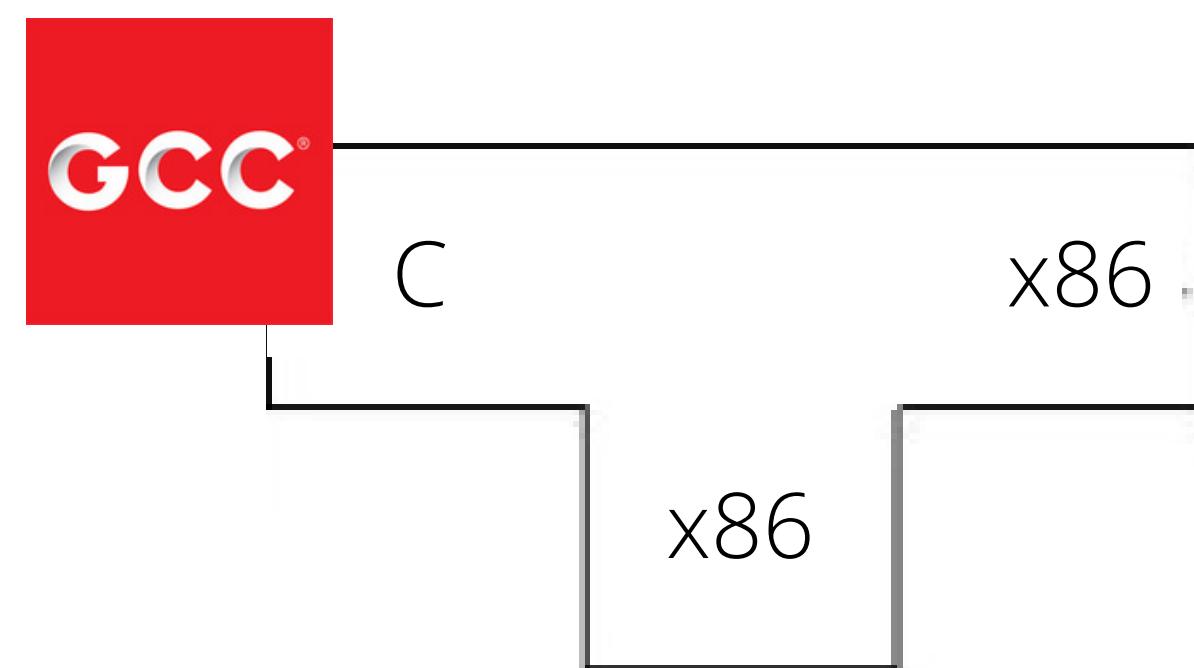




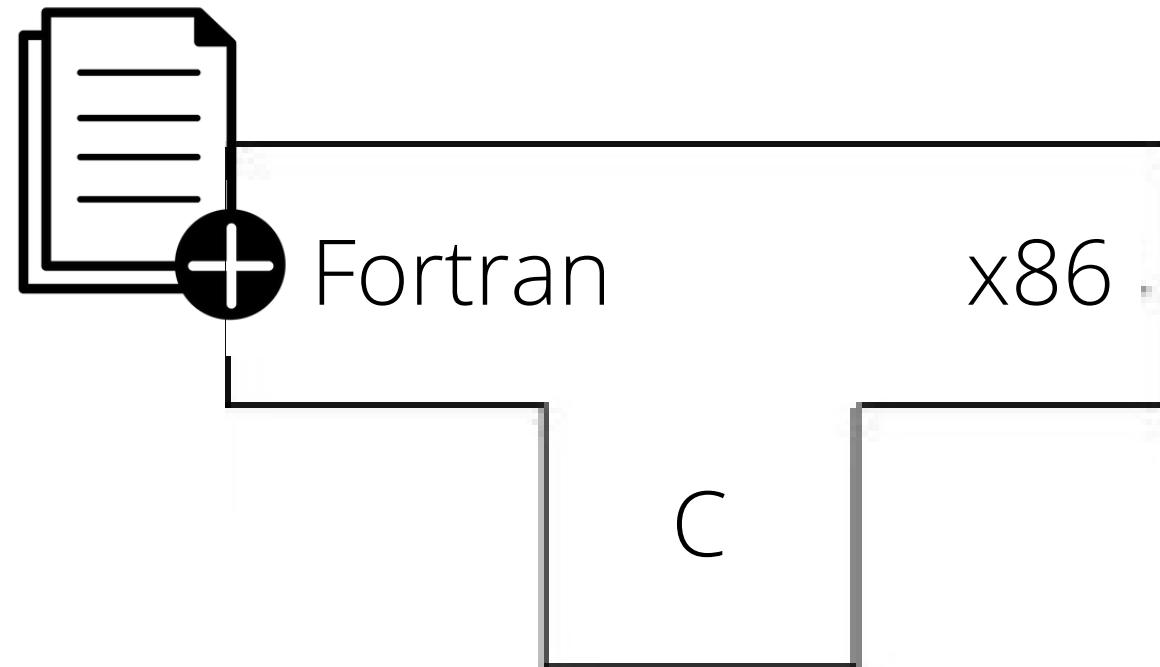


LENGUAJE OBJETO.

El Lenguaje Objeto (Target) puede ser diferente a la máquina en la que corre el compilador (Host)



PROYECTO 1.



- * Recibe el código **MICRO** •
•
•
- * Genera **x86** Corre directamente sobre el hardware
•
•
•
- * Escrita en lenguaje **C** •
•
•
- * Compilamos con el **GCC** Diagrama T para el proyecto
•
•
•

ARQUITECTURA ARM →

- * Sistemas Empotrados
- * Muy usados en celulares, laptops y tablets
- * ARM Holdings los diseña pero no los fabrica - 1980
- * Es la arquitectura mas usada del mundo



Hoy en día, cerca del 75% de los procesadores de 32 bits poseen este chip en su núcleo.

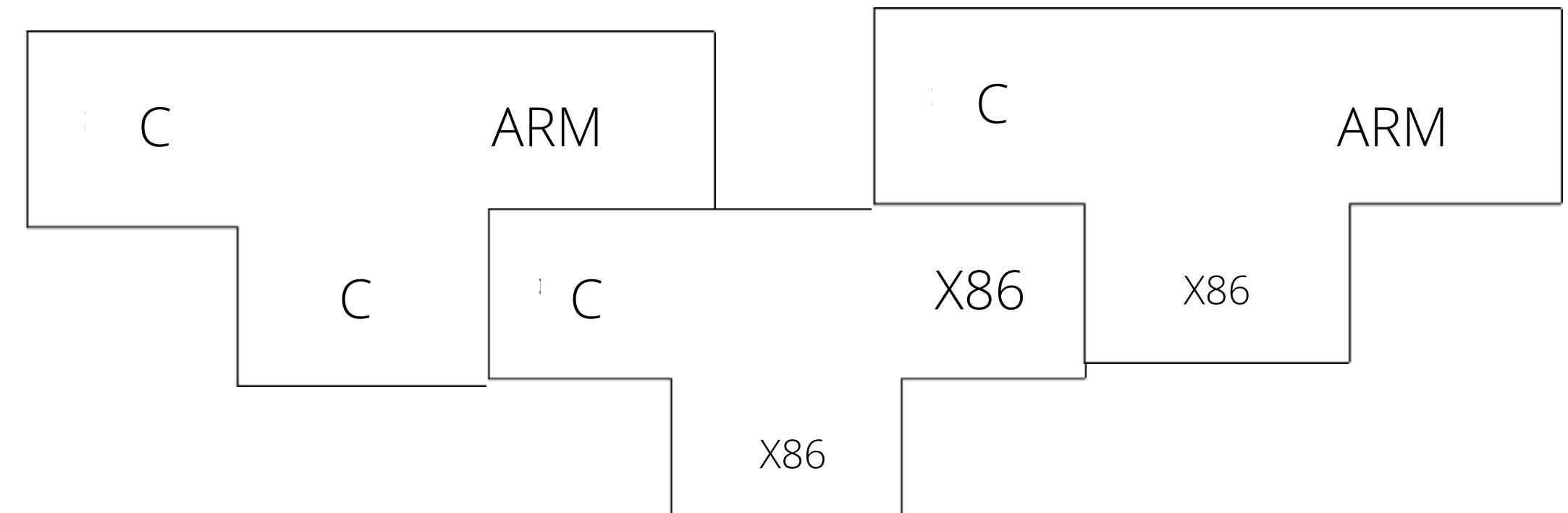
“COMPILADOR ARM PARA X86”

* Tenemos Compilador C que genera ARM
- Escrito en C

* Compilamos con GCC

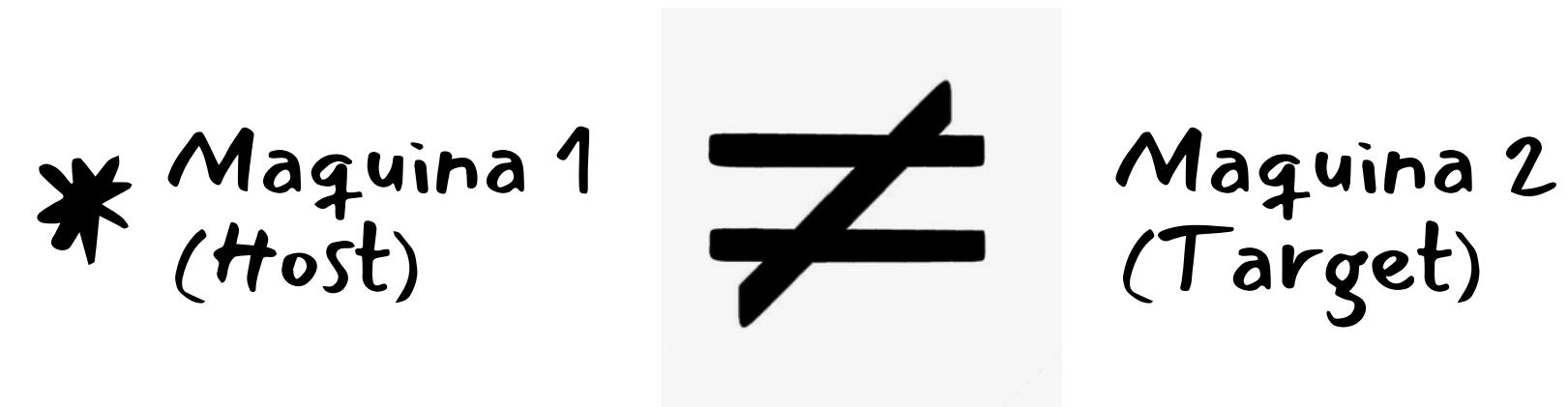
* Un Compilador de C que genera
ARM corriendo sobre x86

* Cross Compiler



CROSS COMPILER

Situacion donde un codigo objeto generado por un compilador no es el mismo que el lenguaje maquina de la maquina donde corre



¿Cuando se necesitan?

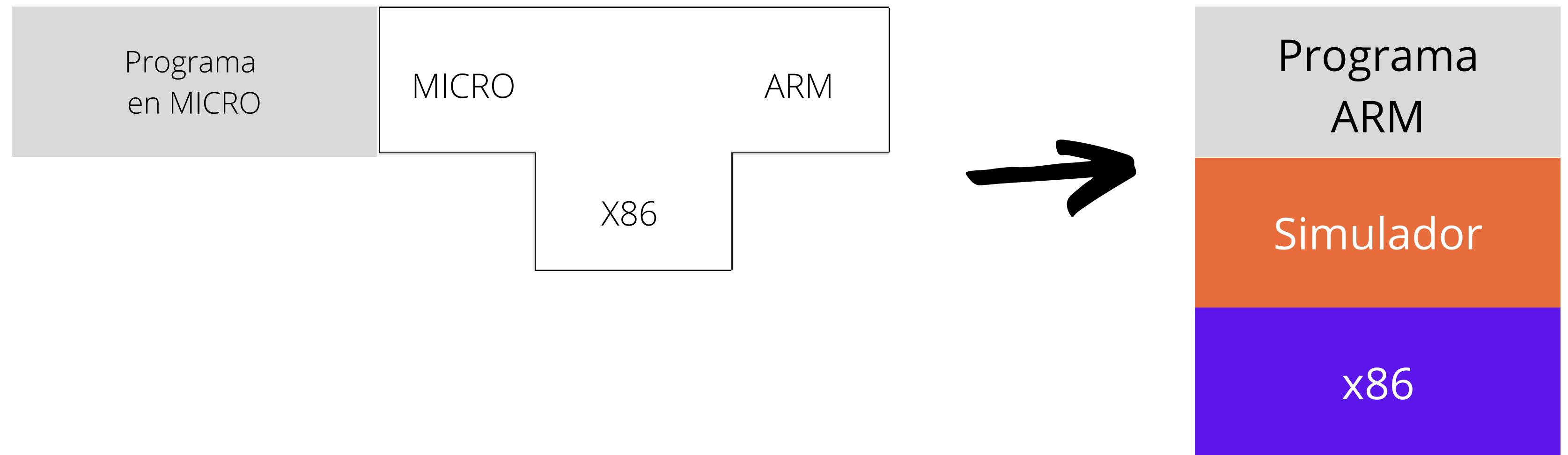
- Maquina 1 aun no existe o no la tenemos
- Maquina 1 tiene muchas limitaciones de recursos como para correr un compilador

Sistemas Empotrados

EJECUTANDO PROGRAMAS MICRO -> VARIANTES

- Tenemos un programa MICRO
- Compilamos con Proyecto 1
- Programa Escrito en MIPS
- ¿Como lo ejecutamos?
- SPIM - Simulador de MIPS que corre sobre x86

EJEMPLO VISUAL



LENGUAJE IMPLEMENTACION.

- Es muy corriente escribir un compilador en el mismo lenguaje implementacion
- Casi el mismo dilema del huevo o la gallina.
Parece ser un circulo vicioso

VERSIONES DE GCC

GCC Timeline

The table is sorted by date. Please refer to our [development plan](#) for future releases.

Release	Release date
GCC 10.2	July 23, 2020
GCC 10.1	May 7, 2020
GCC 9.3	March 12, 2020
GCC 8.4	March 4, 2020
GCC 7.5	November 14, 2019
GCC 9.2	August 12, 2019
GCC 9.1	May 3, 2019
GCC 8.3	February 22, 2019
GCC 7.4	December 6, 2018
GCC 6.5	October 26, 2018
GCC 8.2	July 26, 2018
GCC 8.1	May 2, 2018
GCC 7.3	January 25, 2018
GCC 5.5	October 10, 2017
.....



GCC 5.2 -> Julio, 2015
GCC 4.5.3 -> Abril, 2011

Fuente GCC -> Richard Stallman

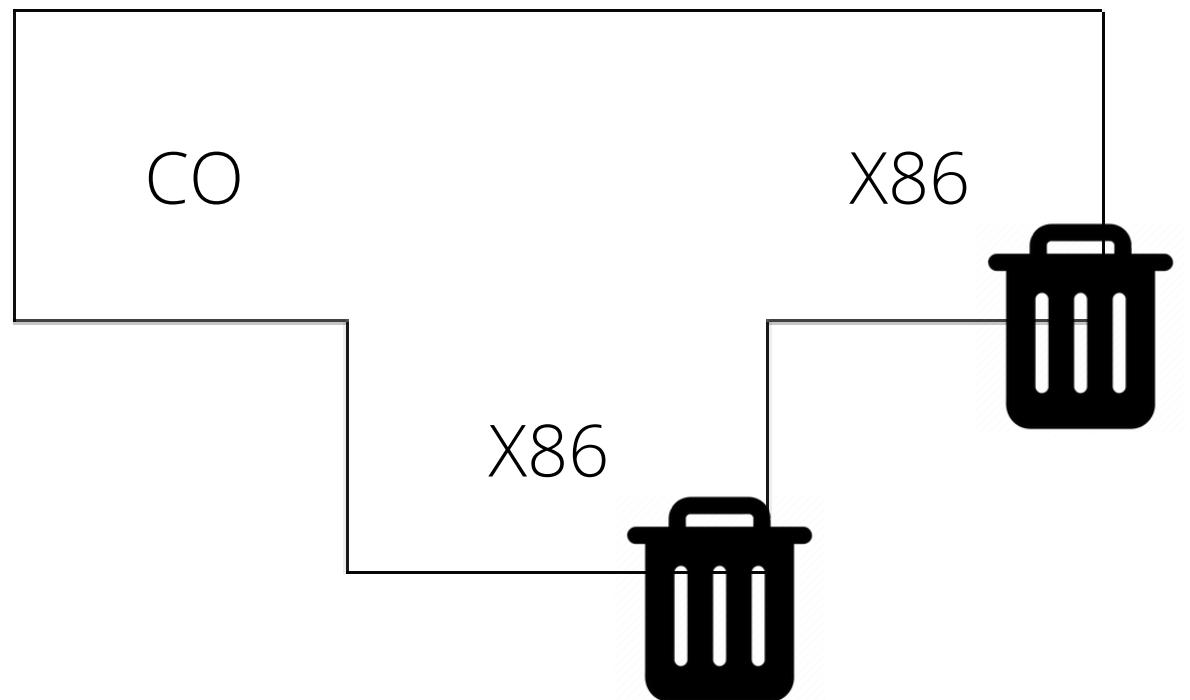
Obtenida de:

<https://gcc.gnu.org/releases.html>

GCC 4.5.3

Abril, 2011

Genera lenguaje maquina de
mala calidad

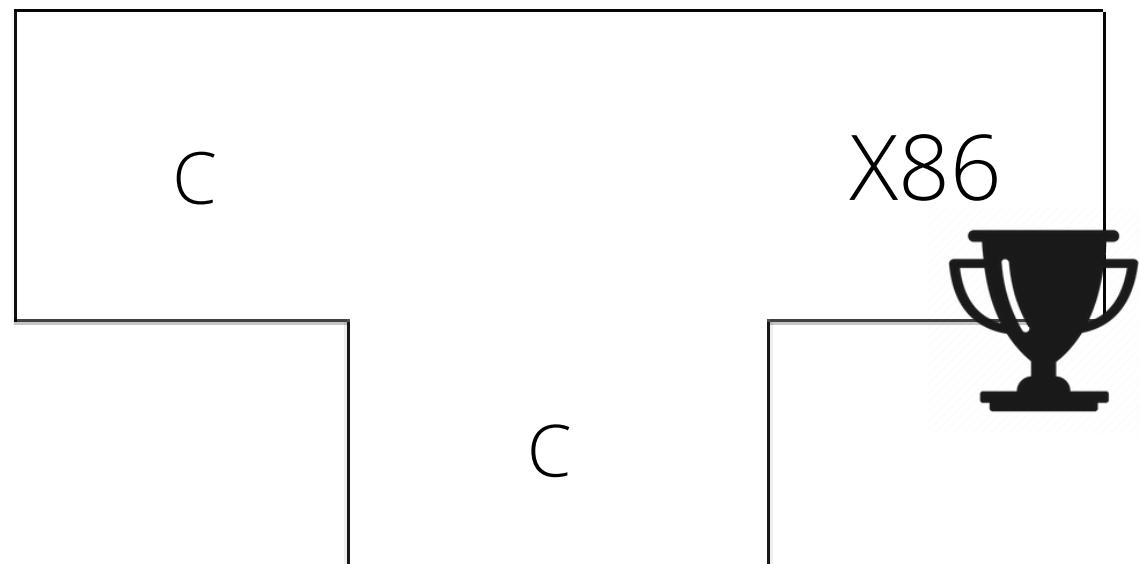


GCC 5.2

Julio, 2015

Genera lenguaje maquina de muy
buena calidad

Solo tenemos el fuente



BOOTSTRAPPING

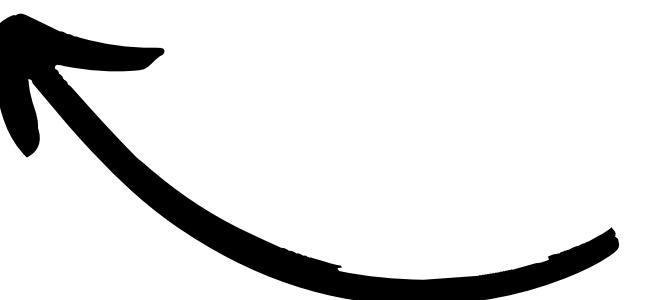
- Definicion: Usar una misma herramienta para crease a si misma (o levantarse)
- Muy frecuente en ciencias de la computacion



NIKLAUS EMIL WIRTH.



- Científico de la Computación. Suiza (1934 -)
- Estudio en el ETH, Canadá, Berkley
- Establece muchos conceptos fundamentales en lenguajes de programación
- Muy influyente en las maneras de enseñar programación y compiladores
- Diseñador de múltiples lenguajes: Algol, Pascal, Modula2, Modula3, Oberon, Oberon2





PASCAL

- Diseñado por Wirth a inicios de los 70's
- Nombre en homenaje de Blaise Pascal
- Lenguaje simple y elegante
- Programación estructurada
- Se usa alrededor del mundo en cursos de introducción a la programación

The screenshot shows the Borland Pascal 7.0 IDE interface. The title bar reads "Borland Pascal 7.0". The menu bar includes File, Edit, Search, Run, Compile, Debug, Tools, Options, Window, Help, and a file tab labeled "SOL4.PAS". The code editor displays the following Pascal program:

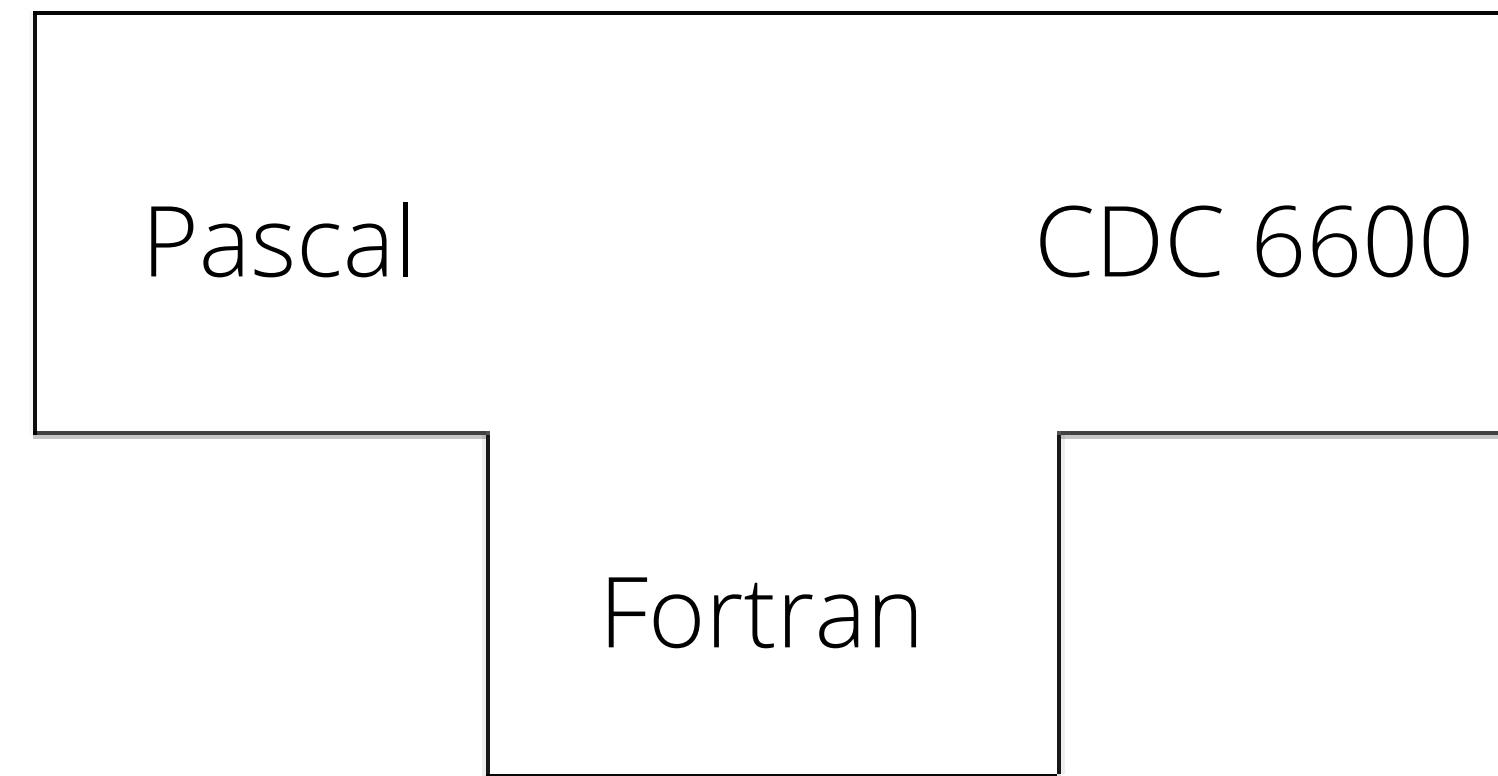
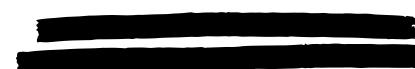
```
var F,L:real;
i,j,n:integer;
x:array[1..10] of real;
y:array[1..10] of real;

begin
  write('n=');readln(n);
  FOR i:=1 TO n DO
    begin
      write('x['+',i,'']=');readln(x[i]);
      write('y['+',i,'']=');readln(y[i]);
    end;
  begin
    write('x['+',n+1,'']=');readln(x[n+1]);
  end;
  y[n+1]:=0;
  F:=0;
  FOR j:=1 TO n DO begin
    L:=1;
    FOR i:=1 TO n DO
      begin
        IF i<>j THEN
          begin
            L:=L*(x[n+1]-x[i])/(x[j]-x[i]);
          end;
        y[n+1]:=y[n+1]+y[j]*L;
      end;
    writeln('y['+',n+1,'']=',y[n+1]:1:0);
    FOR i:=1 TO n DO
      begin
        writeln('x['+',i,'']=',x[i]:10:10,' y['+',i,'']=',y[i]:10:10);
      end;
    begin
      writeln('x['+',n+1,'']=',x[n+1]:10:10,' y['+',n+1,'']=',y[n+1]:10:10);
    end;
    readln;
  end.
```

The status bar at the bottom shows "37:23" and the keyboard shortcut "F1 Help F2 Save F3 Open Alt+F9 Compile F9 Make Alt+F10 Local menu".

PRIMER INTENTO

*Wirth se siente frustrado por Fortran.
Abandona el intento y decide escribir
completamente el compilador de Pascal
*Pequeño problema tecnico



• • •



CDC 6600

EEI CDC 6600 fue la primera supercomputadora de la historia. Diseñada en 1965 por Seymour Cray y fabricada por Control Data Corporation

PRIMER COMPILADOR DE PASCAL



- Wirth diseña completamente el lenguaje PASCAL (1969)
- Empieza a escribir el compilador de Pascal en Fortran
- Genera código para CDC 6600)

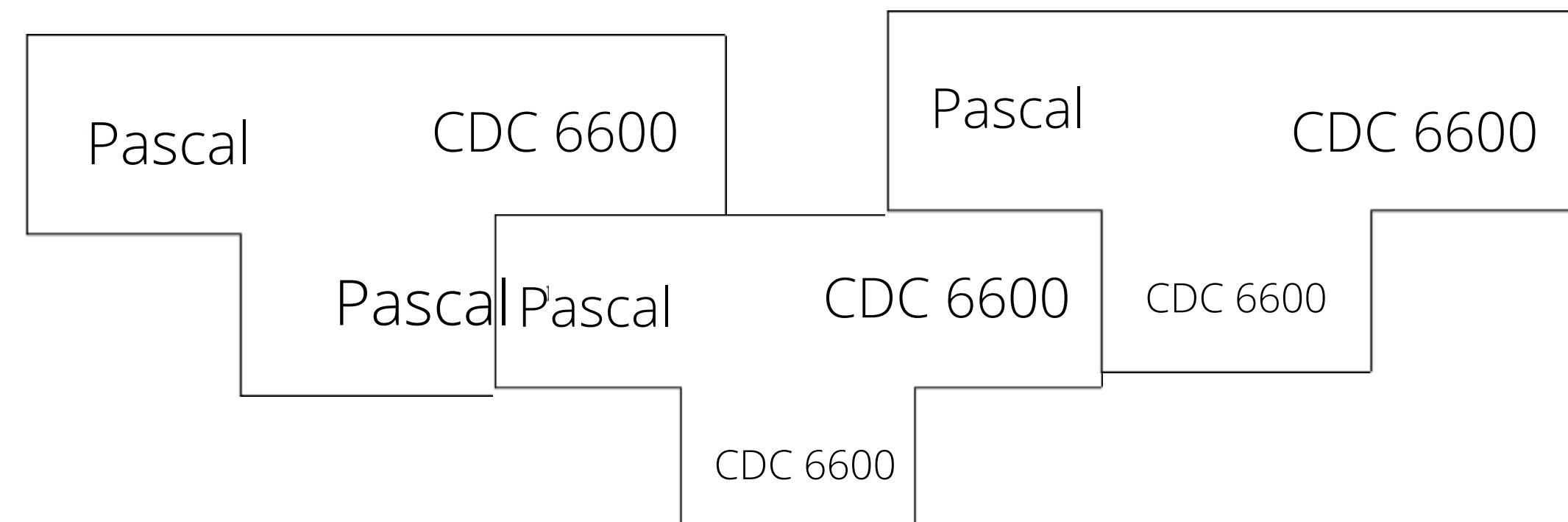
¿COMO COMPILAR EL COMPILADOR DE PASCAL?

- Wirth es experto en compilacion
- Compila "a mano" y genera codigo CDC-6600

No es muy optimizado

¿COMO MEJORAR EL COMPILADOR DE PASCAL?

- Compilador actual genera buen código, pero es lento (código no optimizado)
- Compilemos el fuente del compilador con el mismo
- Compilador eficiente que genera buen código



**Quiz del Miércoles incluye la
materia de las dos ultimas
semanas vistas.**



Apuntes

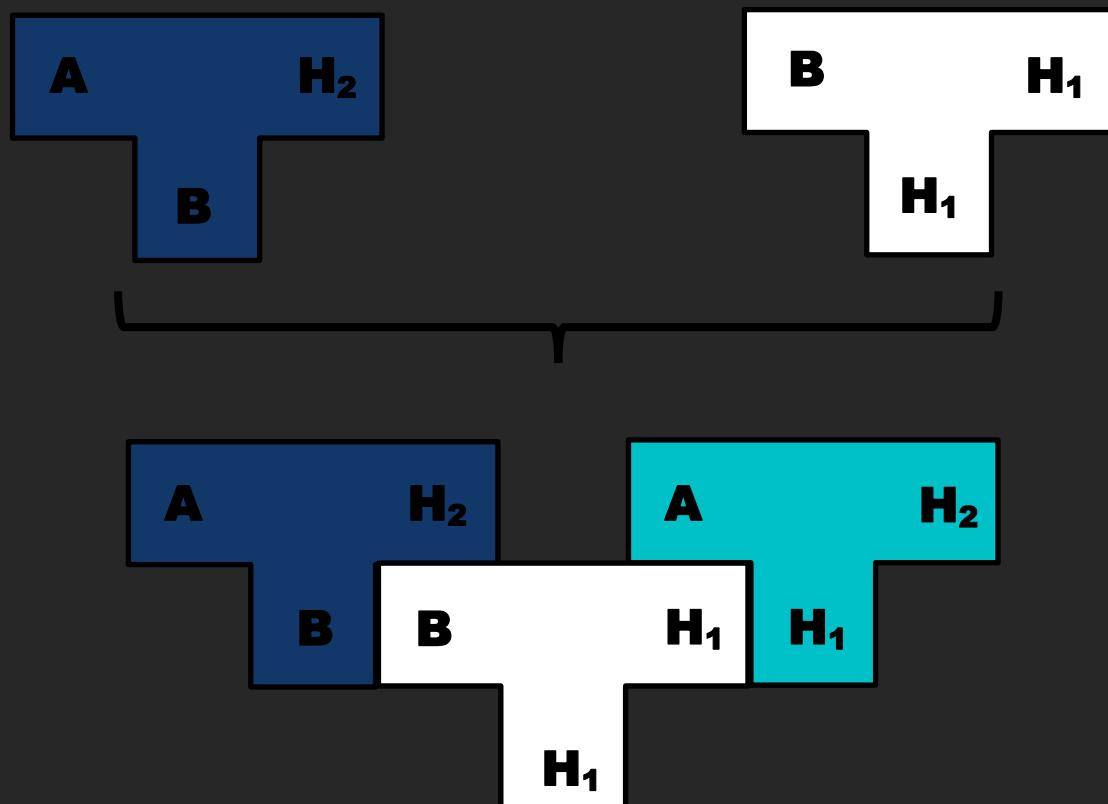
16/09/2020

Compiladores e Intérpretes

*Andrés Darío Gutiérrez
Castro
2018244874*

Lenguaje Objeto

El lenguaje objeto (target) puede ser diferente a la máquina en la que corre el compilador (host).

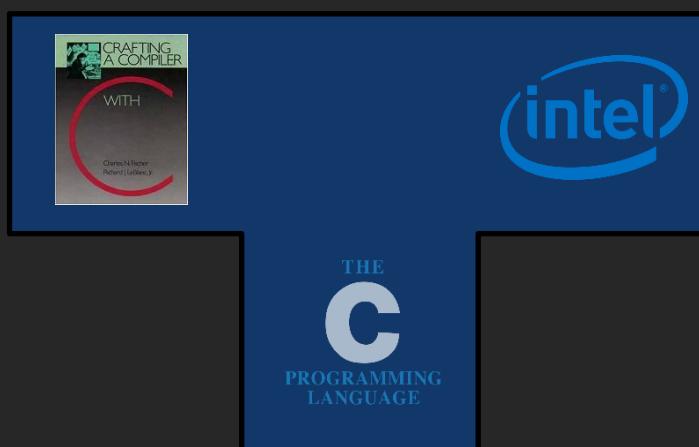


Proyecto 1

Recibe código
MICRO

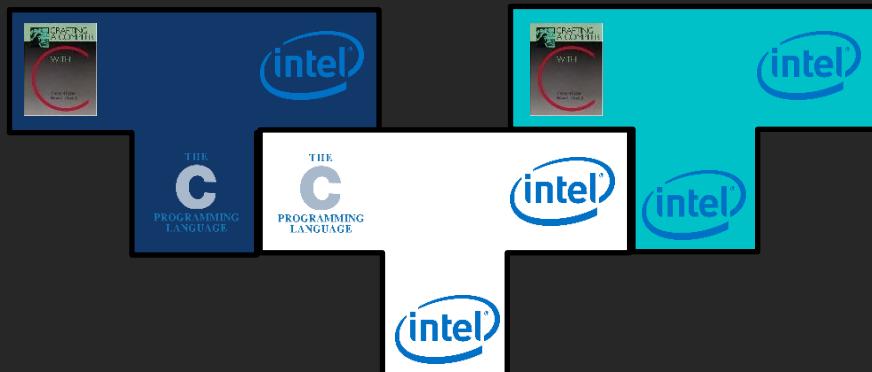
Genera x86 (corre
directamente sobre
hardware)

Escrito en
lenguaje C



Compilando con GCC.

*El proyecto debe compilar con gcc, un compilador de micro que corre x86 y genere x86.



Arquitectura ARM

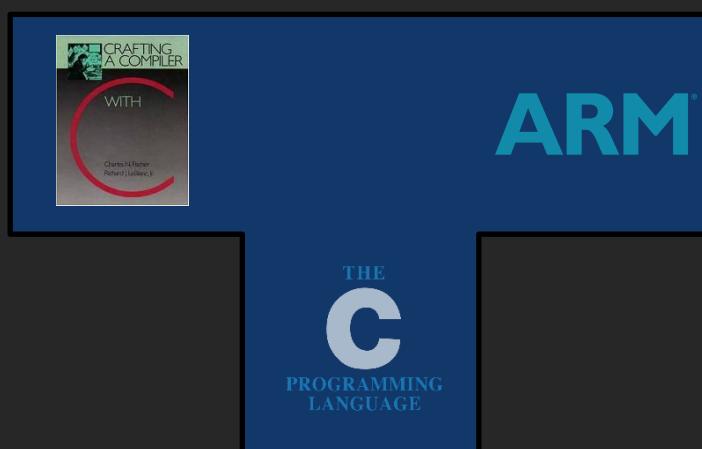
- Familia de conjuntos de arquitectura de lenguajes.
- Diseño original de los 19809 (ARM holdings - británica).
- Arquitectura RISC.
- Muy eficiente en uso de energía y espacio.
- Muy usados en celulares, laptops, tablets.
- Sistemas Empotrados (Software para sistemas esclusivos)
- ARM Holdings los diseña pero no los fabrica.
- Modelos con 64/32 bits
- Es la arquitectura más usada del mundo
 - 50 mil millones al 2014, 60% de todos los aparatos.
 - 160 mil millones al 2020

Variante de Proyecto 1 / ARM

Recibe código
MICRO

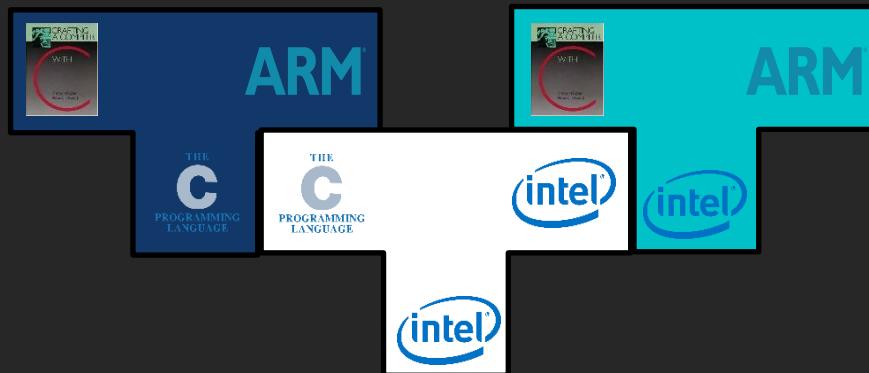
Genera ARM (corre
sobre Simulador en
Linux)

Escrito en
lenguaje C



Compilando variante de proyecto 1

- Compilar con GCC
- Un compilador de MICRO, que genera ARM, corriendo sobre x86
- Cross Compiler

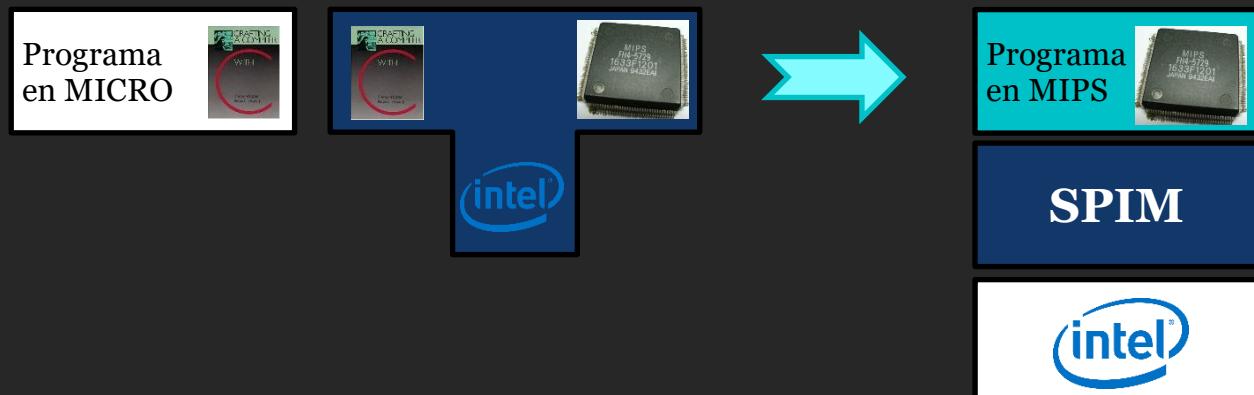


Compilador Cruzado

- Situación donde el código objeto generado por un compilador no es el mismo que en el lenguaje máquina de la máquina donde corre.
- Máquina 1 (host) ≠ Máquina 2 (target).
- **Cross compiler**
- ¿Cuándo se necesita?
 - Máquina 1 aún no existe o no la tenemos.
 - Máquina 1 tiene muchas limitaciones de recursos como para correr un compilador.
- Sistemas empotrados (embedded systems).

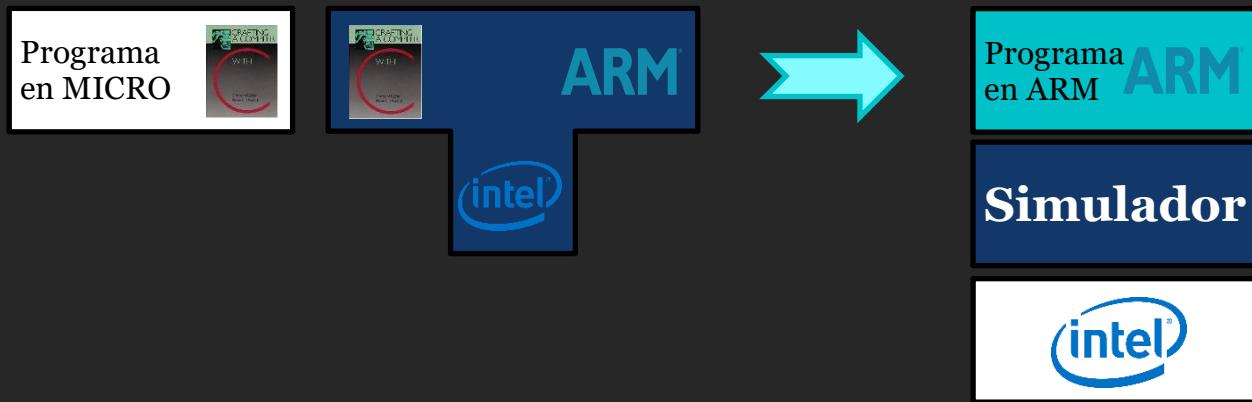
Ejecutando programas MICRO – Variante MIPS

- Tenemos un programa en MICRO.
- Compilamos con Proyecto 1.
- Programa escrito en MIPS.
- ¿Cómo lo ejecutamos?
- SPIM – simulador de MIPS que corre sobre x86



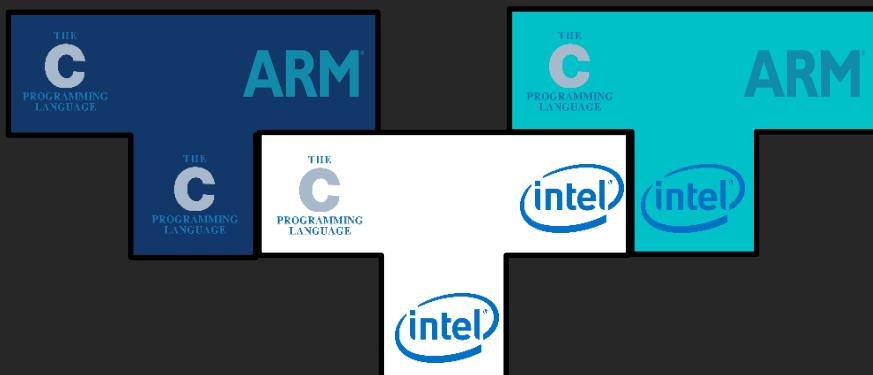
Ejecutando programas MICRO – Variante ARM

- Tenemos un programa en MICRO.
- Compilamos con Proyecto 1.
- Programa escrito en ARM.
- ¿Cómo lo ejecutamos?
- Simulador de ARM que corre sobre x86



Compilador para ARM en x86

- Tenemos compilador de C que genera ARM – escrito en C.
- Compilamos con GCC.
- Un compilador de C, que genera ARM, corriendo sobre x86.
- Cross Compiler



Compilador Cruzado – ARM

Traducción para ARM

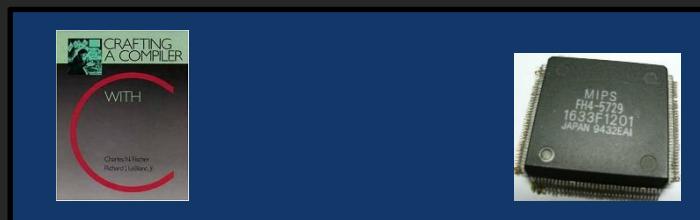
A screenshot of a terminal window titled "manc@manpc: ~" showing assembly code for an ARM processor. The code is generated from a C program that includes printf statements. The assembly includes instructions like push, mov, and printf, along with comments explaining the stack frame and memory layout. The terminal also shows some environment variables and the path to the assembly file.

Variante de Proyecto 1 / MIPS

Recibe código
MICRO

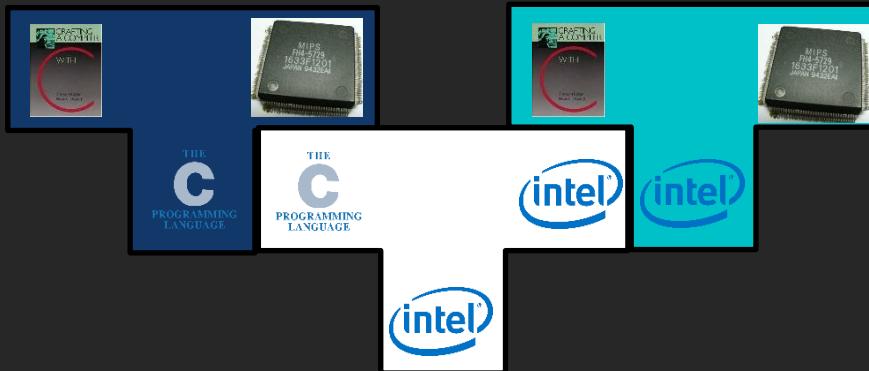
Genera MIPS (corre
sobre simulador
SPIM)

Escrito en
lenguaje C



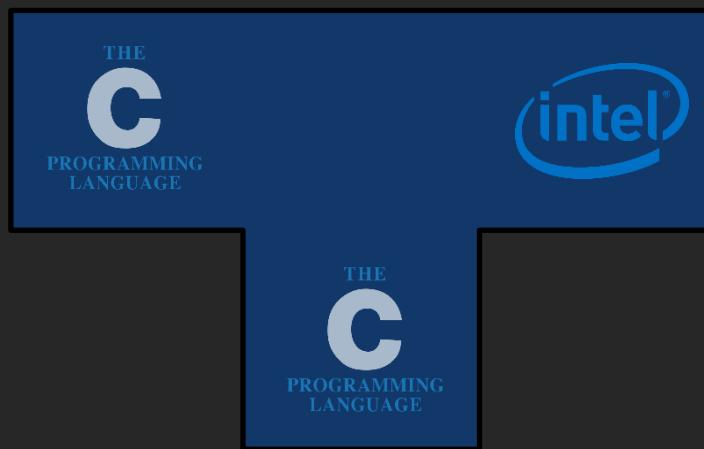
Compilando Variante de Proyecto 1

- Compilamos con GCC.
- Un compilador de MICRO, que genera MIPS, corriendo sobre x86.
- Cross Compiler que genera MIPS.



Lenguaje de Implementación

- Es muy corriente escribir un compilador en el mismo lenguaje de implementación.
- Compilador de C, escrito en C que genere x86, usando un tipo de trampa con gcc:



Versiones de GCC

<https://gcc.gnu.org/releases.html>

GCC Releases

Download

GCC releases may be downloaded from our [mirror sites](#).

Important: these are source releases, so will be of little use if you do not already have a C++ compiler installed. As one option, there are [pre-compiled binaries](#) for various platforms.

You can also retrieve our sources [using Git](#).

GCC Timeline

The table is sorted by date. Please refer to our [development plan](#) for future releases and an alternative view of the release history.

Release	Release date
GCC 10.2	July 23, 2020
GCC 10.1	May 7, 2020
GCC 9.3	March 12, 2020
GCC 8.4	March 4, 2020
GCC 7.5	November 14, 2019
GCC 9.2	August 12, 2019
GCC 9.1	May 3, 2019
GCC 8.3	February 22, 2019
GCC 7.4	December 6, 2018
GCC 6.5	October 26, 2018
GCC 8.2	July 26, 2018
GCC 8.1	May 2, 2018
GCC 7.3	January 25, 2018

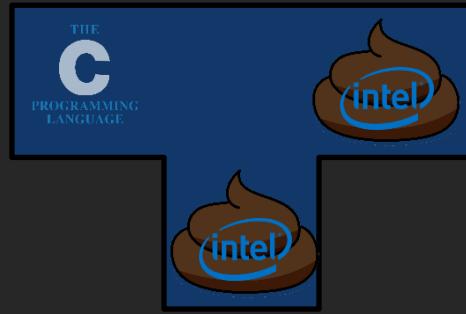
GCC 5.2	July 16, 2015
GCC 4.9.3	June 26, 2015
GCC 4.8.5	June 23, 2015
GCC 5.1	April 22, 2015
GCC 4.8.4	December 19, 2014
GCC 4.9.2	October 30, 2014
GCC 4.9.1	July 16, 2014
GCC 4.7.4	June 12, 2014
GCC 4.8.3	May 22, 2014
GCC 4.9.0	April 22, 2014
GCC 4.8.2	October 16, 2013
GCC 4.8.1	May 31, 2013
GCC 4.6.4	April 12, 2013
GCC 4.7.3	April 11, 2013
GCC 4.8.0	March 22, 2013
GCC 4.7.2	September 20, 2012
GCC 4.5.4	July 2, 2012
GCC 4.7.1	June 14, 2012
GCC 4.7.0	March 22, 2012
GCC 4.4.7	March 13, 2012
GCC 4.6.3	March 1, 2012
GCC 4.6.2	October 26, 2011
GCC 4.6.1	June 27, 2011
GCC 4.3.6	June 27, 2011
GCC 4.5.3	April 28, 2011

GCC 5.2
Julio, 2015

GCC 4.5.3
Abril, 2011

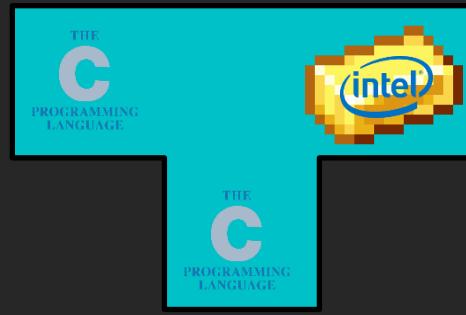
GCC 4.5.3

- Abril, 2011.
- Genera lenguaje de máquina x86 de baja calidad.



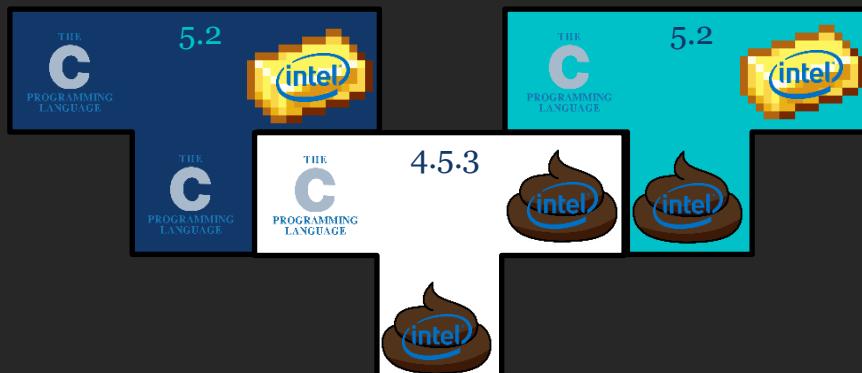
GCC 5.2

- Julio, 2015
- Genera lenguaje de máquina x86 de muy buena calidad
- Solo tenemos el fuente.
- Una versión más actualizada.



Compilando el Compilador

- Compilamos el 5.2 con el 4.5.3 (sólo tenemos eso).
- Un compilador "lento" que genera muy buen código.
- El 5.2 no lo tenemos en la máquina.



- Compilamos el 5.2 con el 5.2 (es lento compilando pero genera buen código).
- Un compilador rápido que genera muy buen código.
- Para actualizar a una versión más moderna del compilador.



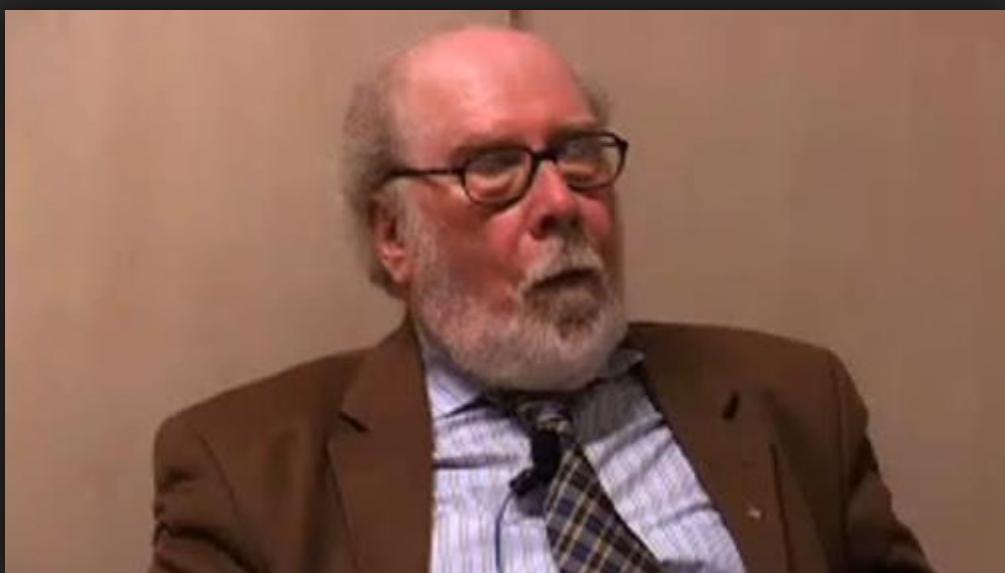
Bootstrapping

- Usar la misma herramienta para crearse o levantarse a sí misma.
- Muy frecuente en Ciencia de la Computación.
- Sistema Operativos, Compiladores, Diseño de Arquitecturas, etc.



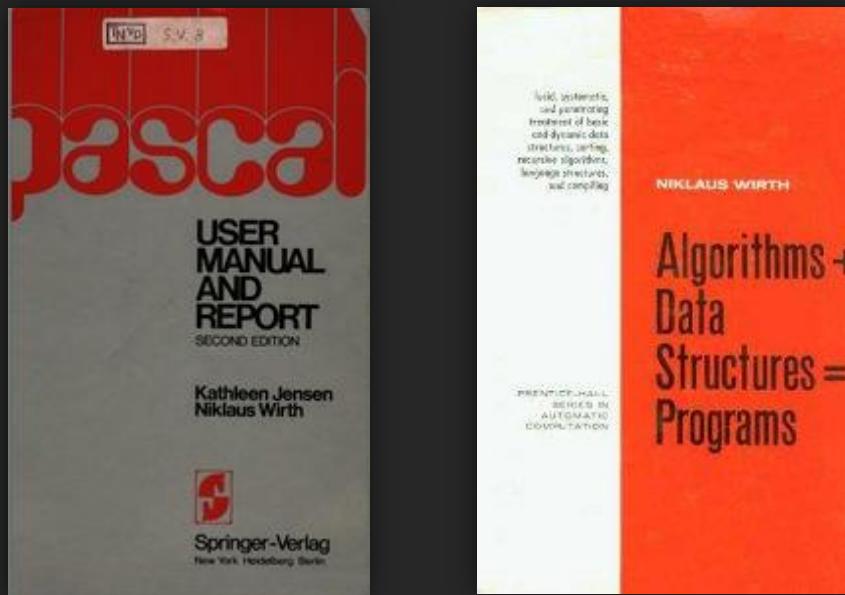
Niklaus Emil Wirth

- Científico de la Computación, Suiza (1934 - ...)
- ETH (B.Sc), Canadá (M.Sc), Berkeley (Ph.D.)
- Establece muchos conceptos fundamentales en lenguajes de programación e ingeniería de Software.
- Muy influyente en las maneras de enseñar programación y compiladores.
- Diseñador de múltiples lenguajes: Algol-W, [Pascal](#), Modula, Modula 2, Modula 3, Oberon, Oberon 2.



Pascal

- Lenguaje de programación diseñado por Wirth a inicios de 1970s
- Diseñado para que la enseñanza a la programación sea sencilla.
- Nombre en homenaje a Blaise Pascal
- Libro de texto “Algorithms + Data Structures = Programs”.
- Objetivos: simplicidad, elegancia, apropiado para enseñar a programar con buenos hábitos.
- Programación estructurada.
- Muy exitoso e influyente.
- Se usó alrededor del mundo en cursos introductorios de computación.
- Origen académico, tuvo derivados comerciales muy exitosos.



Primer compilador de Pascal

- Wirth diseña completamente el lenguaje PASCAL (1969).
- Empieza a escribir el compilador de PASCAL en FORTRAN.
- Genera código para CDC 6600.

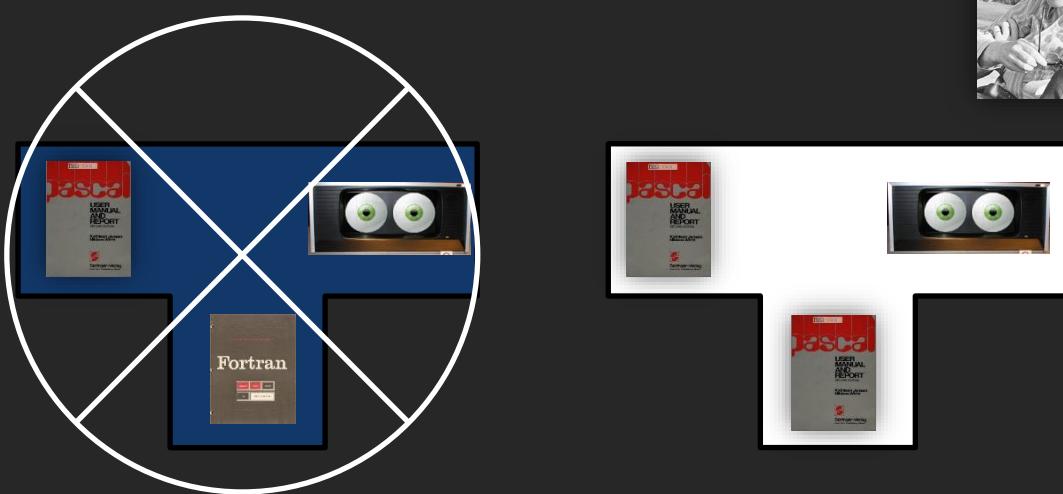


Pascal (Versión 1 de Niklaus/ Primer intento)



Pascal (Segundo intento)

- Wirth se siente muy frustrado por Fortran. Abandona el intento.
- Decide escribir completamente el compilador de PASCAL en PASCAL.



Compilar el compilador de Pascal

- Wirth es experto en compiladores.
- Compila "a mano" y genera código CDC-6600 (no muy optimizado)
- Compilador funcional, lento, pero genera buen código.



Mejorar el compilador de Pascal

- Compilador actual genera buen código, pero es lento (código no optimizado).
- Compilar el fuente del compilador con él mismo.
- Compilador eficiente que genera buen código.



COMPILEDORES E INTÉPRETES



APUNTES 18/09/2020

Fabián Ceciliano Ramírez

Retomando...

1

Idea de Wirth de escribir Pascal con Pascal.

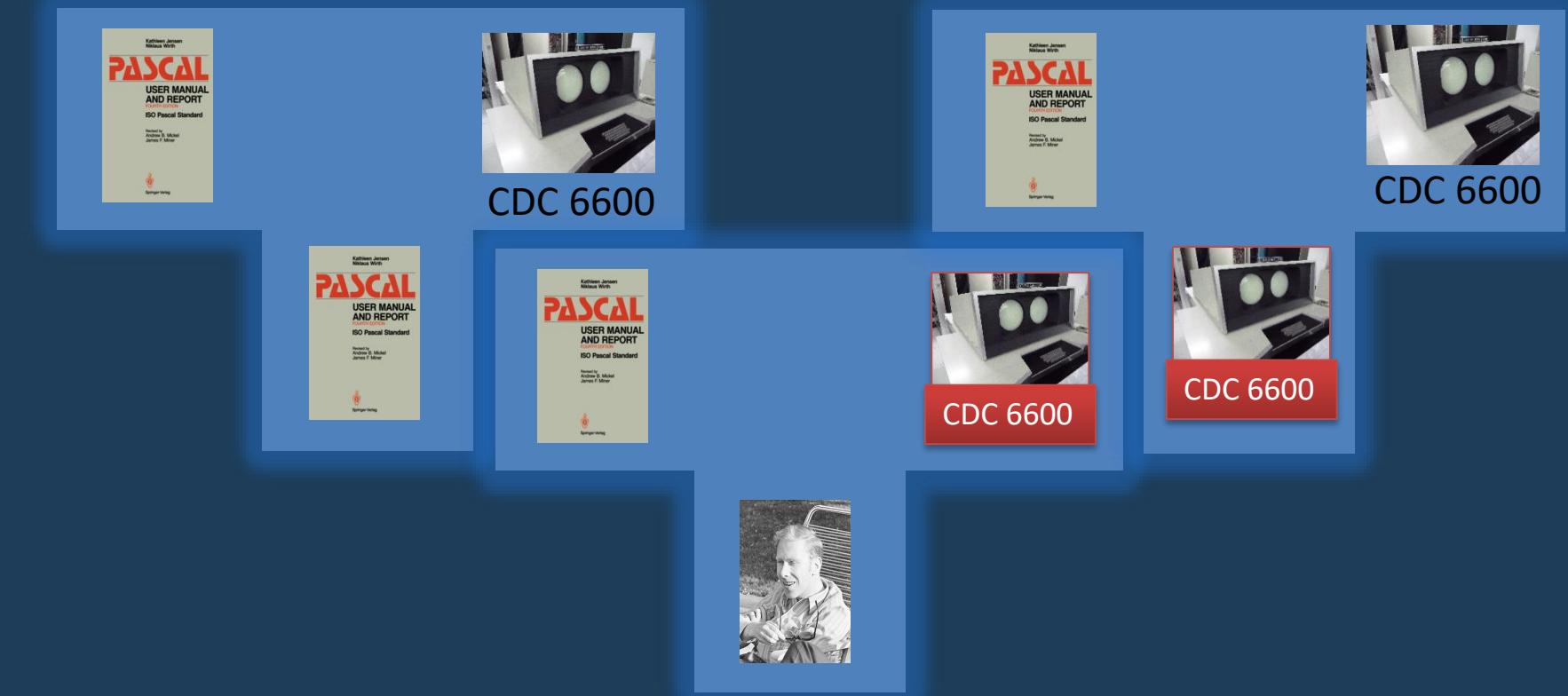


2

Al no haber compilador de Pascal, lo hace "a mano", sin embargo genera código poco optimizado.



El combinar los dos compiladores resulta un compilador lento pero que genera buen código



3

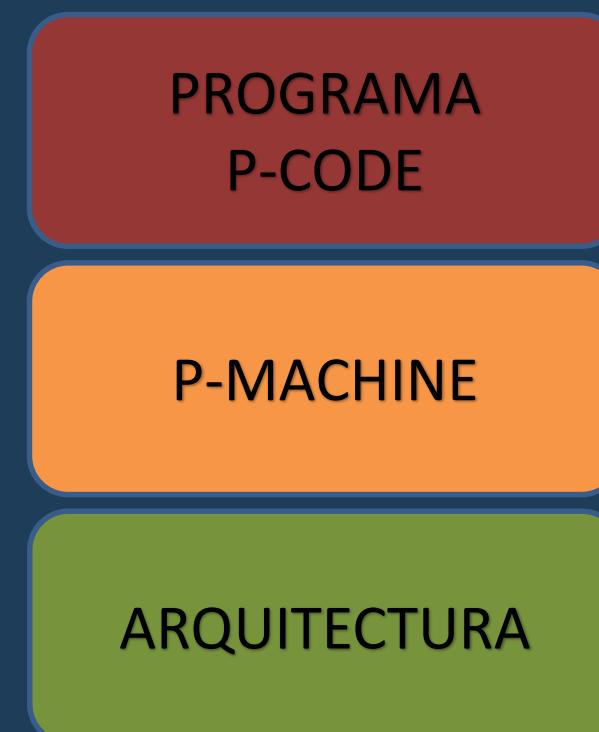
Para mejorarlo, se compila el fuente del compilador con él mismo. Obteniendo uno eficiente y que genera buen código



4

P-machine

**Concepto generico:
“Portable machine”**



1

Es un simulador para ejecutar P-code.

2

Maquina Virtual, que corre sobre alguna arquitectura.

3

Siempre y cuando se escriba el simulador completo.

Generalmente son simples y no son difíciles de hacer.

Recordar que existen máquinas de:

3 Direcciones

EJEMPLO :

ADD X Y Z

Probablemente signifique
realizar $X = Y + Z$

2 Direcciones

EJEMPLO :

ADD X Y

Probablemente signifique
realizar $X = X + Y$

1 Direcciones

EJEMPLO :

ADD X

Como funcionan los
acumuladores, en este
caso se le suma X a algún
registro por defecto

0 Direcciones

EJEMPLO :

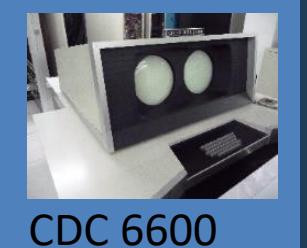
ADD

Utiliza la pila para realizar
la suma, hace pop de 2
valores de la pila y la suma
la ingresa nuevamente en
pila



Zurich P-machine

- Se hizo por preocupación a la dependencia del compilador con la CDC-6600
- Es una P-machine orientada a pila (0 direcciones).
- Programada en Pascal.
- Se compila con el compilador de Pascal que genera y esta escrito en CDC-6600 bueno.
- Resulta un simulador de P-Machine que corre en CDC-6600.



Adelantando materia...



Un compilador tiene:

Front-end

Es la parte que tiene que ver con el código fuente, el código que recibe

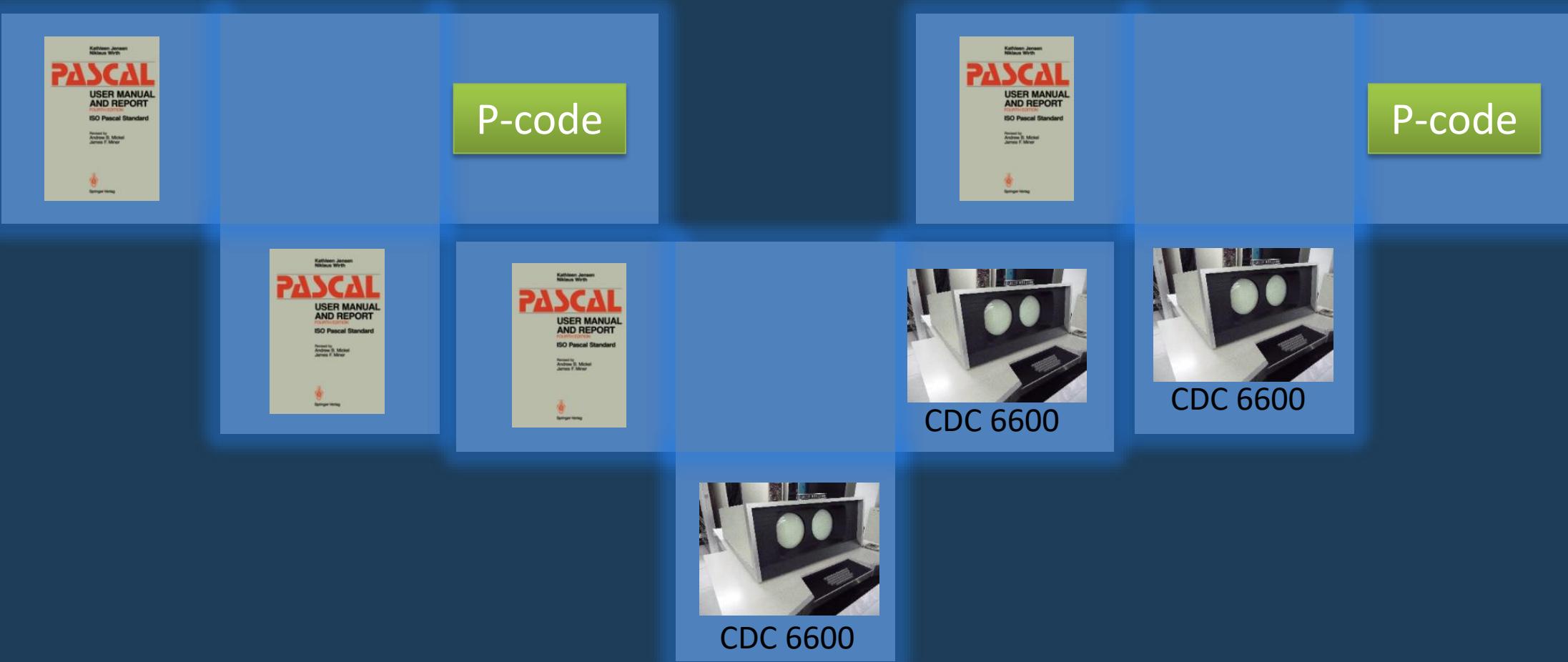
Back-end

Es la parte que tiene que ver con la arquitectura de destino, el código que genera



Generando P-code

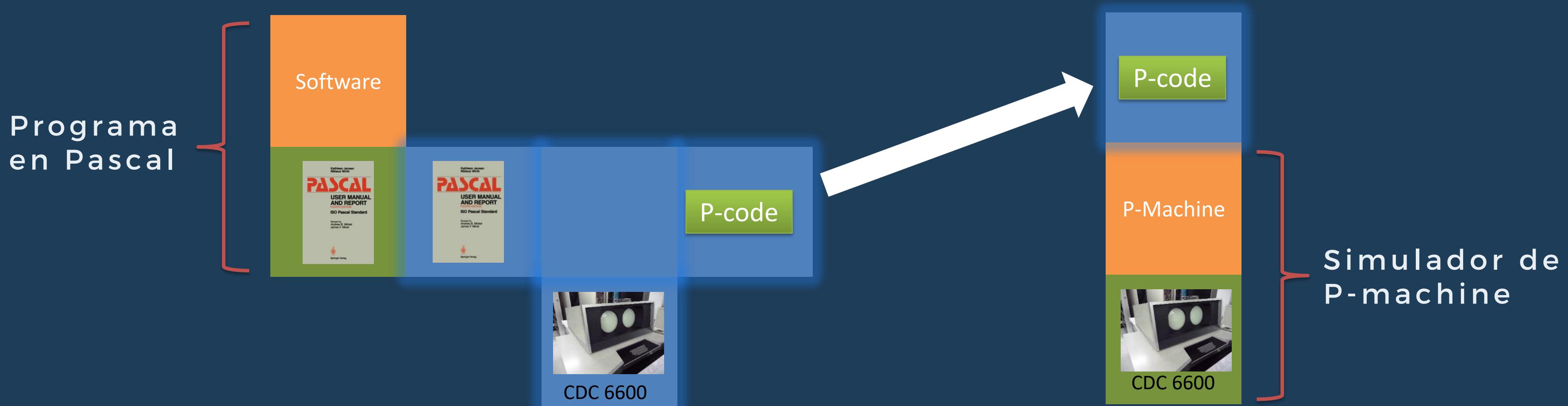
- Wirth modifica en compilador de Pascal, para generar P-code (modifica el back-end).
- Se compila con el compilador de Pascal que genera y esta escrito en CDC-6600 bueno.
- Da como resultado un...



...Cross-compiler
de Pascal que
genera P-code

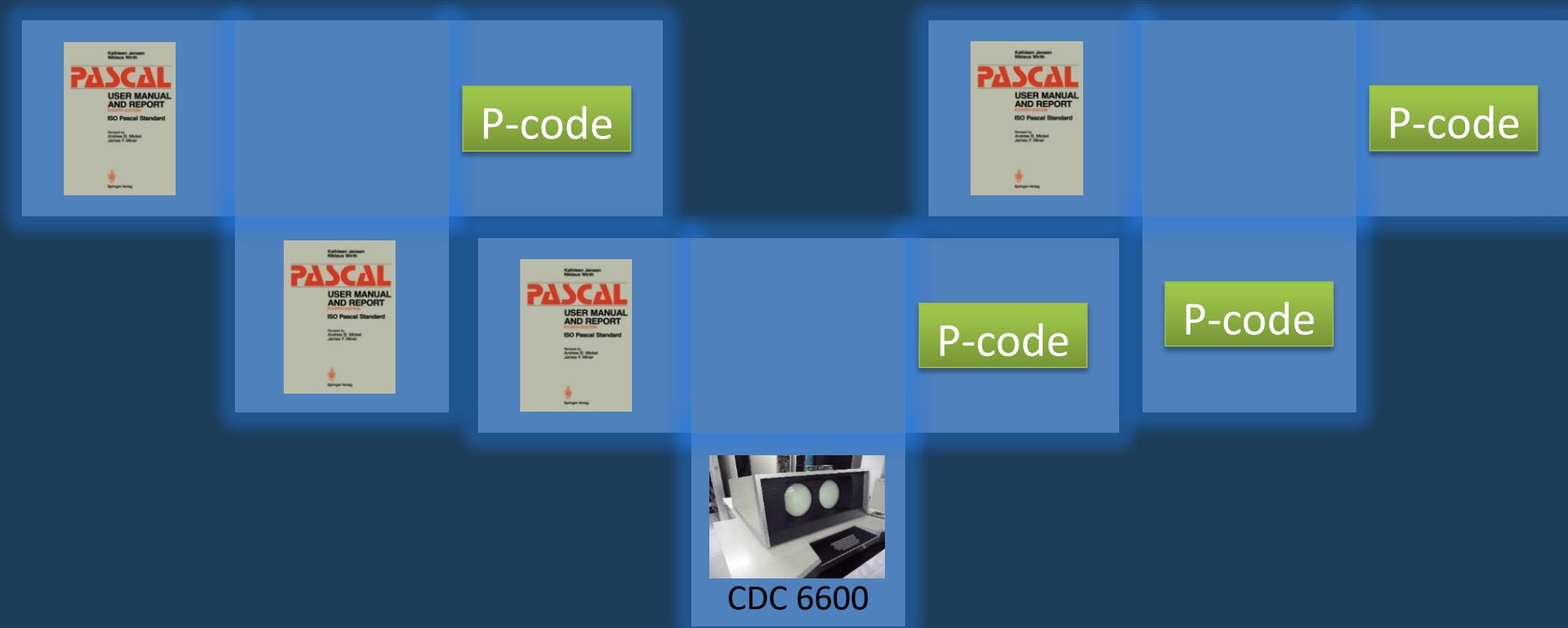
Software para P-machine

- Con el simulador de P-machine corriendo en CDC-6600
- Ahora puede escribir programas en Pascal para correr en P-machine.
- Solo se debe compilar con el cross compiler
- Y el código generado corre en el simulador de P-machine.



Compilando el compilador(de nuevo)

- 1 Compilar el compilador de Pascal con el cross compiler
- 2 Se obtiene un compilador de Pascal de P-machine que genera P-machine.

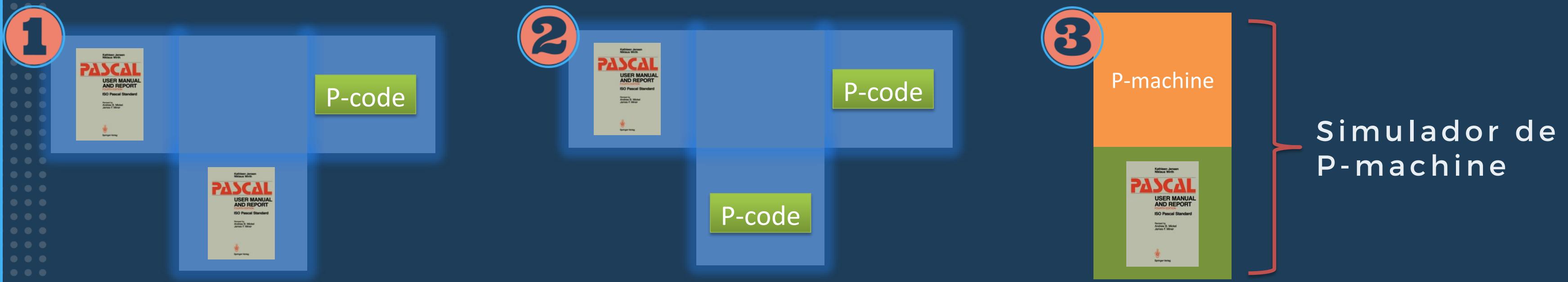


Ah ~~shit~~, here we go again.



Zurich Compiler kit

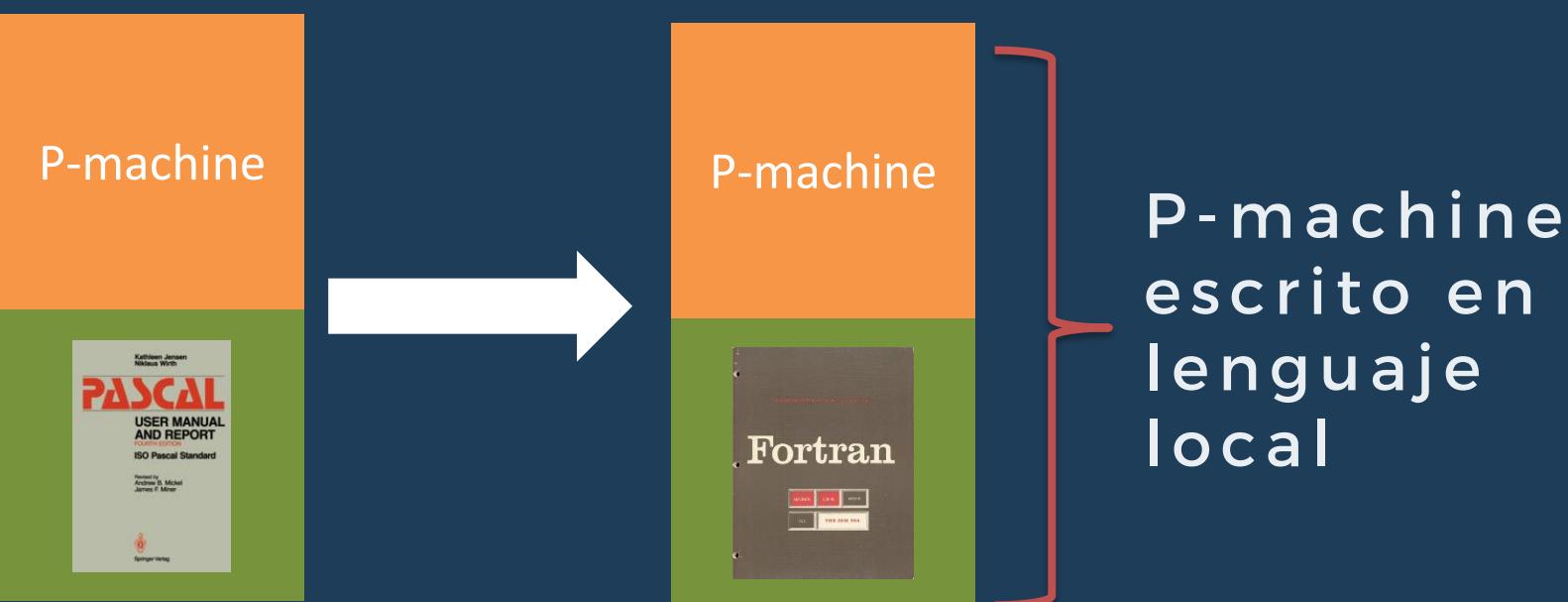
- Wirth y su grupo enviaba su “compiler kit” a quien lo pedia, e incluia:



Que hacer?

Paso 1

- Estudiar el intérprete de P-machine
- El cual es pequeño y esta en Pascal(legible)
- Reescribir “a mano” en algún lenguaje disponible



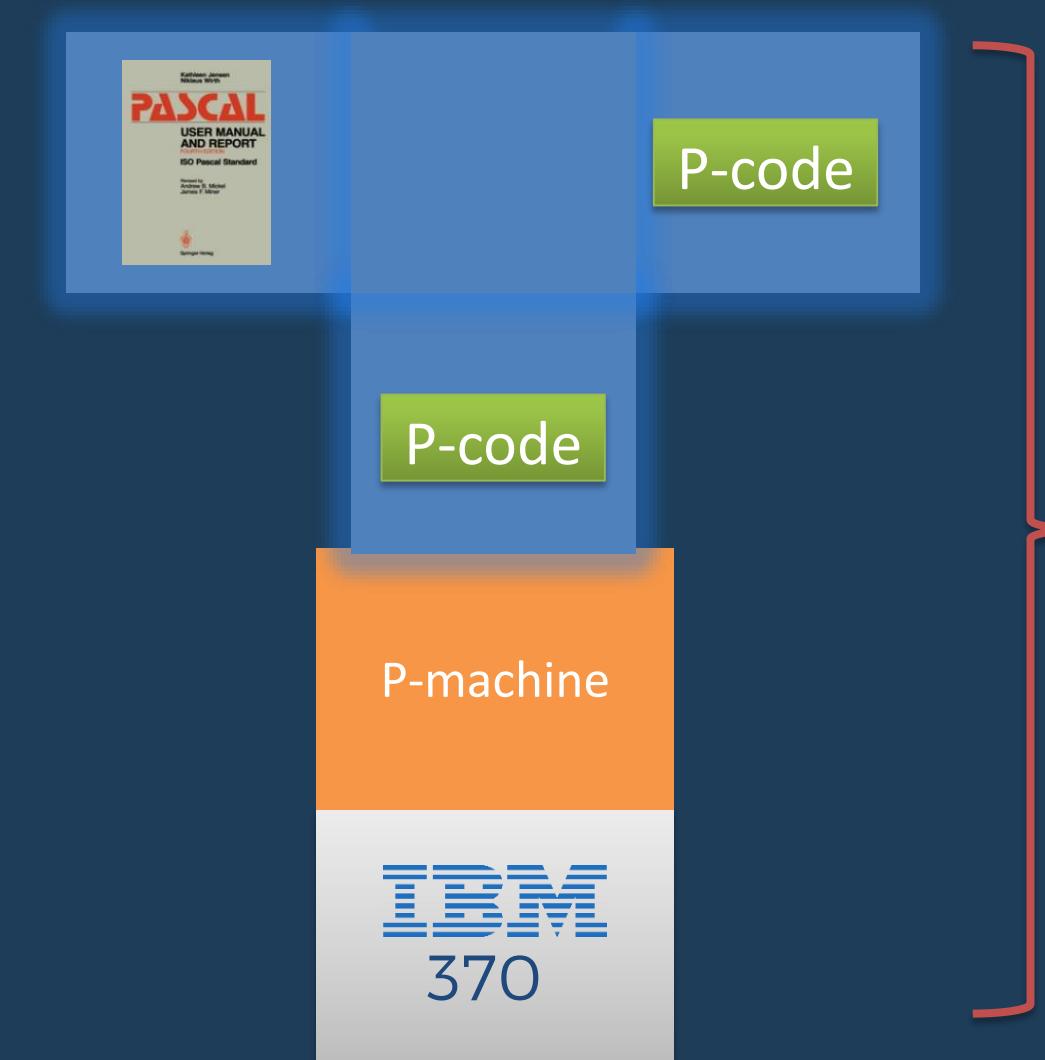
Paso 2

- Con el interprete de P-Machine escrito en lenguaje local
- Compilar hacia maquina actual con un compilador existente
- Como resultado, simulador de P-machine corriendo en hardware local



Paso 3

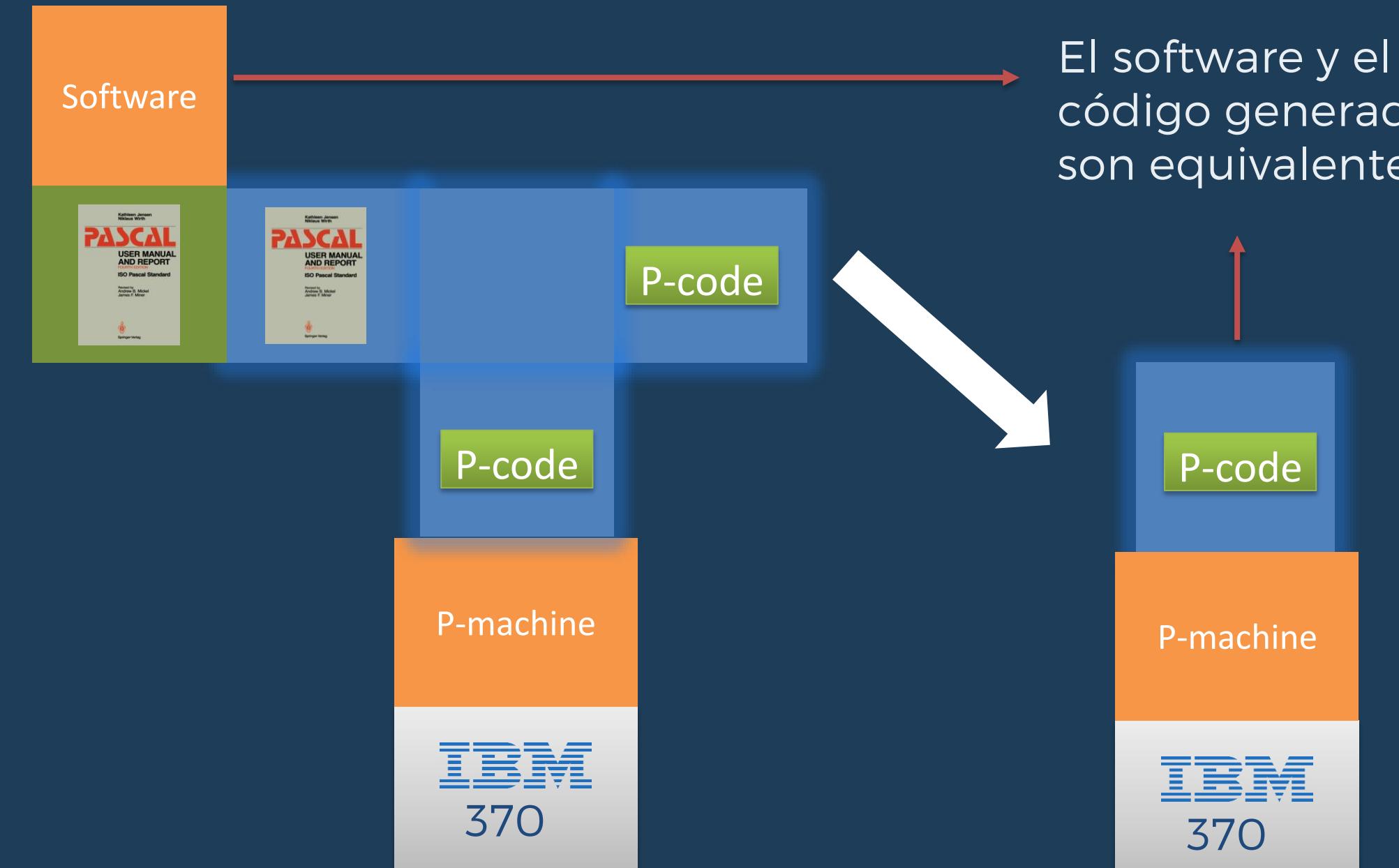
- Se tiene el P-machine funcional
- Se podría correr P-code
- Sin embargo, lo único que se tiene del “Zurich compiler kit” es el compilador de Pascal escrito en P-code
- Por lo que se puede programar y compilar , pero todo corre sobre la P-machine.



IBM-370,
corriendo el
simulador de
P-machine y
sobre esto el
compilador

Paso 4

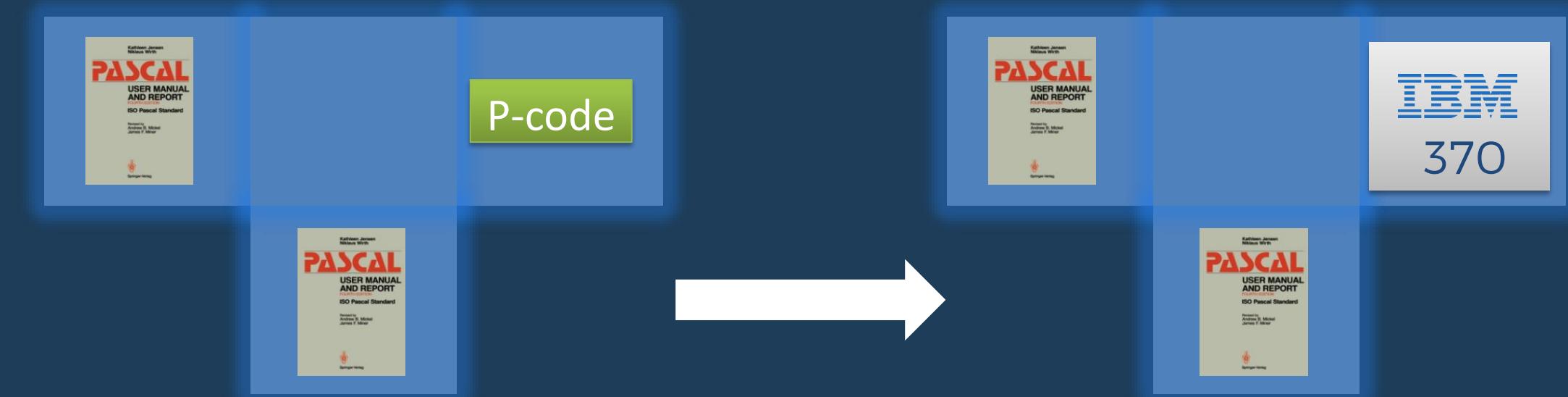
- Ahora se puede escribir software en Pascal y compilarlo para que corra sobre P-machine



El software y el código generado son equivalentes

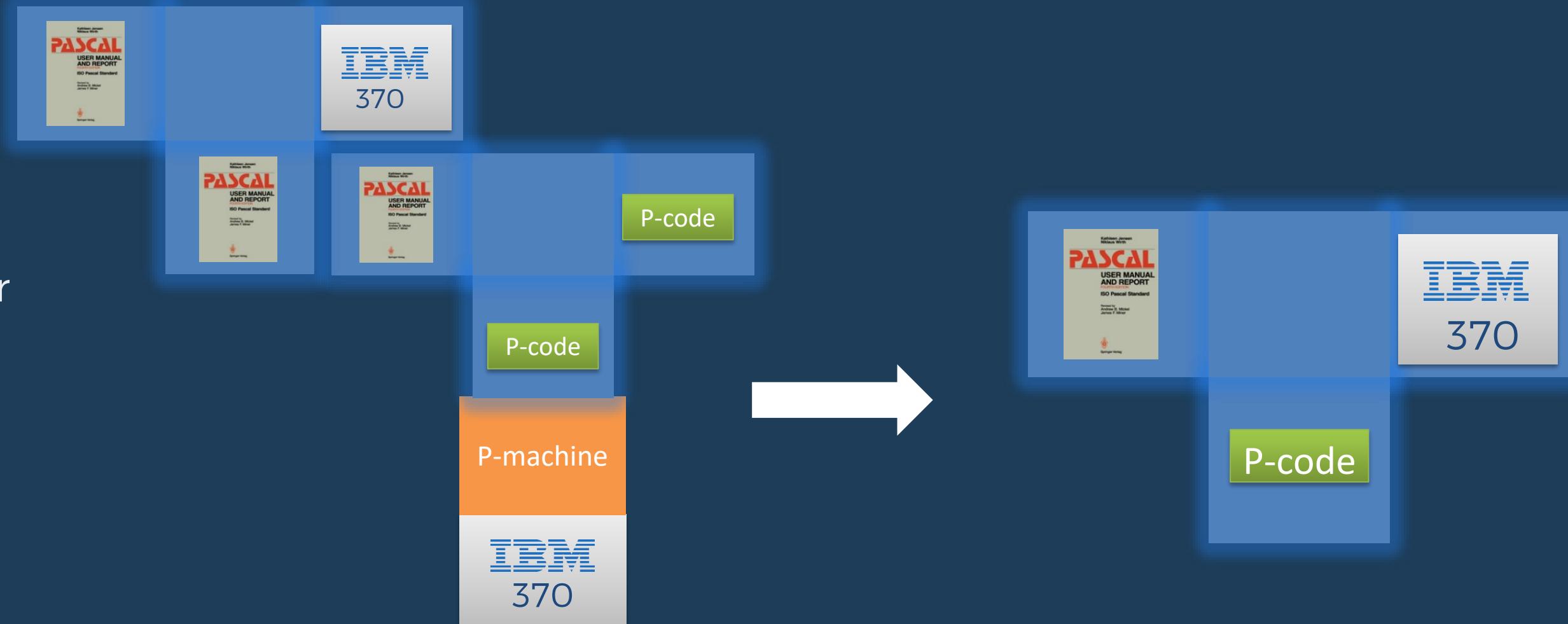
Paso 5

- Estudiar el compilador de Pascal escrito en Pascal.
- Aunque no es trivial, esta escrito en Pascal(legible)
- Modificar “a mano” para que genere lenguaje de la maquina actual.



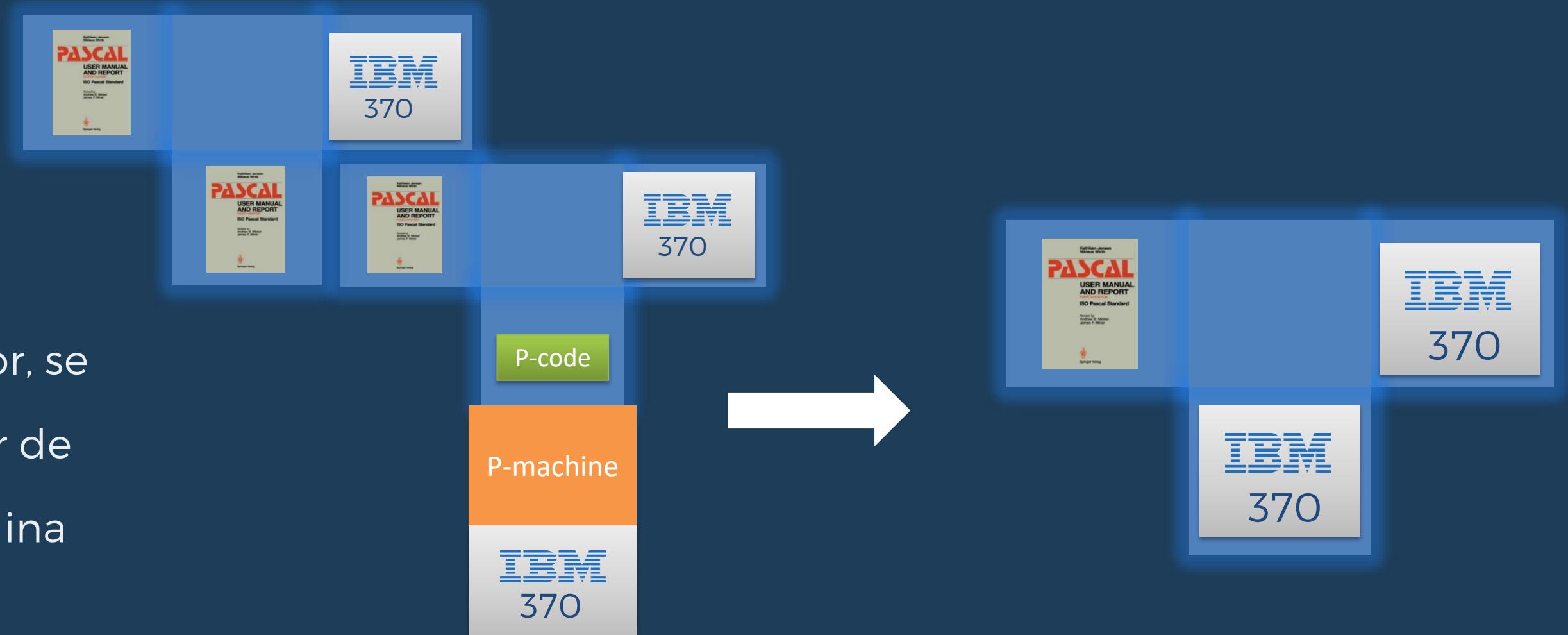
Paso 6

- Compilar el nuevo compilador
- Se obtiene el compilador de Pascal, escrito en P-code y genera 370



Paso 7

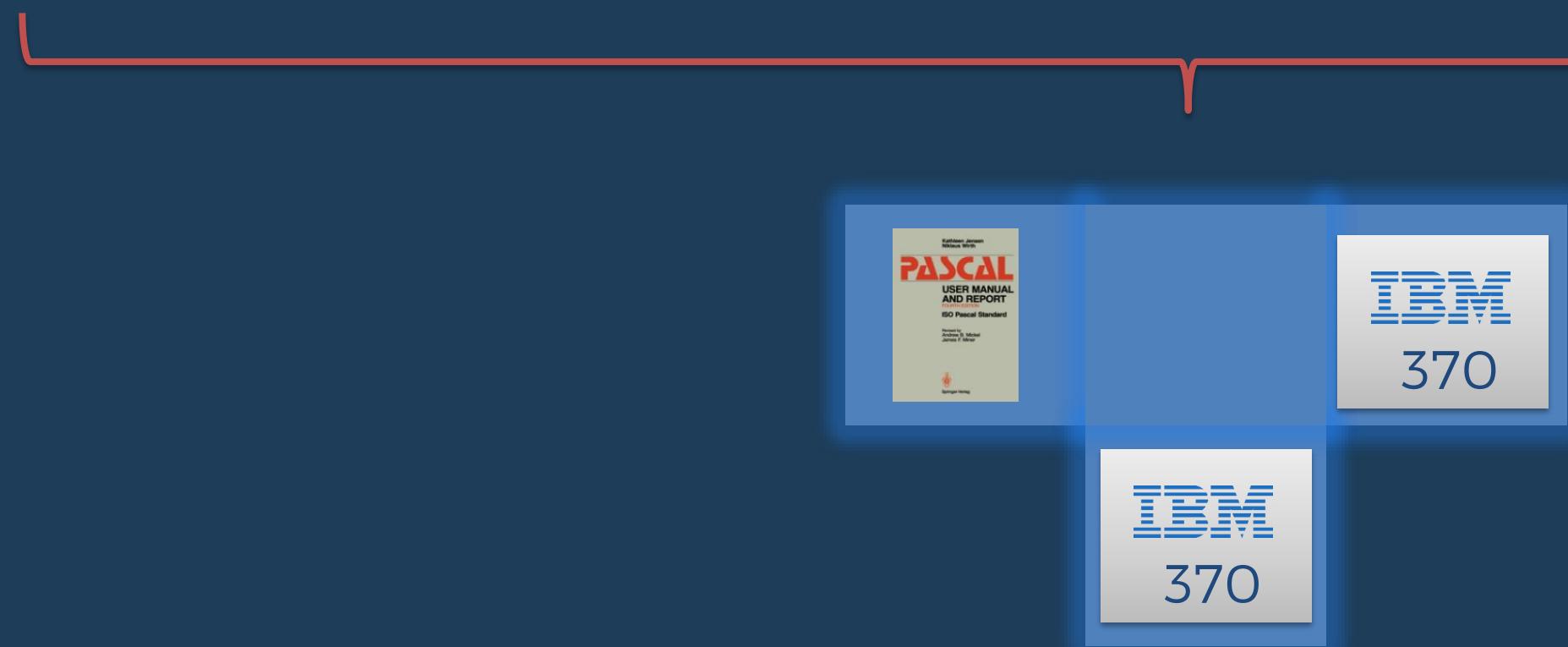
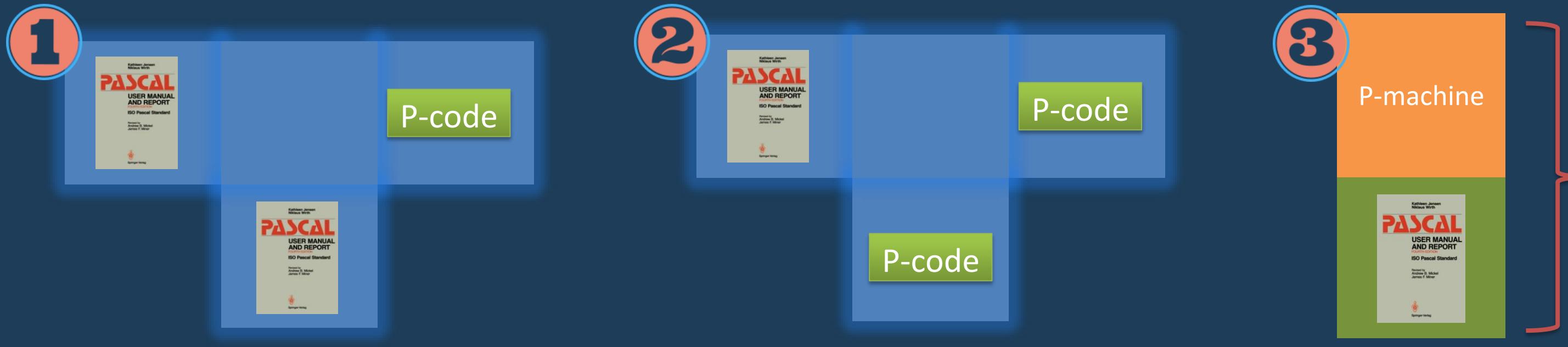
- Compilar el nuevo compilador :)
- Ahora ni siquiera se necesita el simulador, se tiene un compilador de Pascal para la maquina

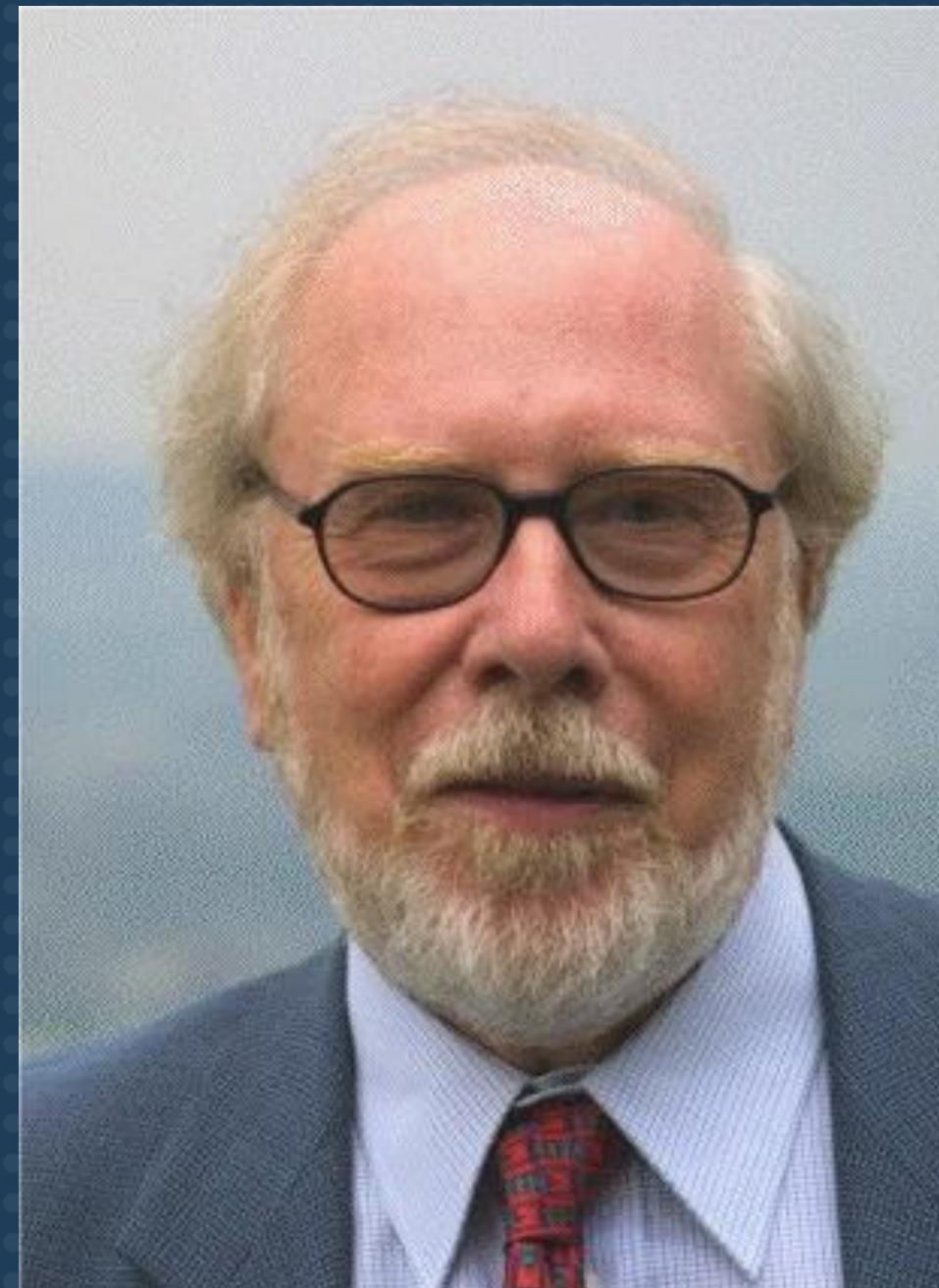


Wey ya

Resultado final...

Del “Zurich compiler kit” a...





Niklaus Wirth

Turing Award 1984



Programas relacionados con compiladores



- Mucho software del Sistema se relaciona con compiladores

- Los sistemas operativos y lenguajes de alto nivel han sido muy exitosos

- Desarrollo de software actual totalmente asociado a su uso

- Las técnicas de compiladores influencian otros programas

Ensambladores

- Inventados en los años 50's
- Aún viven.
- Programación de bajo nivel
- Relacion 1 a 1 con el lenguaje de maquina.
- Código eficiente y código simple.
- Usualmente usan técnicas de compiladores.
- Lo mas típico es que sea de 2 pasadas:
 - Creacion de tabla de simbolos
 - Generar código
- Para ensambladores de 1 pasada se usa Backpatching
- Hay que saber ensamblador para escribir un compilador

```
.file "hola.c"
.section .rodata
.LC0:
.string "Hola elbinario"
.text
.globl main
.type main, @function

main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
movl $.LC0, %edi
call puts
movl $0, %eax
popq %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE0:
.size main, .-main
.ident "GCC: (SUSE Linux) 4.8.5"
.section .note.GNU-stack,"",@progbits
```

En backpatching se
dejan pendientes la
localización de
símbolos en la tabla
de símbolos, cuando
se encuentre se
vuelve a actualizarla

Desensambladores

- Hace el proceso inverso del ensamblador
- Toma el lenguaje de maquina y lo muestra en ensamblador
- Para que podria usarse?
 - Piratear software
 - Estudiar y aprender software que ya esta implementado
- Etiquetas y nombres de variables arbitrarias, no es tan legibles ya que no se sabe que nombres tenian.
- Similar a un simulador:
 - Debe conocer formatos binaries
 - Reconocer codigos de operacion y formatos de direccionamiento
- De 2 pasadas aunque Tambien se puede usar Backpatching
- Difícil saber si hay macros

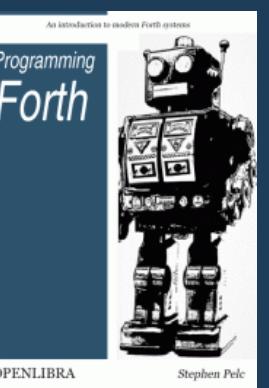
The screenshot shows a web-based disassembler interface. At the top, it says "onlinedisassembler.com/odaweb/run_hex — ODA — Online Disassembler". Below that is a navigation bar with "ODA", "Copy Hex", "File Upload", "Blog", and "Contact Us!". A gear icon is next to the "Select Your Architecture:" dropdown, which is set to "i386:x86-64". There's also an "Options" button. The main area has tabs for "Hexadecimal Only" and "Put binary below". A text input field contains the hex code: 55 31 D2 89 E5 8B 45 08 56 8B 75 0C 53 8D 58 FF 0F B6 0C 16 88 4C 13 01 83 C2 01 84 C9 75 F1 5B. To the right, there's a scrollable list of assembly instructions:

```
.data:0x00000000      55          push    %rbp
.data:0x00000001      31d2        xor     %rdx,%rdx
.data:0x00000003      89e5        mov     %esp,%rbp
.data:0x00000005      8b4508      mov     %eax,%rbp
.data:0x00000008      56          push    %rsi
.data:0x00000009      8b750c      mov     %esi,%rsi
.data:0x0000000C      53          push    %rbx
.data:0x0000000D      8d58ff      lea     -0x1(%rax),%rbx
.data:0x00000010      0fb60c16   movzbl (%rsi,%rdx,1),%rcx
.data:0x00000014      884c1301   mov     %cl,0x1(%rbx,%rdx,1)
.data:0x00000018      83c201      add    $0x1,%rdx
.data:0x0000001B      84c9        test   %cl,%cl
.data:0x0000001D      75f1        jne    loc_00000010
.data:0x0000001F      5b          pop    %rbx
```



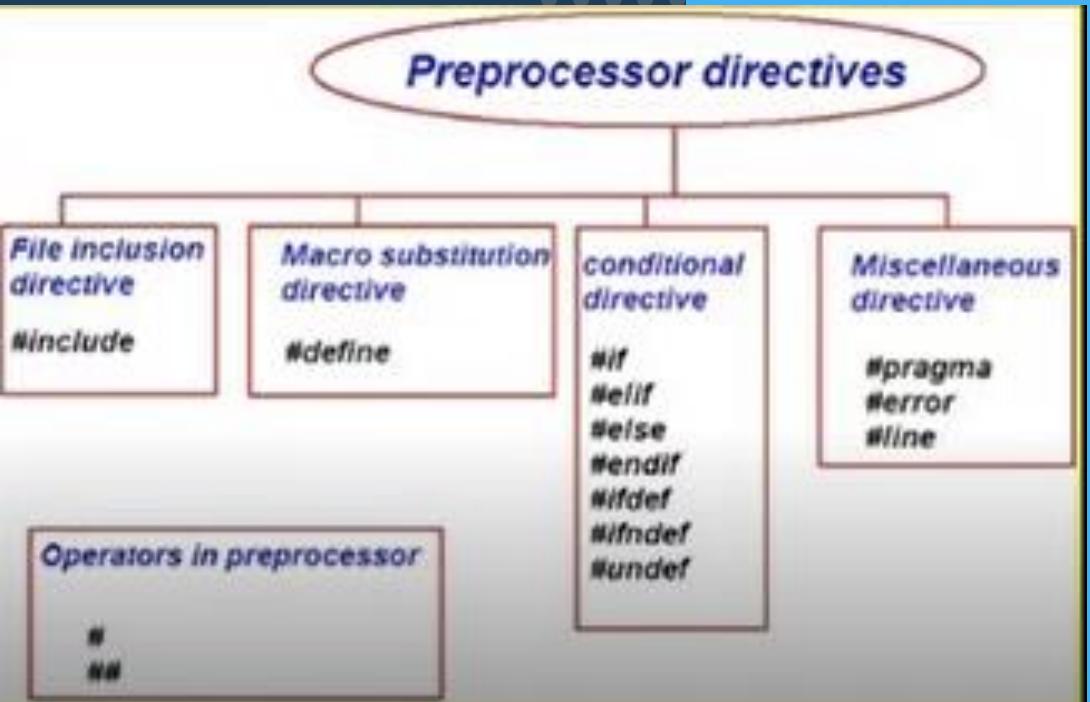
Intérpretes

- Combina traducción y ejecución linea por linea.
- Usa muchas tecnicas de compiladores.
- Muy independiente de la arquitectura.
- Run-time environment.
- Lentos en ejecución.



Preprocesadores

- “Antes del proceso”.
- Ocurre antes de la compilación.
- Funciones:
 - Incluir archivos
 - Reemplazo de hileras
 - Macros
 - Eliminar comentarios
 - Compilación condicional
- Se considera buen estilo de programación.
- Usa técnicas de compilación .
- Son independientes del lenguaje.

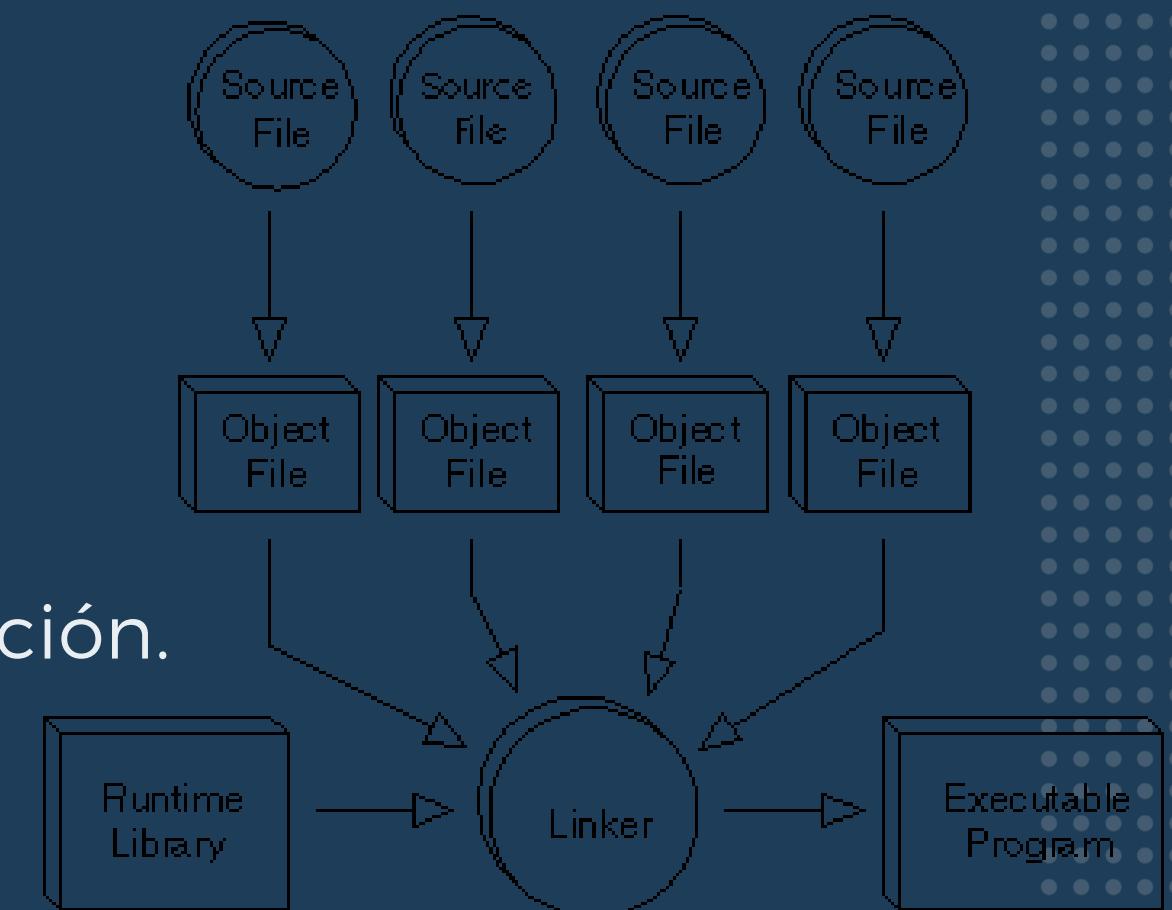


En c:
• #include, incluye declaraciones, se puede ver como insertar un archivo sobre el que se está.



Linker

- “Ligador” o “Eslabonador”.
- Al compilar o ensamblar se generan “módulos objeto”.
- Programas o funciones compilados independientes.
- Bibliotecas de funciones.
- El linker debe crear un solo módulo ejecutable
- Integrado en el compilador.
- Existen:
 - Linking estático: ocurre en tiempo de linking
 - Linking dinámico: ocurre en tiempo de ejecución.

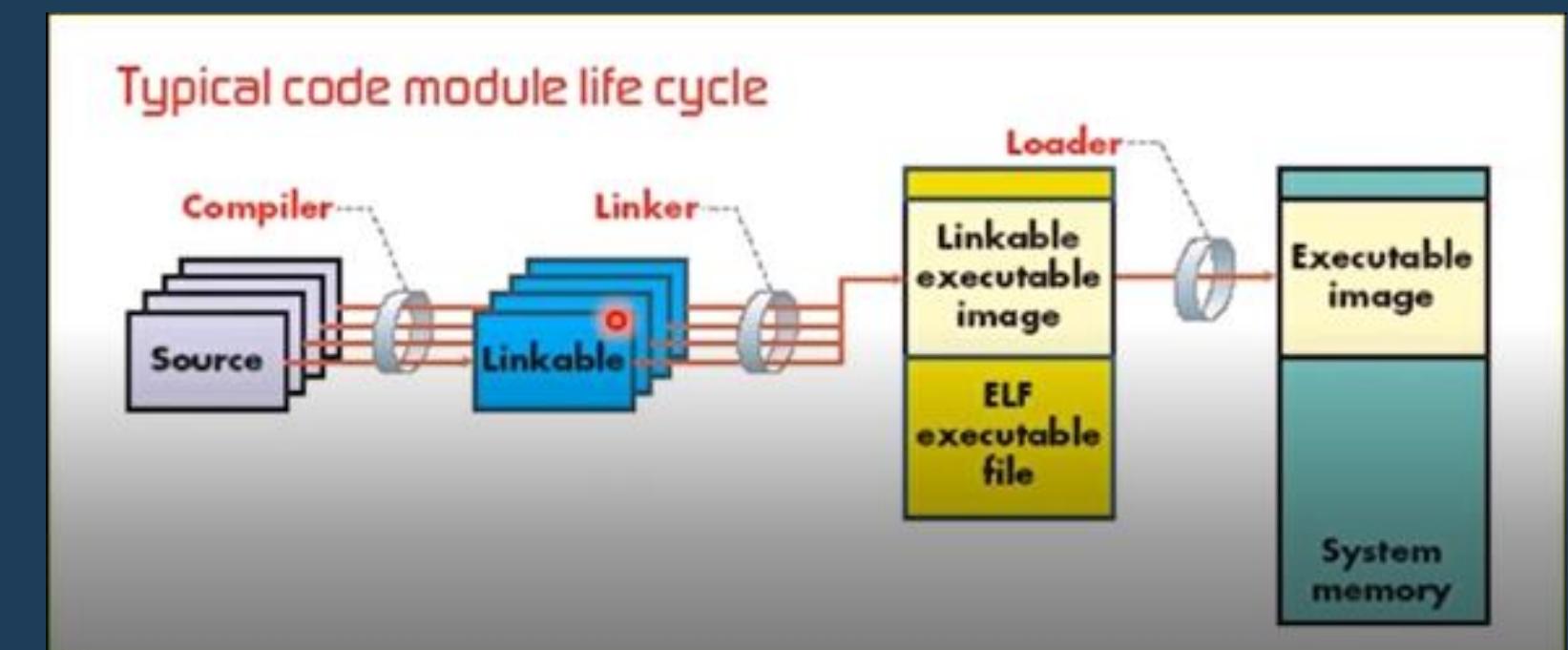


Los DLL de Windows son linking dinámico

Los ejecutables son más pequeños en linking dinámico.
El linking estático consume más memoria, pero es más rápido que el dinámico

Loader

- Toma un modulo ejecutable y lo carga en memoria.
- Relocalizar
- Normalmente es parte del Sistema Operativo
- Negocia espacio, consigue memoria para que pueda ejecutarse
- Crea estructuras de control
- Process Control Block(PCB): estructura de datos del sistema operativo donde esta la informacion acerca un proceso para administrarlo.





Debugger

- Software que permite ejecutar de forma controlada otro programa.
- Instrucción por instrucción.
- Permite visualizar el contenido de variables.
- Hace uso de break points.
- El compilador le da información al debugger, como la table de símbolos

The screenshot shows the Microsoft Visual Studio IDE during a debugging session. The main window displays a C# file named Program.cs with the following code:

```
10 public static string GetName() => "Maarten";
11 
12 public static string GetWord()
13 {
14     return "hello";
15 }
16 
17 public static void Main(string[] args) args = {string[0]}
18 {
19     var name = GetName(); name = {null}
20     var word = GetWord();
21 
22     var result = $"{name} says {word}!";
23 
24     Console.WriteLine(result);
25 }
```

The code editor highlights line 20 with a yellow background. A red breakpoint icon is visible on line 19. The Locals window at the bottom left shows the current values of the variables:

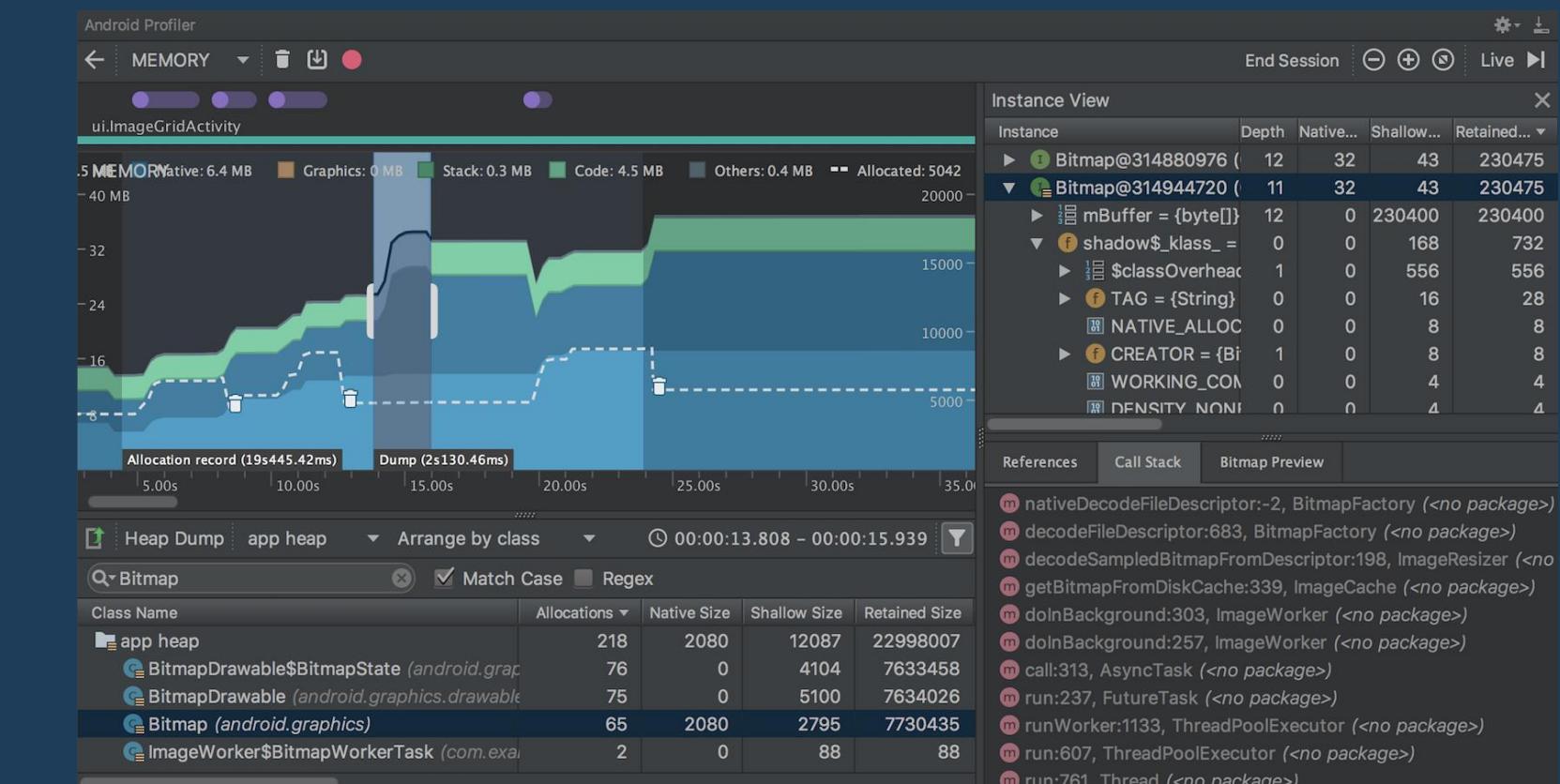
Name	Type	Value
args	string[]	{string[0]}
name	string	null
word	string	null
result	string	null

The Call Stack window on the right shows the current call stack entry:

Name	Lang
DebuggingDemo.exe!DebuggingDemo.Program.Main()	C#

Profiler

- Similar al debugger, en que es software que ejecuta otro.
- Se encarga de recolectar estadísticas de la ejecución, genera un perfil de un programa.
- Por ejemplo:
 - Cantidad de llamadas a una función
 - Duración promedio de llamadas
 - Porcentajes del tiempo de ejecución
- Permite saber si es necesario optimizar el código y que parte específicamente.
- Necesita información creada por el compilador.
- El compilador se puede ver beneficiado, para que este, por sí solo, utilice técnicas de optimización (recompilación automática).



Consideraciones

- 1** Pendiente el proyecto compilador MICRO
- 2** Lectura #3 de Pinker – 25/09/20

Apuntes 18-sept

Kendall Tames Fernández
2018112153



Instituto
Tecnológico de
Costa Rica

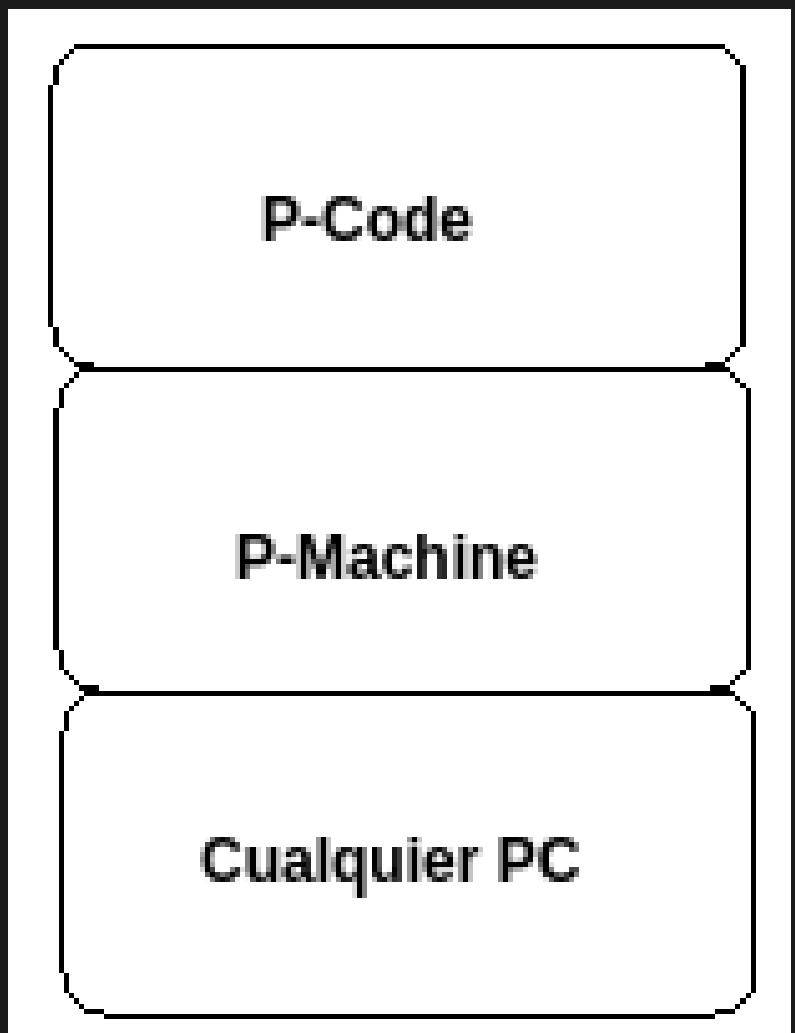
P-Machine

Se quería tener portabilidad al escribir código

Ejecutaba p-code, que es el ASM de un procesador hipotético

Sumamente sencillas pero 100% funcionales

Era una máquina virtual, así que cualquier PC que la tuviera podría correr P-Code



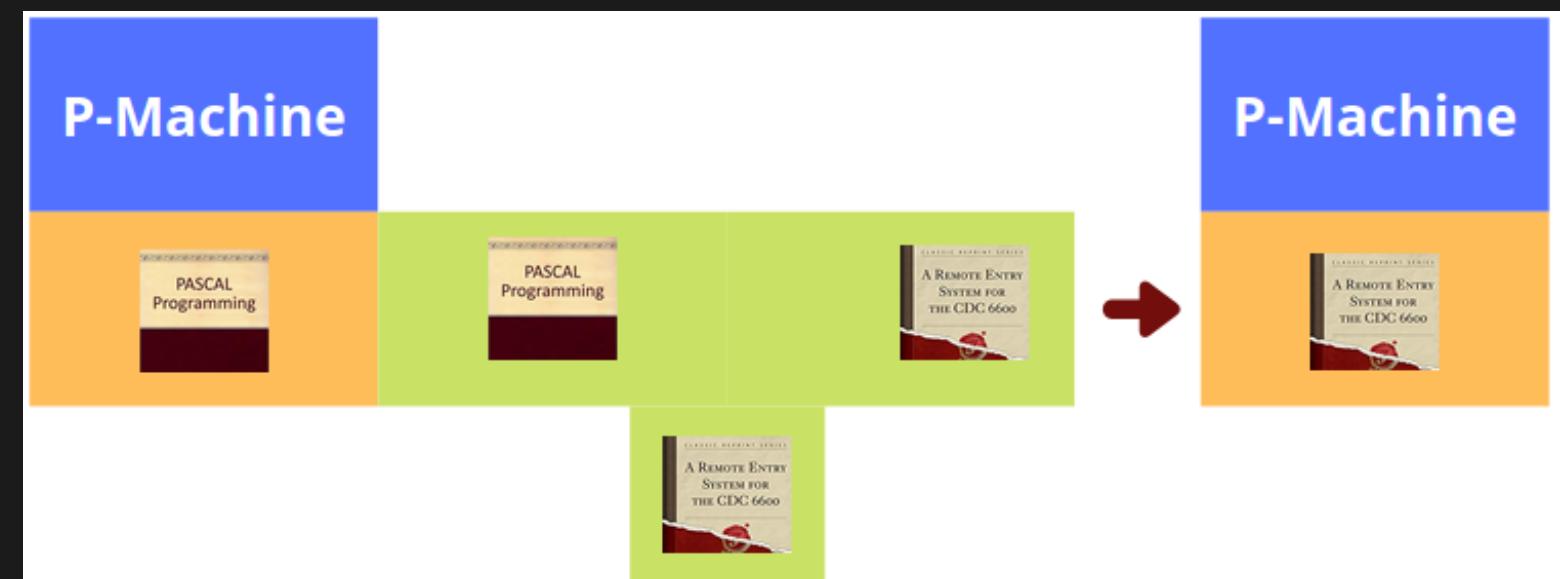
Zurich P-Machine

Creada por Niklaus Wirth en 1975 para evitar las dependencias que existían con el procesador CDC-6600

Era una máquina orientada a pila, también llamada de cero direcciones (explicación en la siguiente diapositiva).

Estaba programada en Pascal, esto ayudaba a que el código fuera muy legible. Además, junto con la naturaleza de cero direcciones, se podía reproducir en otro lenguaje con relativa facilidad.

Fue compilada con el mejor compilador de Pascal que tenían de Pascal --> CDC 6600.



Cantidad de direcciones

Tres direcciones

add x, y, z

Suma **y** y **z**. Deja el resultado en **x**.

Dos direcciones

add x, y

Suma **x** y **y**. Deja el resultado en **x**.

Una dirección

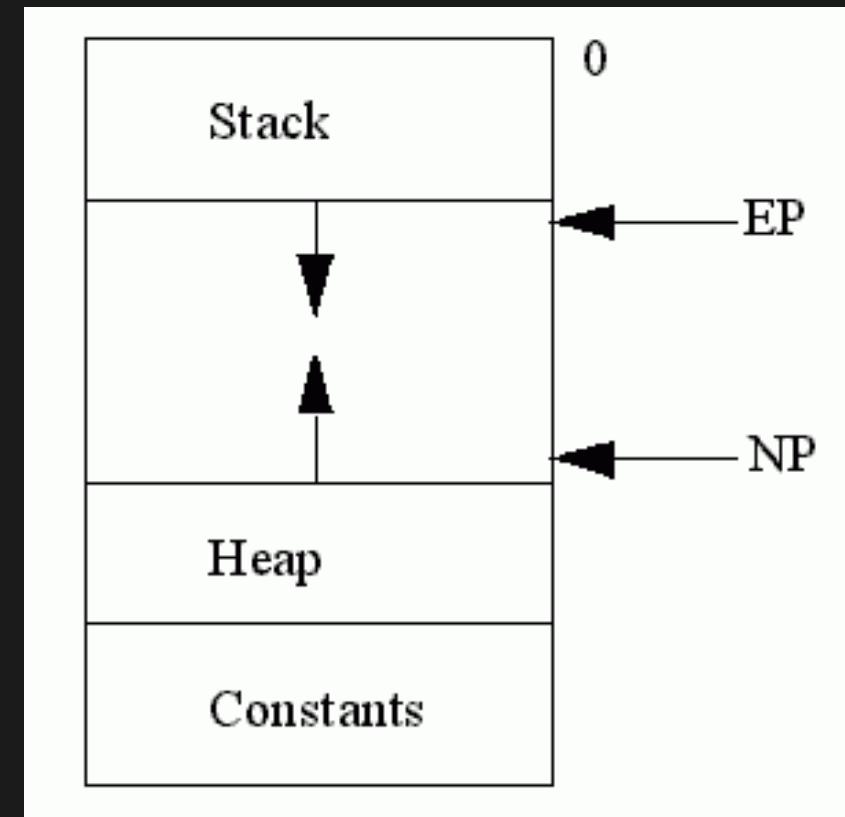
add x

Estas tienen un **contador**, así que le suma **x** a lo que ya está guardado

Cero direcciones

add

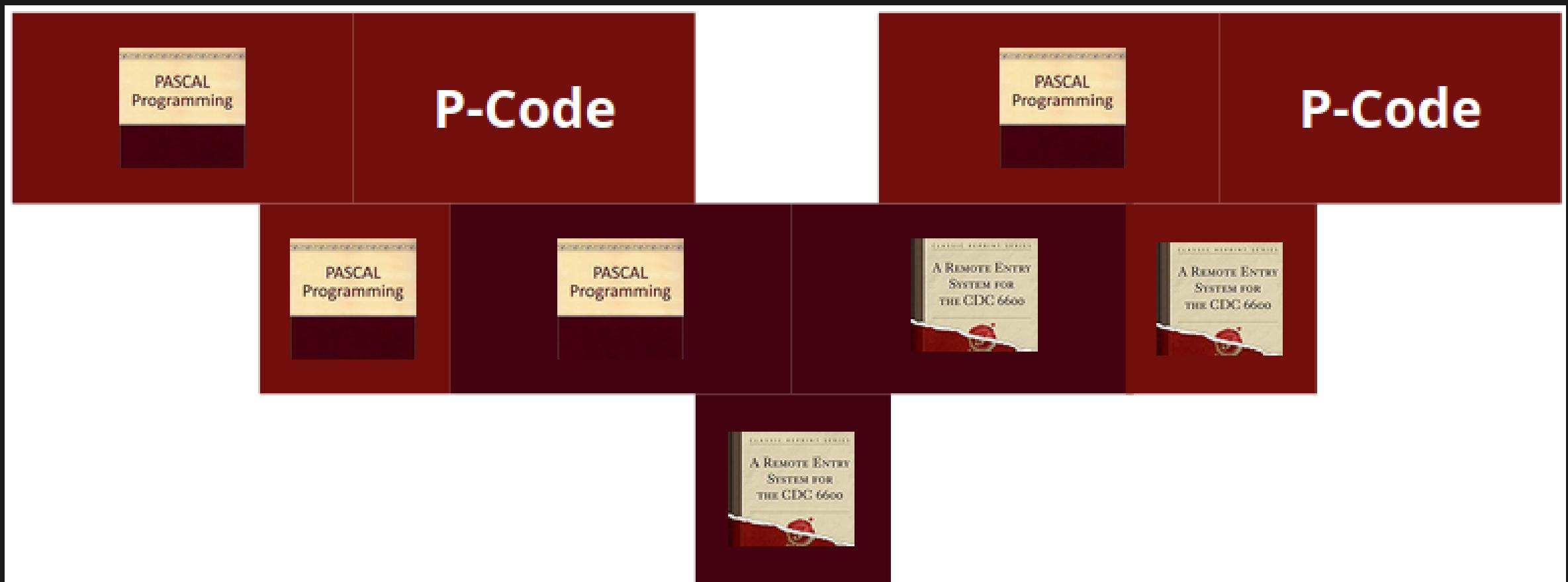
Se manejan completamente por la **pila**. Hace **pop** al primer operando, hace **pop** al segundo, y al resultado le hace un **push**.



Generar P-Code

Se quería tener una forma de pasar de PASCAL a P-Code.

Se logró modificando el back-end del procesador de PASCAL, que originalmente generaba CDC-6600.



En un compilador:

Front end = Código que recibe

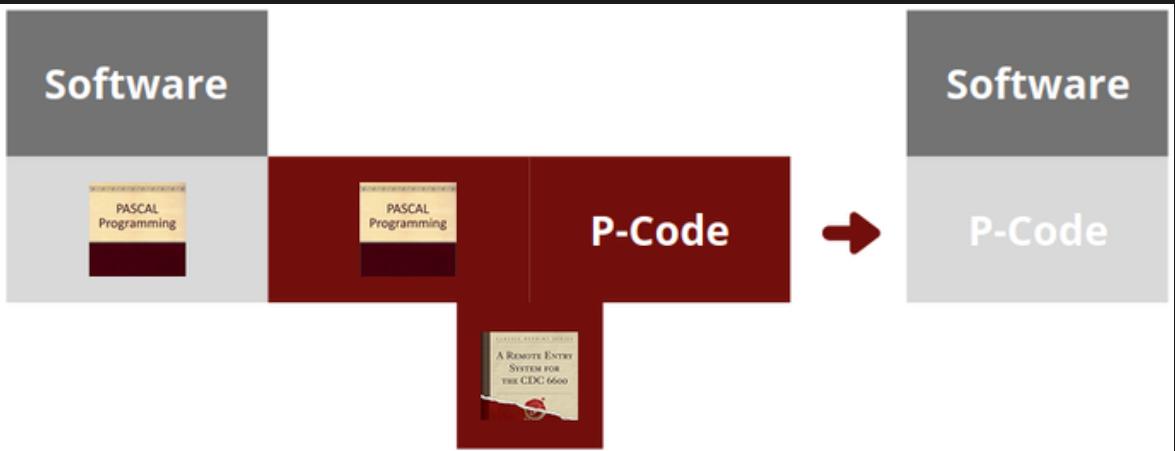
Back end = Código que genera

Cuando se tuvo, se compiló con el mejor compilador disponible de PASCAL a CDC-6600.

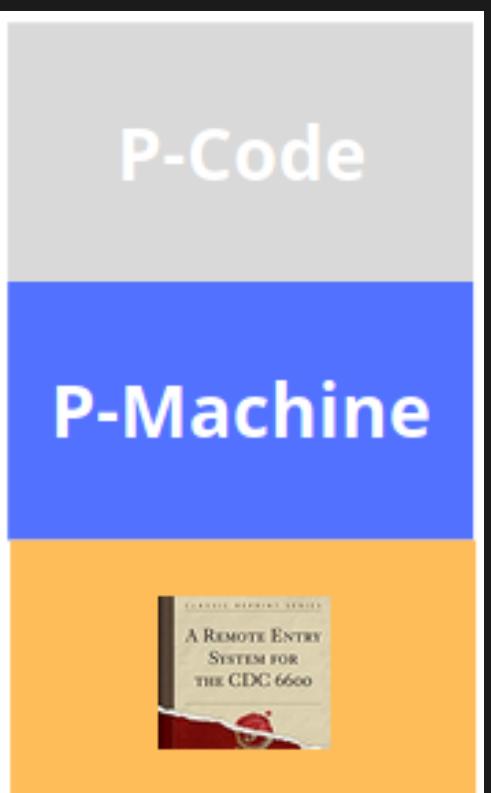
El resultado es un Cross-Compiler de Pascal que genera P-Code

Software para P-Machine

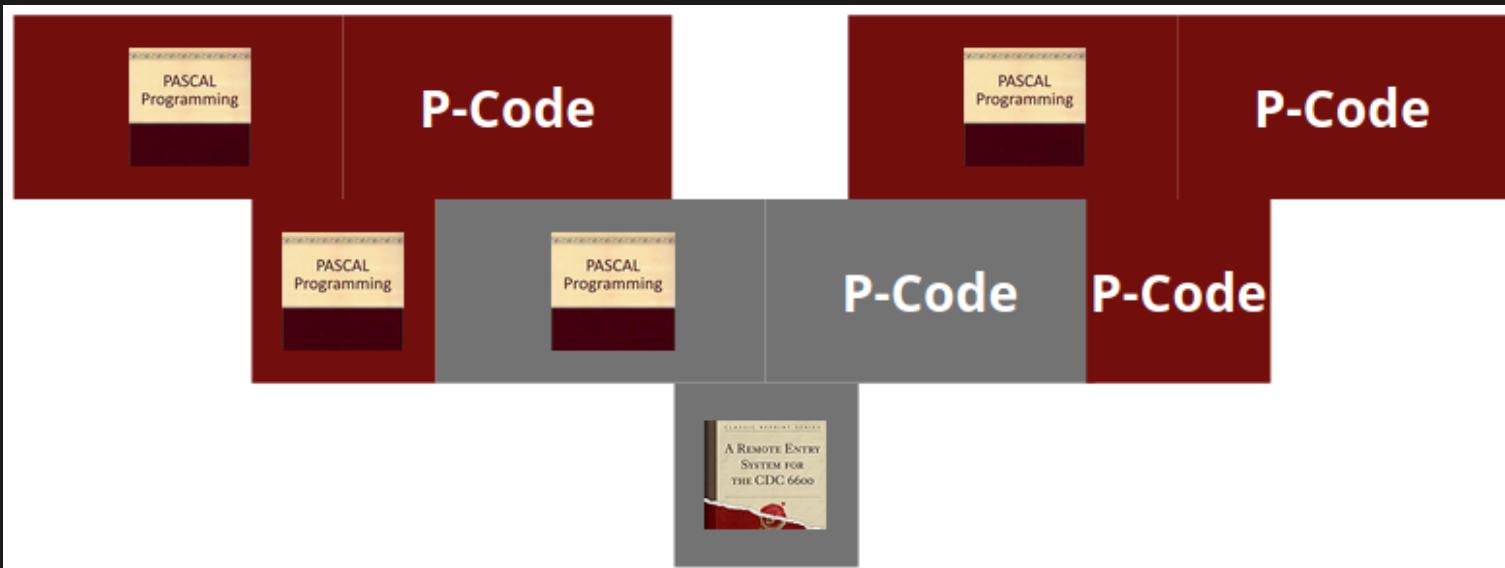
Con el resultado anterior, era posible compilar programas de Pascal para que estuvieran escritos en P-Code.



Con esto, la P-Machine podía ejecutar ese software.



Compilando el compilador



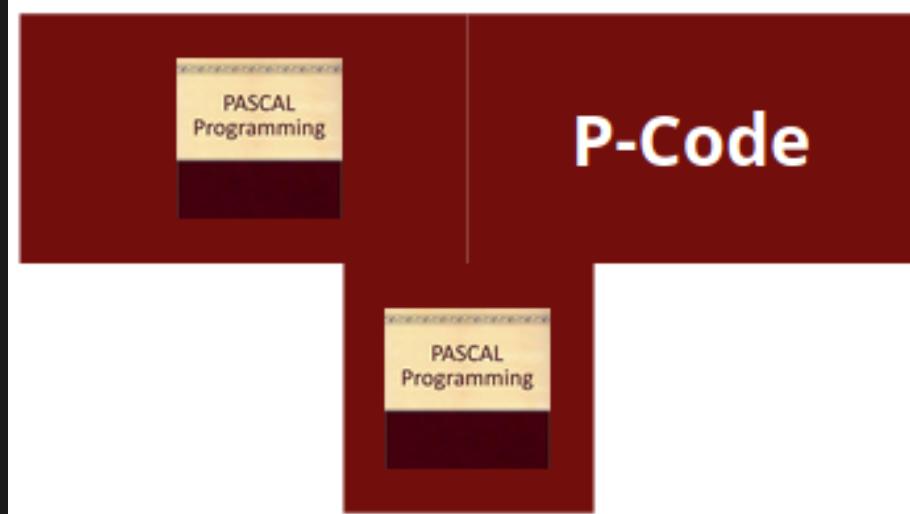
Se compiló el compilador de P-Code que estaba escrito en PASCAL para que ahora estuviera escrito en el mismo P-Code. Todo esto para que ahora pudiera **correr sobre la P-Machine**.

Zurich Compiler Kit

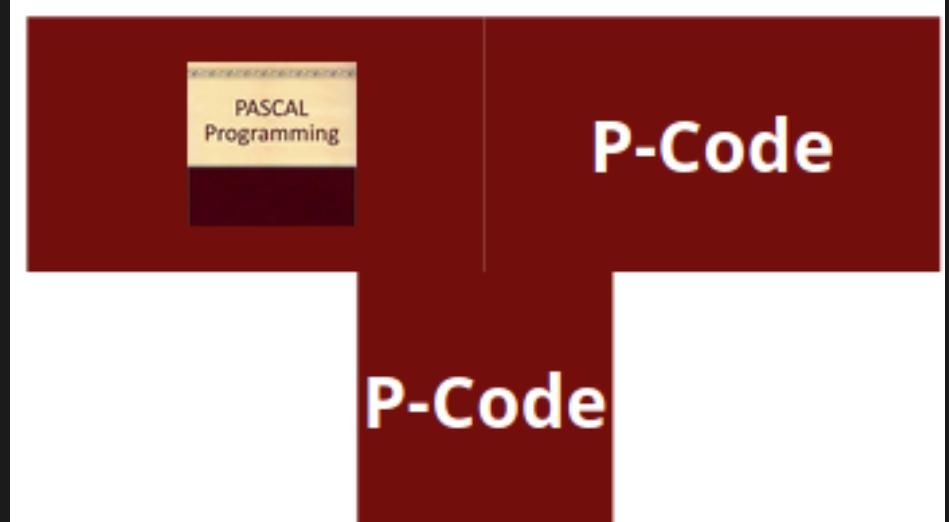
Como eran bien generosos, se armaron un Kit para poder distribuir el P-Code y que PASCAL dominara el mundo. Sí, todo lo que se quería era que todo el mundo pudiera correr PASCAL sin importar si tenían CDC-6600 o no. No cobraban nada más que el envío por correo.

El Kit tenía los siguientes tres componentes:

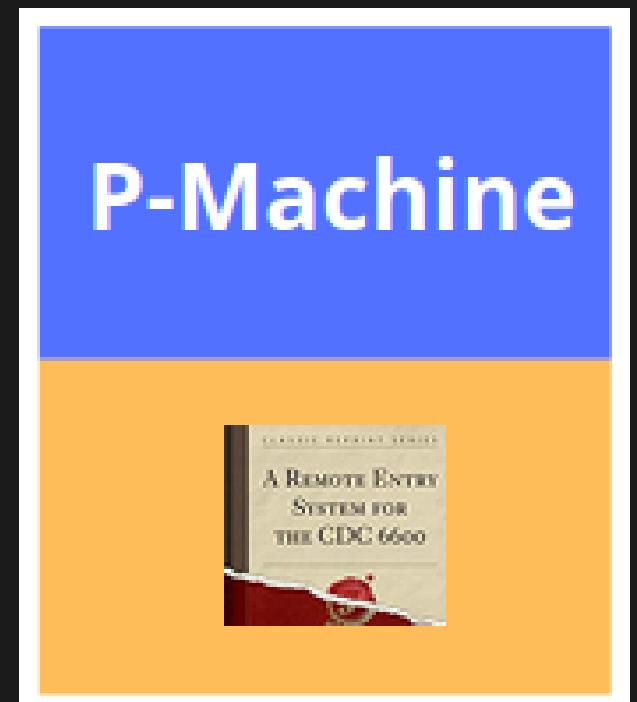
1.



2.



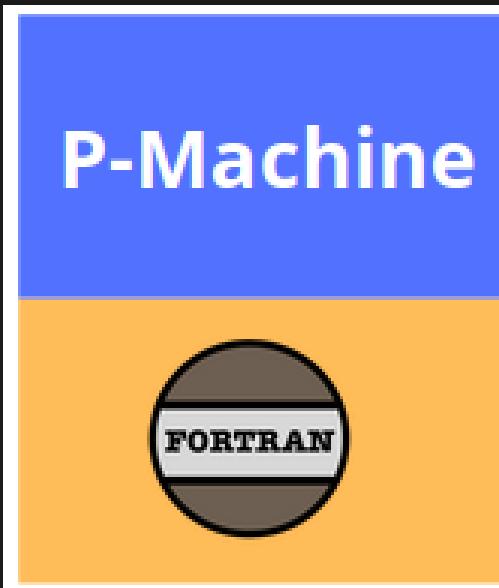
3.



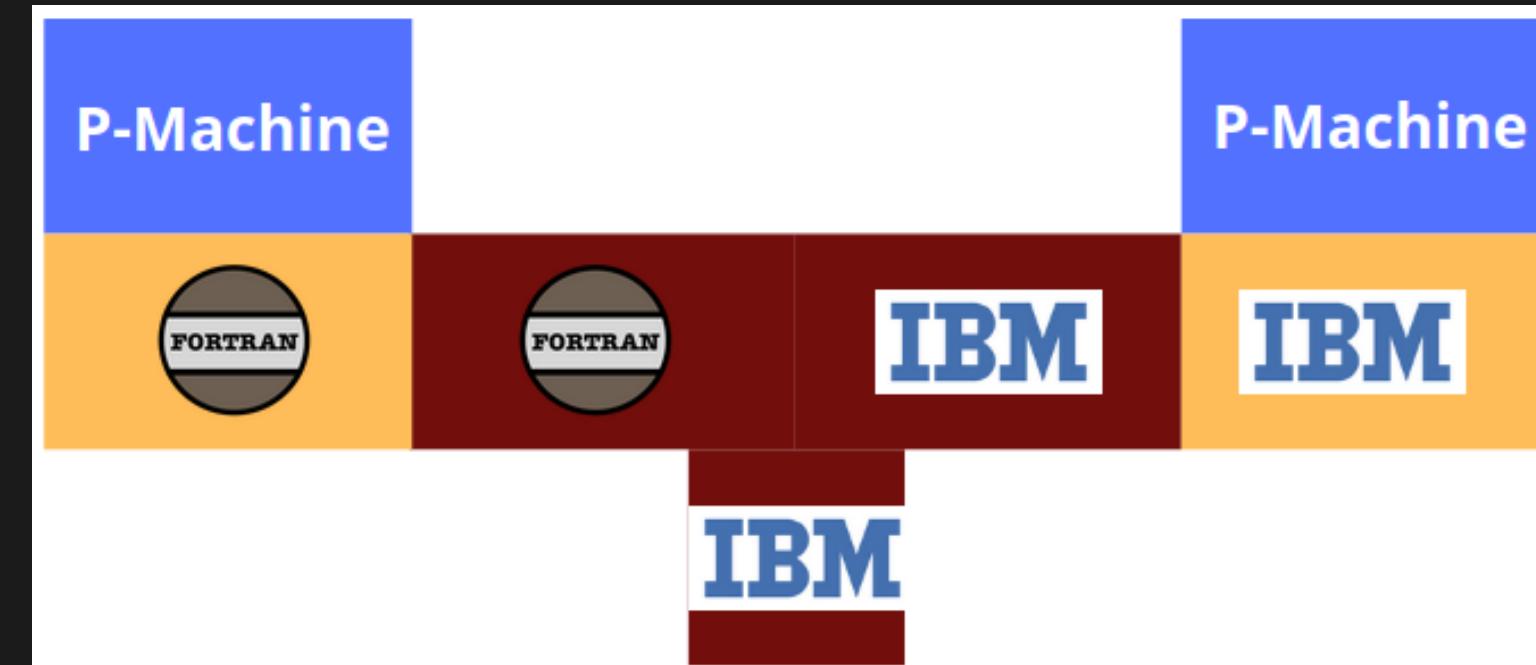
Muy bien, ya que se tiene todo esto, ¿qué prosigue? ¿cómo se usa?

Usando el Zurich Compiler Kit

1. Reescribir toda la máquina virtual en otro lenguaje que mi computadora sí pueda correr.
Como se mencionó antes, era relativamente fácil de reproducir. Supongamos que se escribe en Fortran.

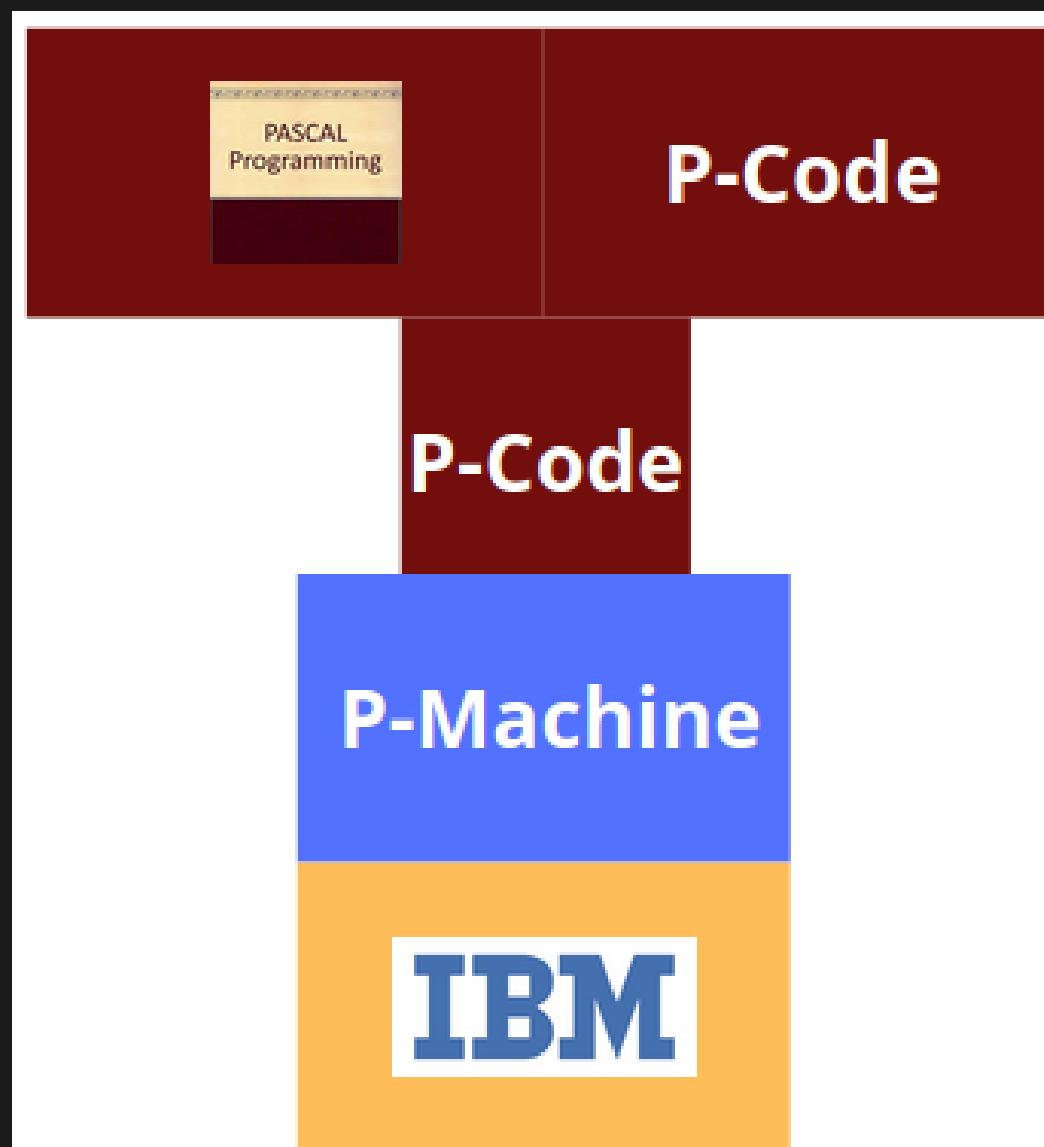


2. Compilar la máquina nueva en mi arquitectura.
Digamos que se hace con un compilador de Fortran a IBM-370.



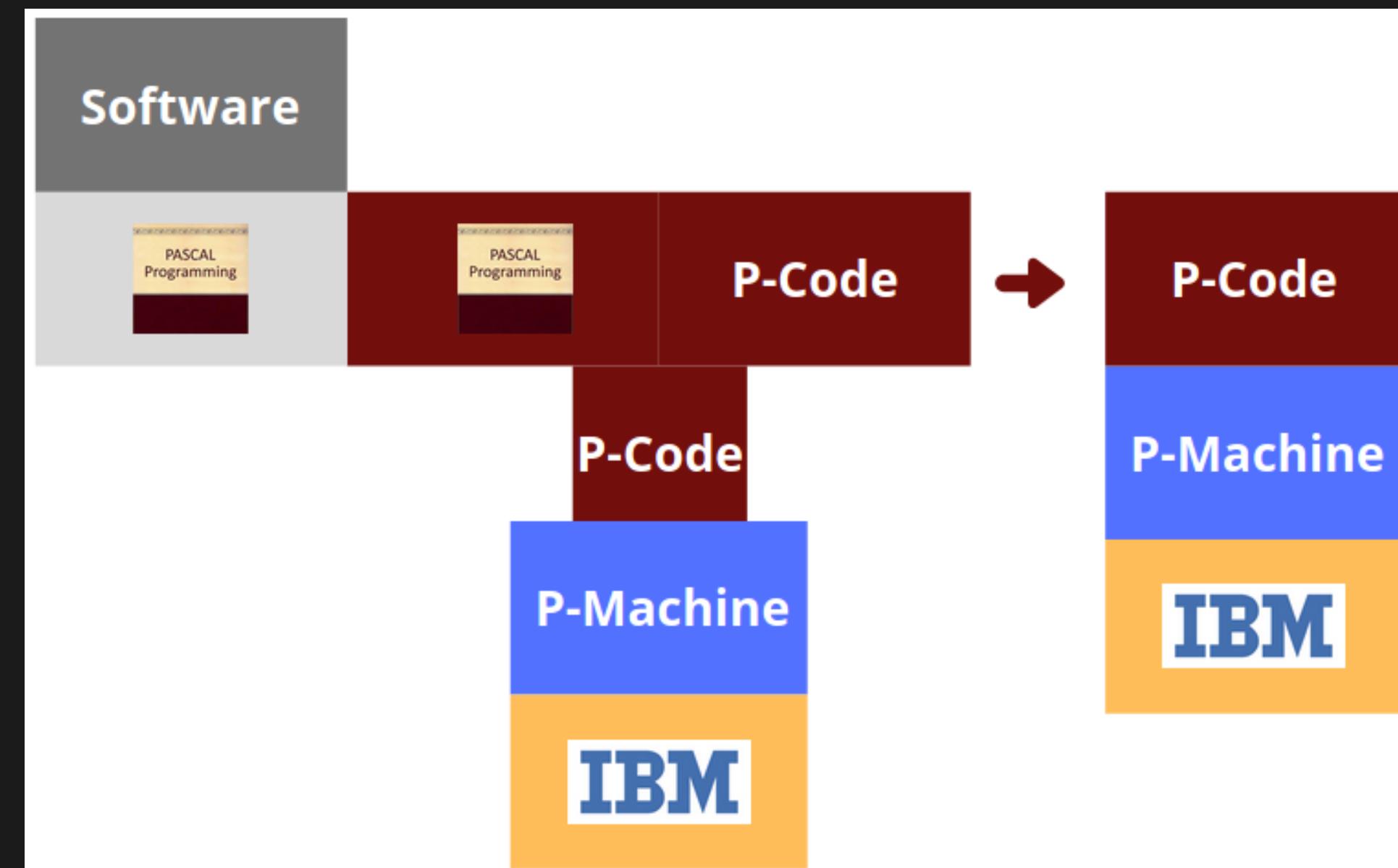
Usando el Zurich Compiler Kit

- Hasta ahora tenemos la máquina virtual ya corriendo en mi máquina. Lo que sigue es poner el compilador 2 que venía en el Kit sobre la P-Machine, y ya se puede compilar y ejecutar PASCAL, pero todo sobre la máquina virtual.



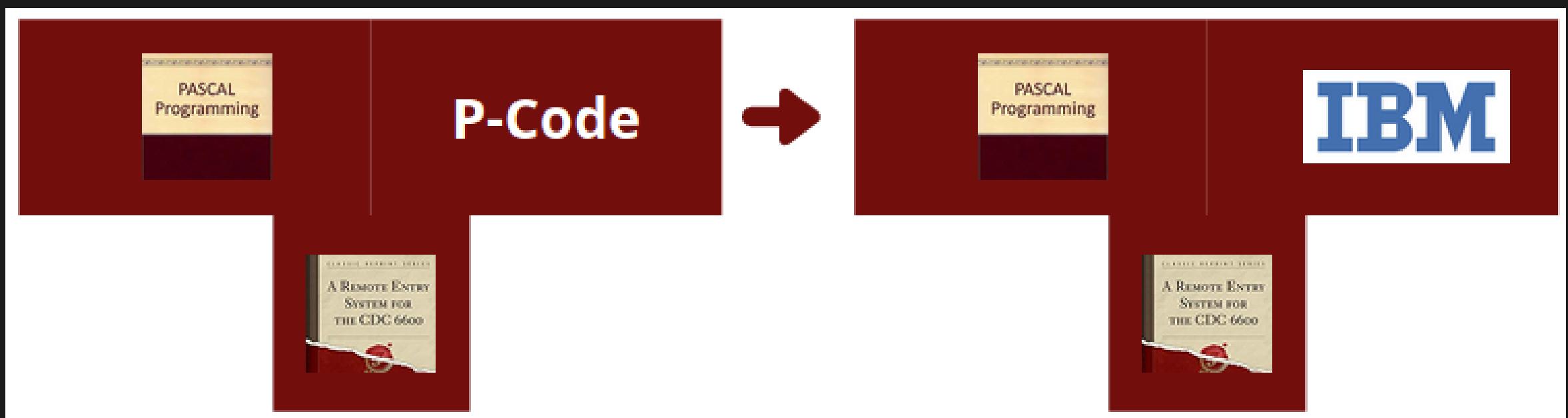
Usando el Zurich Compiler Kit

- Ejecutar PASCAL sobre la P-Machine. Cuando se compila el PASCAL con el compilador que corre sobre la P-Machine, se obtenía el equivalente en P-Code del programa original. Este P-Code podía ser ejecutado en la máquina virtual.



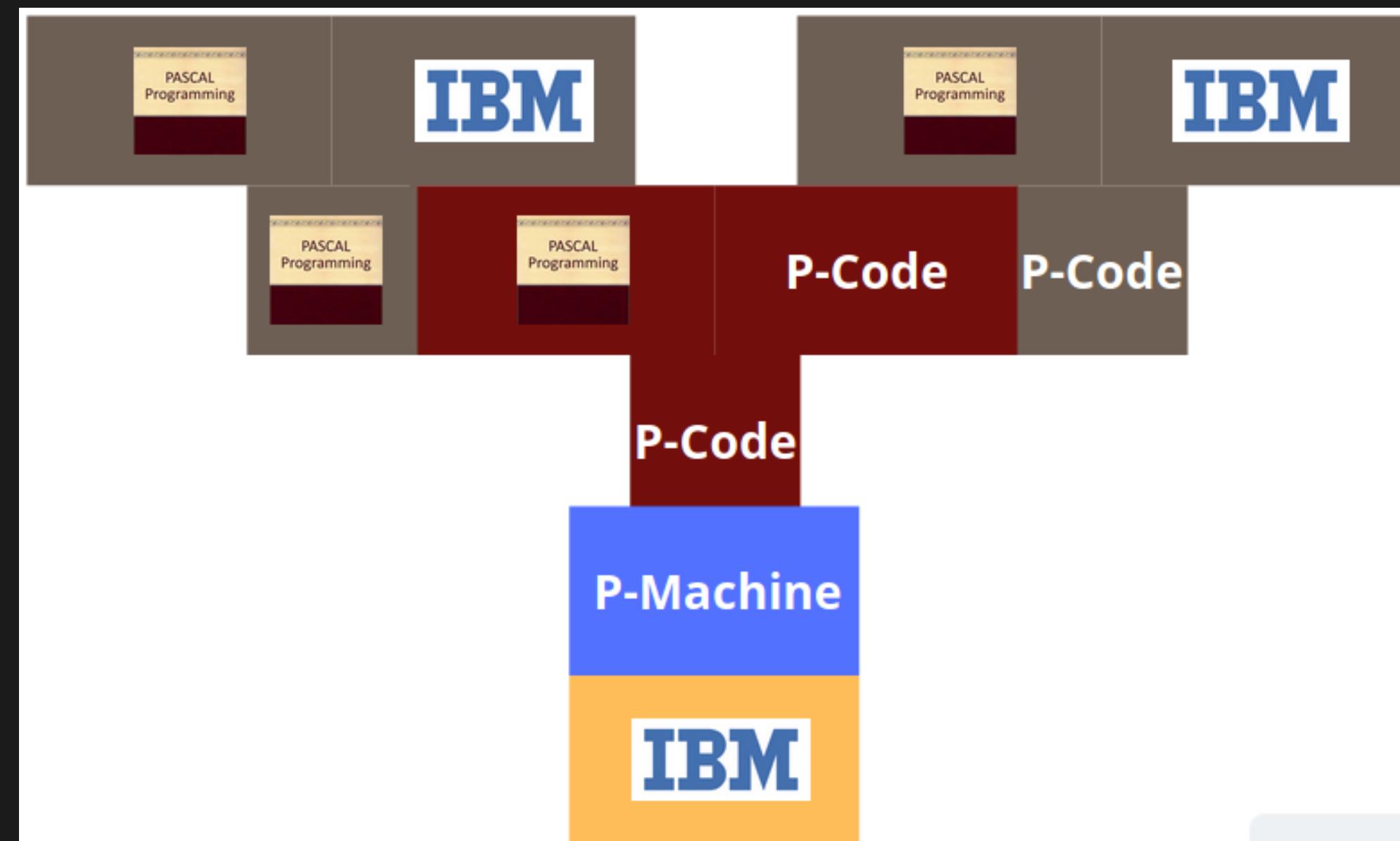
Usando el Zurich Compiler Kit

5. Estudiar el compilador de PASCAL que venía en el puesto 1 del kit, esto para poder modificar a mano su backend para que genere software de la arquitectura de mi máquina. Este proceso no era trivial, pero al estar escrito en PASCAL era bastante legible.



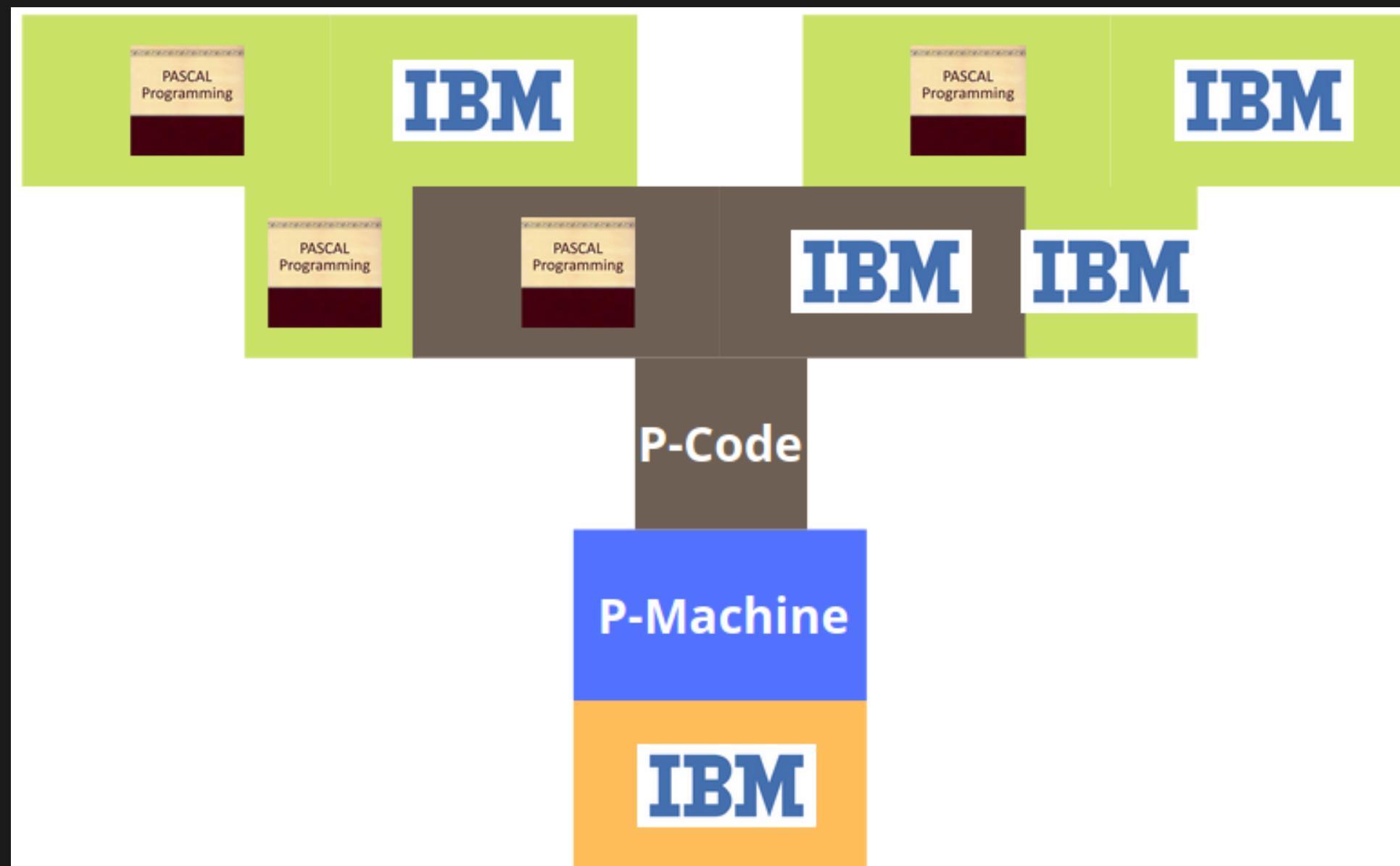
Usando el Zurich Compiler Kit

6. Se compila el compilador creado en el paso anterior.



Usando el Zurich Compiler Kit

7. Se compilan los compiladores que se ocupan para que se pueda correr PASCAL sin necesidad de la máquina virtual.



Este es el último paso. A partir de ahí, ya la máquina virtual y el P-Code no se necesitan para poder ejecutar PASCAL. Gracias a eso PASCAL ganó una gran popularidad.

Héroe: Niklaus Wirth

Gracias a sus trabajos en compiladores, ganó el **Turing Award** en **1984**.



Programas relacionados con compiladores

Ensambladores

Fueron inventados en los años 50. Como su código es traducido directamente a lenguaje máquina, se dice que tienen una relación 1 a 1. El código es muy eficiente, pero la sintaxis es muy simple.

Cada instrucción tiene una **etiqueta**, un **código de instrucción**, y **argumentos**.

Normalmente son de 2 pasadas:

La primera es para crear una tabla de símbolos: Se ven todas las etiquetas y variables, y llena la tabla con sus direcciones.

La segunda ya se ensambla con la tabla de símbolos.

¿Se puede hacer un compilador de una pasada?

Habría que usar la técnica de **backpatching**. Es decir, cuando se llegue a usar algo que todavía "no existe" (como una etiqueta que se declara más abajo), se deja como pendiente y cuando sepa la dirección que es se devuelve a arreglarlo.

```
file "hola.c"
.section .rodata
.LC0:
.string "%d"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $16, %rsp
movl $3, %edx
movl $2, %eax
addl %eax, %edx
;; movl $.LC0, %eax
:: movl %edx. %esi
```

Desensambladores

Son la inversa de los ensambladores, pasan de máquina a ensamblador.

Lo que hacen es ir leyendo los códigos de operación, variables, etiquetas, argumentos, y genera el código en ensamblador. Como en lenguaje máquina no hay nombres (como mi_variable_1), puede que los nombres autogenerados sean difíciles de asimilar.

Igual que los ensambladores, pueden ser de una o dos pasadas. Se pueden hacer bastante sofisticados para que reconozcan código repetido en el cual sólo cambien ciertos parámetros, para que así haga macros.

Un problema a resolver es saber cuándo termina el código y comienzan los datos.

Se les puede usar para ingeniería inversa.

```
004012A7  90      NOP
004012A8  $ 53    PUSH EBX
004012A9  . 56    PUSH ESI
004012AA  . 57    PUSH EDI
004012AB  . 8BF2    MOU ESI,EDX
004012AD  . 8BD8    MOU EBX,EAX
004012AF  . 85F6    TEST ESI,ESI
004012B1  . 8BFB    MOU EDI,EBX
004012B3  . 74 35    JE SHORT RTRACE.004012EA
004012B5  > 6A 04    PUSH 4
004012B7  . 68 00100000    PUSH 1000
004012BC  . 68 00100000    PUSH 1000
004012C1  . 53    PUSH EBX
004012C2  . E8 15870000    CALL <JMP.&KERNEL32.VirtualAlloc>
004012C7  . 85C0    TEST EAX,EAX
004012C9  . 75 0F    JNZ SHORT RTRACE.004012DA
004012CB  . 8BD3    MOU EDX,EBX
004012CD  . 8BC7    MOU EAX,EDI
004012CF  . 2BD7    SUB EDX,EDI
004012D1  . E8 1E000000    CALL RTRACE.004012F4
004012D6  . 33C0    XOR EAX,EAX
004012D8  . EB 15    JMP SHORT RTRACE.004012EF
004012DA  > 81C3 00100000    ADD EBX,1000
004012E0  . 81EE 00100000    SUB ES1,1000
004012E6  . 85F6    TEST ESI,ESI
004012E8  . 75 CB    JNZ SHORT RTRACE.004012B5
004012E9  > B8 01000000    MOU EAX,1
004012EF  . 5F    POP EDI
004012F0  . 5E    POP ESI
004012F1  . 5B    POP EBX
004012F2  . C3    RETN
004012F3  . 90    NOP
```

Protect = PAGE_READWRITE
AllocationType = MEM_COMMIT
Size = 1000 (4096.)
Address VirtualAlloc



Intérpretes

Combinan traducción con ejecución. Van línea por línea. Aplican muchas técnicas que son compartidas con los compiladores. Se les caracteriza por ser bastante independientes de la arquitectura.

Run-time environment: El intérprete crea todo un ambiente disponible en el momento que se está corriendo el código.

Son lentos en ejecución porque todo se repite una y otra vez, pero tiene sus virtudes.

Ejemplos:

- Lisp
- Basic
- Python
- Ruby (popular entre los de electrónica pero no saben programar)
- Forth (antes era popular para manejar punteros y listas enlazadas)



Preprocesadores

Programa que se corre antes de que se compile el código. En el caso del preprocesador de C, cuando se encuentra una instrucción como `#include`, `#define` o `#if`, genera un fuente de C con esas instrucciones ya evaluadas y hasta después lo compila.

`#include` — Incluir otro archivo en el fuente en el que estoy

`#define` — Macros

compilación condicional — Quitar pedazos según el ambiente de código

Un buen uso del preprocesador hace que se siga un buen estilo de código. Además usa técnicas de compilación, es casi un lenguaje.

Los preprocesadores son independientes del lenguaje. Esto quiere decir que, eventualmente, el preprocesador de C me podría servir para pre-procesar código escrito en otro lenguaje de programación.

C Preprocessor Directives

- ① Macros → `#define NAME<space>VALUE`
- ② File Inclusion → `#include "filename1.h"`
`#include <filename2.h>`
- ③ Conditional Compilation →
`#ifdef`

`#elif`

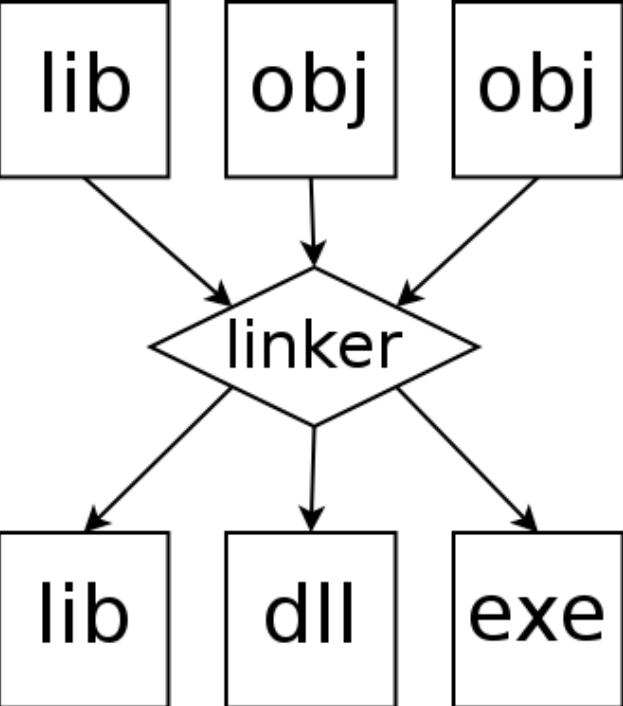
`#else`

`#endif`

Linkers

Se les llama ligadores o eslabonadores. Cuando se ensambla o se compila, se genera un código objeto, pero es posible que hayan varios fuentes por aparte. El linker es el que se encarga de ir a buscar los módulos objeto y unirlos para que funcionen, ya que al final sólo se puede tener un archivo ejecutable. Muchas veces ya viene integrado con el compilador.

Un conjunto de módulos objeto se llama **biblioteca, no librería**. La biblioteca presta libros, la librería los vende.



Estático

Todo lo que se necesite (y algunas cosas que tal vez no se van a necesitar) se unen al ejecutable. Es más rápido, pero no admite cambios en las bibliotecas individualmente (hay que recompilar todo), y el tamaño del archivo es muy grande.

Dinámico

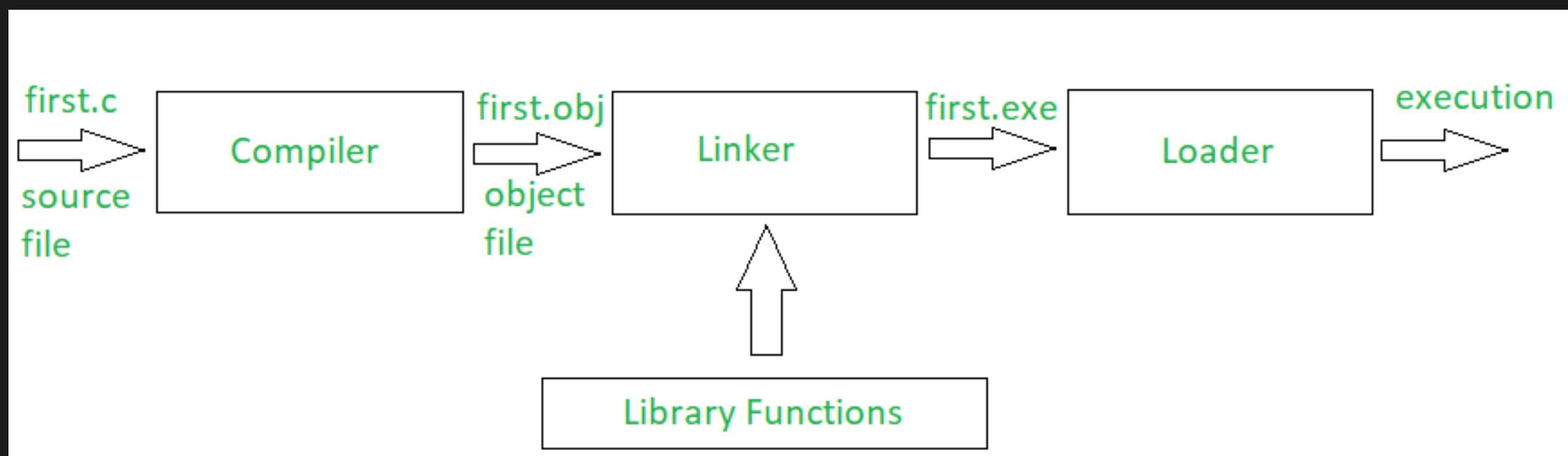
Cuando se necesite algo, lo va a buscar (en Windows, busca los archivos DLL). Se pueden hacer cambios en las bibliotecas, pero esto puede ser un problema de seguridad (por ejemplo, para poder jugar gratis se modifican las bibliotecas). Más ligero pero más lento.

Loader

Cuando se corre el programa, este tiene que ser leído y arreglar todo para que funcione el memoria. El loader lo carga en memoria, puede ser tan sencillo como leerlo y cargarlo directo o se puede necesitar relocalizar (cambiar las direcciones de memoria para que se ajusten a las nuevas direcciones).

Es parte del Sistema Operativo. Se encarga de negociar las direcciones nuevas y de generar todas las estructuras de datos que se necesiten para que el programa sea puesto a correr.

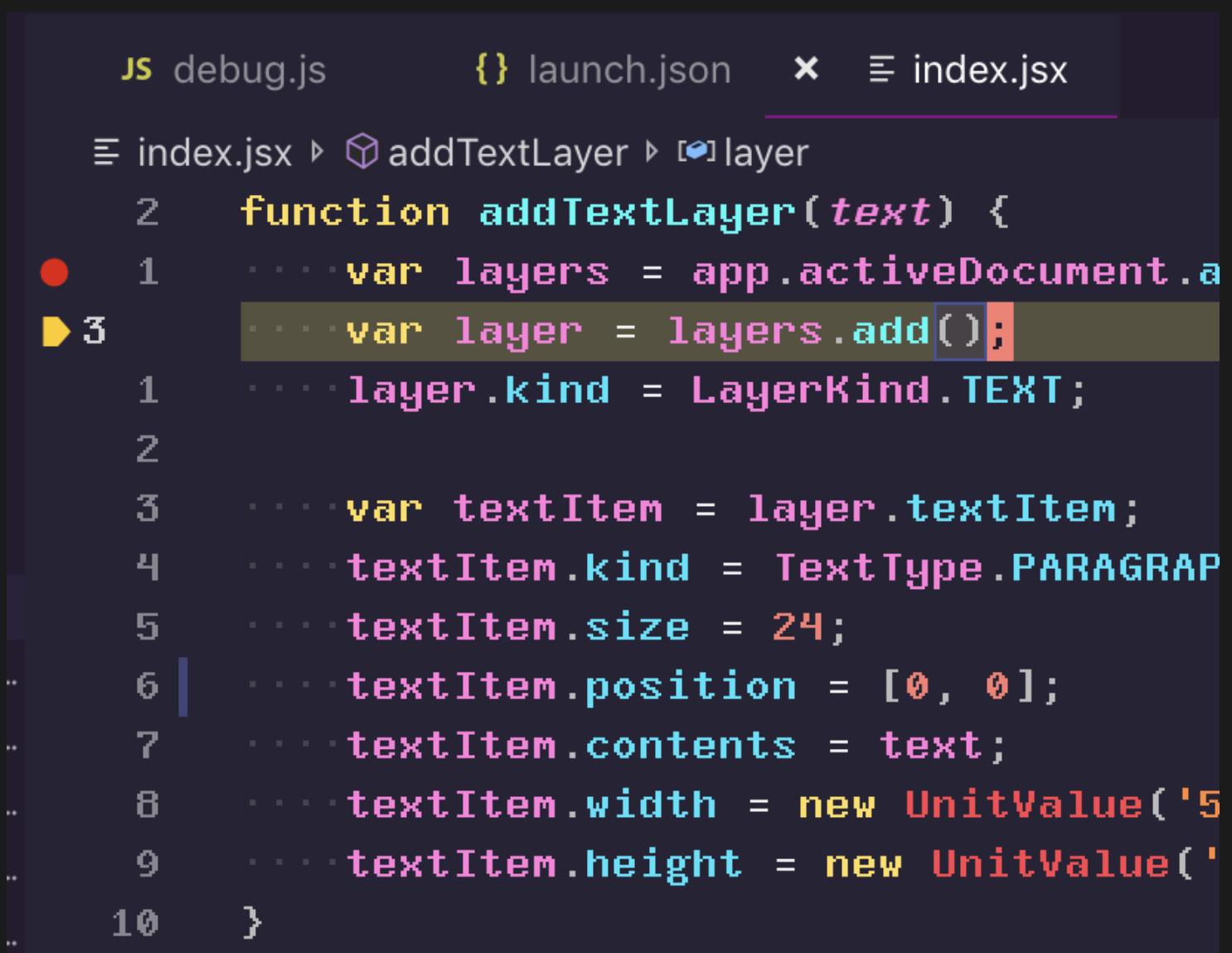
Una estructura muy importante es el **PCB** (process control block). Básicamente almacena toda la información que necesita el SO para administrar un proceso. El Loader es el responsable de crearlo.



Debugger

Es una herramienta para cobardes que permite la ejecución controlada de otro programa. Se puede ir línea por línea, poner breakpoints (que tienen que ver más que todo con hardware), ver estado del flujo de ejecución y las variables.

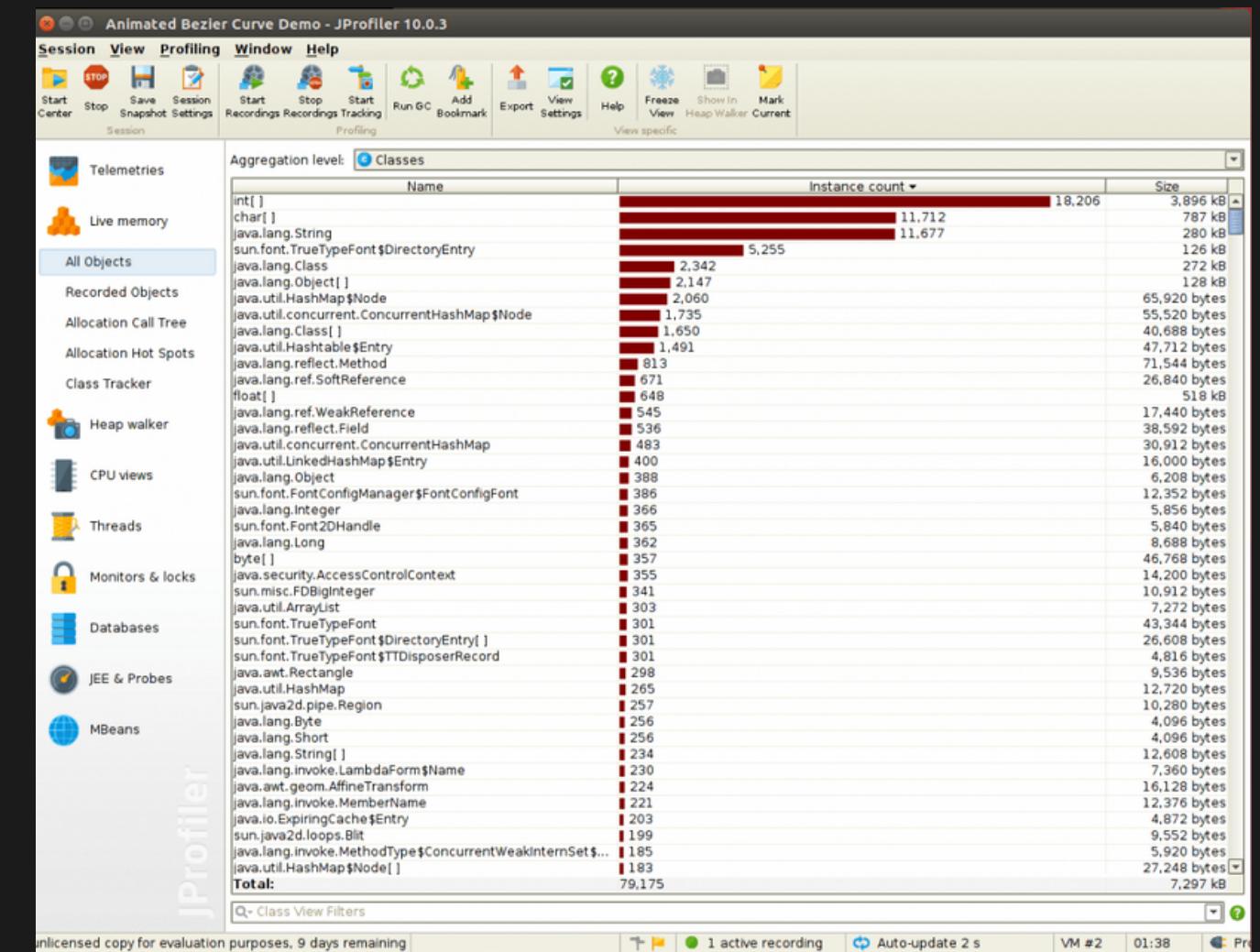
El compilador, en caso de que se vaya a usar un debugger, deja un montón de información, como la tabla de símbolos, nombres de variables, entre otros. Por ese motivo puede que el ejecutable quede muy grande. Para evitar eso, se podría investigar si hay opciones (como activar o desactivar flags) para controlar la información que se deja.



```
JS debug.js      {} launch.json  ✖  index.jsx
Ξ index.jsx ▶ ⚡ addTextLayer ▶ ⚡ layer
  2   function addTextLayer(text) {
●  1     ... var layers = app.activeDocument.a
▶ 3     ... var layer = layers.add();
  1     ... layer.kind = LayerKind.TEXT;
  2
  3     ... var textItem = layer.textItem;
  4     ... textItem.kind = TextType.PARAGRAPH;
  5     ... textItem.size = 24;
  6     ... textItem.position = [0, 0];
  7     ... textItem.contents = text;
  8     ... textItem.width = new UnitValue('5
  9     ... textItem.height = new UnitValue('
10   }
```

Profilers

- Manda a correr otro programa pero recoge un montón de estadísticas de la ejecución. Por ejemplo, cuántas veces se llamó la instrucción, duración promedio de llamadas, y el porcentaje de tiempo que se le dedica a una región del programa.
- Sirve para optimización, dependiendo de los resultados uno podría decidir reescribir funciones en ensamblador o mejorar el diseño en general.
- Como los debuggers, necesita información creada por el compilador, siempre necesita saber en qué parte del código está para poder dar información precisa.
- Podría dejarle información al compilador, puede que se aplique técnicas de optimización que son caras en tiempo de compilación pero que pueden ser útiles en zonas calientes. Se podría implementar también la recompilación automática con los datos del profiler.



Notas de optimización para todos los días:

1. Antes de dejar el producto final, quite todos los prints porque son muy caros.
2. No ponga un if dentro de un ciclo si se va a ejecutar muy pocas veces.

Compiladores e Intérpretes

Apuntes

Miércoles 23 de Setiembre

Edgar Pineda Aguilar

Quiz 02

1. Defina detalladamente los siguientes conceptos:

- A. Preprocesador
- B. Linking Dinamico
- C. Relocalización
- D. Back-patching
- E. Front End

2. Suponga que su sistema Linux tiene un compilador de C, versión 7.

Ud recibe por correo los siguientes archivos:

- A. El programa fuente de la version 10 del compilador de C, escrito en C
- B. El fuente de un compilador de Fortran que genera C, escrito en C.
- C. Un compilador de Cobol, escrito en Fortran, que genera lenguaje máquina.

Usando diagramas T y explicaciones detalladas, diga lo mejor que se puede hacer con esto.

En Capítulos Anteriores...

Preprocesadores

- Ocurre antes de la compilación
- Carga archivos que se vayan a ocupar y elimina comentarios
- Independientes del lenguaje

Debugger

- Software que corre de forma controlada otro programa
- Instrucción por instrucción
- Ver contenido de variables

Linker

- Bibliotecas de funciones
- Crea un modulo ejecutable
- Existen dinámicos y estáticos

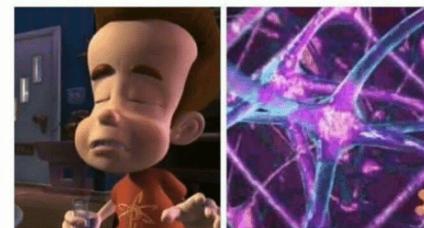
Profiler

- Recolecta estadísticas de la ejecución
- Usa info creada por el compilador
- Nos permite saber donde optimizar código

Loader

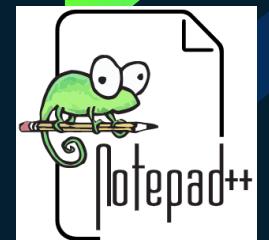
- Carga un modulo ejecutable en memoria
- Relocaliza y negocia espacio en memoria para poder ejecutarse
- Crea estructuras de control

Yo, en el Quiz , tratando de recordar un tema que vi 5min antes de entrar...



Editor de Texto

- Los programas fuente se escriben con editores de texto
- Por dentro utilizan técnicas prestadas de compiladores
- Usan expresiones regulares para búsquedas
- Algunos editores de texto populares son:
 - Emacs (Richard Stallman)
 - Vim
 - Sublime Text
- Brackets
- Visual Studio Code
- Notepad++



Editor de Texto - Cont

- Editores de Texto de un Lenguaje en específico:
 - Tiene funciones especiales para el lenguaje
 - Conoce la sintaxis del lenguaje
 - Marca con colores palabras reservadas
 - Genera código
 - Detecta errores sintácticos
 - Hace una combinación del editor, compilador, profiler y debugger, generando así un IDE.

IDE
=

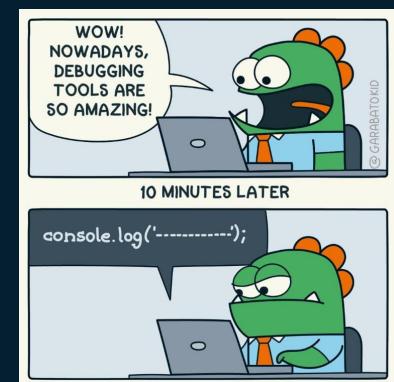
Integrated Development Environment



Netbeans: Error en la linea 42

Yo: Cuál linea 42? mi código tiene 30 líneas

Netbeans:

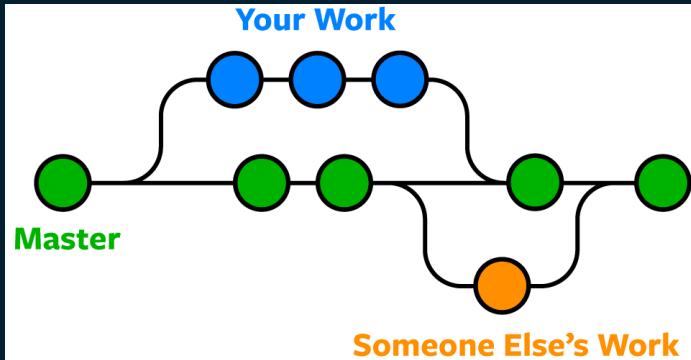


Manejador de Versiones

- Orientado a proyectos con mucha gente
- Si alguien se equivoca, puede devolverse
- Trabajar en Branches
- Queda documentado quien hizo cada parte y lo que hizo
- Puede ser independiente del lenguaje
- Ejemplo: Git, creado por Linus Torvalds



git



Linus Torvalds es el creador de Linux

Cambio de Fecha de Entrega de Proyecto

- Se pasa para el 09 de Octubre, por problemas técnicos con el pdf del capítulo 2 del libro para construir un Compilador
- Se subió la versión que si era actualizada al Foro, utilizar la versión que subió el compañero con usuario sacastro
- Los comentarios en el proyecto tienen que empezar con --
- Seguir lo que diga el libro
- Se pueden hacer asignaciones aritméticas en el compilador

Estructura Básica

Componentes Principales de un
Compilador

Entrada y Salida

- Entrada: Lenguaje Fuente
- Salida: Lenguaje Objeto
- Del paso del Fuente al Objeto pasa por el compilador y hay una representación intermedia.

The diagram illustrates the process of compilation. It starts with a yellow box labeled "Fuente" containing a snippet of C-like source code. An orange arrow points to a blue rounded rectangle labeled "Compilador". Inside the "Compilador" box is a yellow trapezoid labeled "Representación Intermedia". Another orange arrow points to a yellow box labeled "Objeto" containing assembly language code. The background features abstract geometric shapes in green, blue, and red.

```
/* This program adds two numbers and prints the answer. */
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    // Prompt user for first number
    int number1 = getint("Please enter a number: ");
    // Prompt user for second number
    int number2 = getint("Please enter a second number: ");
    // Add numbers
    int total = number1 + number2;
    // Print answer to screen
    printf("The sum of the two numbers is: %i\n", total);
}
```

Fuente

Representación
Intermedia

```
00000000 push    ebp
00000001 mov     ebp, esp
00000002 movzx   ecx, [ebp+arg_0]
00000003 pop    ebp
00000004 movzx   dx, cl
00000005 lea    eax, [edx*edx]
00000006 add    eax, edx
00000007 shl    eax, 2
00000008 add    eax, edx
00000009 shr    eax, 8
0000000A sub    cl, al
0000000B shr    cl, 1
0000000C add    al, cl
0000000D shr    al, 5
0000000E movzx   eax, al
0000000F retn
```

Objeto

Representación Intermedia

- Como sabemos existen 2 fases:
 - Front End: Desde el fuente a una representación intermedia. Sabe como leer el lenguaje fuente
 - Back End: Desde la representación intermedia al lenguaje destino. Sabe como convertir a lenguaje destino



Objeto

Análisis

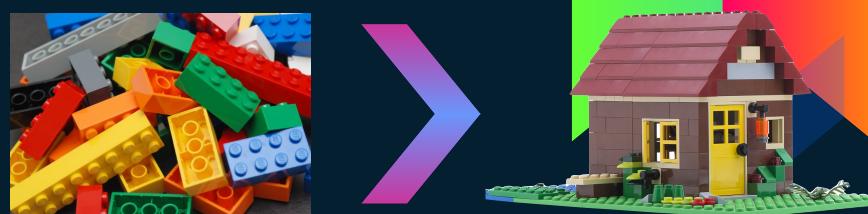
- › Descomponer en partes mas pequeñas un problema para así entender el proceso grande
- › Nunca rompimos un juguete, lo **analizábamos**
- › Front End -> Análisis



>> Entre más independientes sean estas fases, hay mas flexibilidad

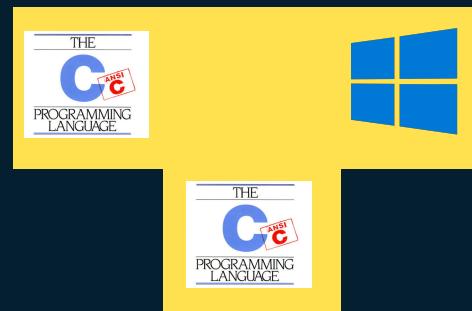
Síntesis

- › Fabricar algo a partir de varias piezas sueltas
- › Cosas sintéticas, que fueron fabricadas
- › Back End -> Síntesis



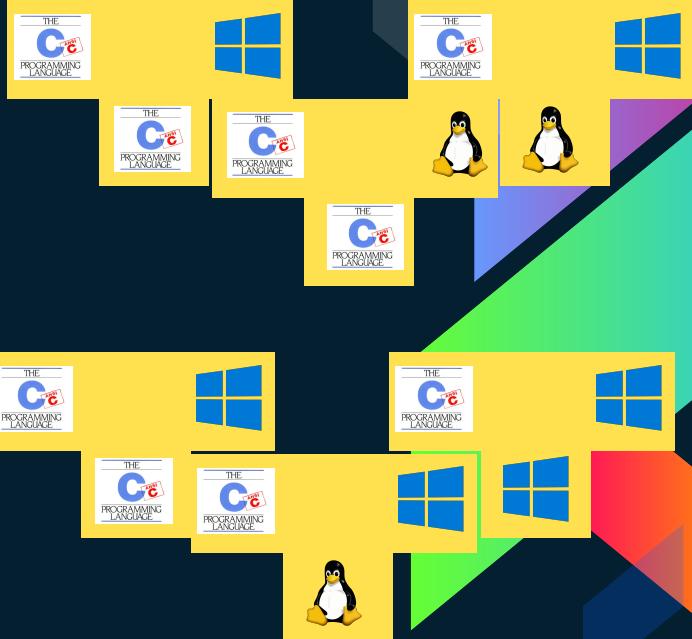
Ejemplo

- Se tiene un compilador de C, escrito en C que genera Linux y se quiere hacer un compilador de C, escrito en C que genere Windows.
- Para hacer que un compilador cambie de lenguaje objetivo, hay que modificar el backend.
- De x86Linux a x86Windows el coding ensamblador es el mismo, lo que cambian son los **system calls**



Ejemplo - Cont

- Después de modificar el back end se compila y se convierte en un compilador de C, escrito en x86Linux que genera x86Windows (Cross Compiler)
- El compilador nuevo se usa para compilar otra vez el compilador de C, escrito en C que genera x86windows
- Ahora nuestro nuevo compilador ya puedo correrlo en windows y ahora tengo gcc corriendo en windows.



Estructura Detallada

Fases de Compilación

**Disclaimer: Todo lo que se va a hablar de estas fases
son mentiras, con algunas verdades**

Pasadas

- Una pasada es una lectura completa del fuente de inicio a fin
- Los traductores(compiladores o interpretes) requieren 1 o mas pasadas
- Los traductores hacen "N" pasadas porque aveces algunas cosas están definidas fisicamente después de haberlas usado



Descripción gráfica del compilador haciendo N pasadas

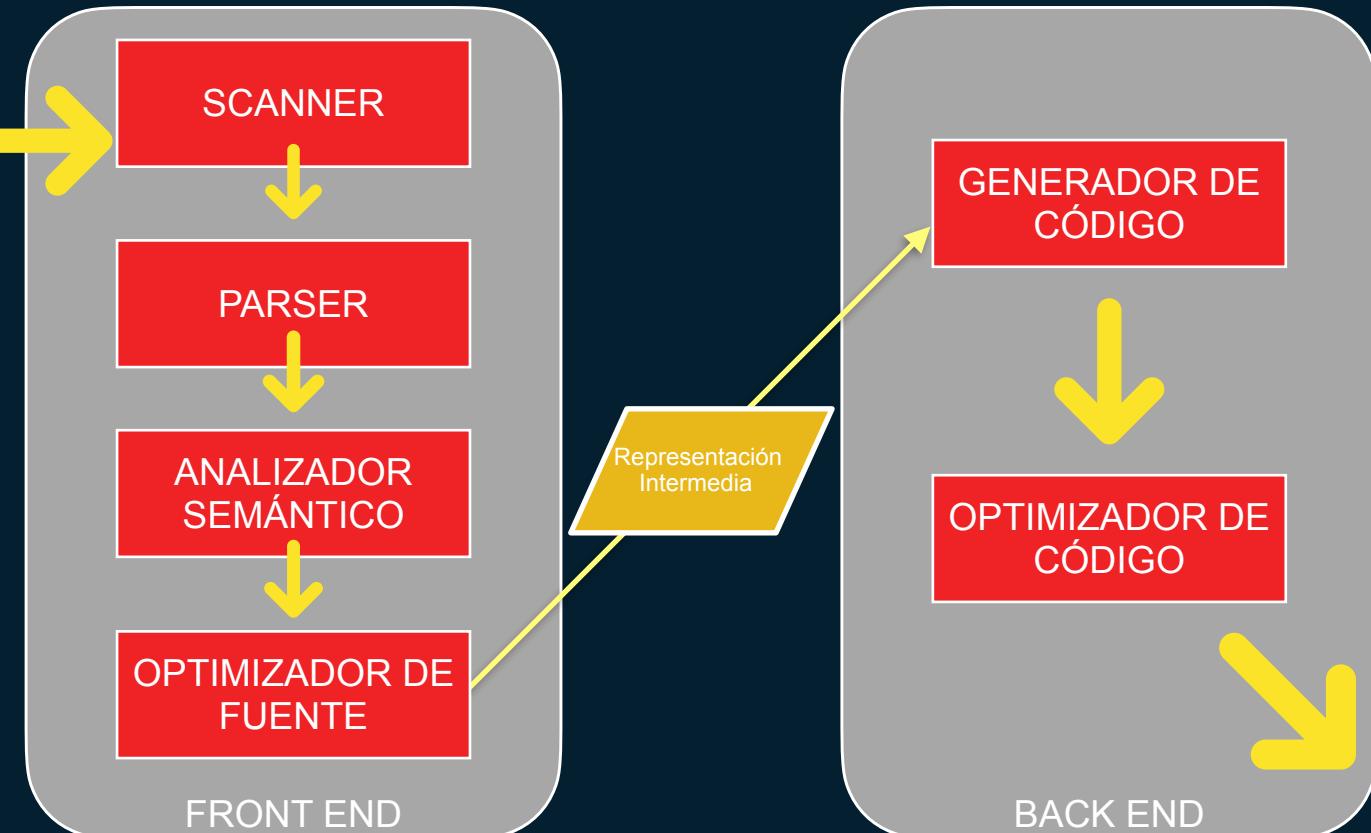
- Más Pasadas =>
- Más Flexibilidad =>
- Mayor Tiempo Compilación*

Si solo hace una pasada hace Backpatching

- *El tiempo de compilación ya no afecta tanto porque los compiladores son muy eficientes (complejidad lineal $O(n)$) y aparte las computadoras son más rápidas.
- Big Win para la computación!

```
1; this program has no name and does whatever it wants.
2; because it's cool.
3; and it's fun.
4;
5; double main = 42;
6; double main = 42;
7; double main = 42;
8; double main = 42;
9; double main = 42;
10; double main = 42;
11; double main = 42;
12; double main = 42;
13; double main = 42;
14; double main = 42;
15; double main = 42;
16; double main = 42;
17; double main = 42;
18; double main = 42;
19; double main = 42;
20; double main = 42;
21; double main = 42;
22; double main = 42;
23; double main = 42;
24; double main = 42;
25; double main = 42;
26; double main = 42;
27; double main = 42;
28; double main = 42;
29; double main = 42;
30; double main = 42;
31; double main = 42;
32; double main = 42;
33; double main = 42;
34; double main = 42;
35; double main = 42;
36; double main = 42;
37; double main = 42;
38; double main = 42;
39; double main = 42;
40; double main = 42;
41; double main = 42;
42; double main = 42;
43; double main = 42;
44; double main = 42;
45; double main = 42;
46; double main = 42;
47; double main = 42;
48; double main = 42;
49; double main = 42;
50; double main = 42;
51; double main = 42;
52; double main = 42;
53; double main = 42;
54; double main = 42;
55; double main = 42;
56; double main = 42;
57; double main = 42;
58; double main = 42;
59; double main = 42;
60; double main = 42;
61; double main = 42;
62; double main = 42;
63; double main = 42;
64; double main = 42;
65; double main = 42;
66; double main = 42;
67; double main = 42;
68; double main = 42;
69; double main = 42;
70; double main = 42;
71; double main = 42;
72; double main = 42;
73; double main = 42;
74; double main = 42;
75; double main = 42;
76; double main = 42;
77; double main = 42;
78; double main = 42;
79; double main = 42;
80; double main = 42;
81; double main = 42;
82; double main = 42;
83; double main = 42;
84; double main = 42;
85; double main = 42;
86; double main = 42;
87; double main = 42;
88; double main = 42;
89; double main = 42;
90; double main = 42;
91; double main = 42;
92; double main = 42;
93; double main = 42;
94; double main = 42;
95; double main = 42;
96; double main = 42;
97; double main = 42;
98; double main = 42;
99; double main = 42;
100; double main = 42;
```

Fuente



Fases de Compilación

A gran escala

```
00000000 push ebp
00000000 mov esp,ebp
00000000 movzx eax,[ebp+arg_0]
00000000 movzx dx,cl
00000000 lea rdx,[eax+edx]
00000000 add eax,rdx
00000000 sub eax,al
00000000 shr cl,1
00000000 and al,cl
00000000 shr al,5
00000000 movzx eax,al
00000000 retb
```

Scanner o Análisis Léxico

- Análisis **Léxico** : Descomponer en palabras el **vocabulario**
- Agarra el fuente, lo divide en palabras y genera los **tokens**
- Se implementa con autómatas de estado finito
- Sabe diferenciar palabras entre ellas

: Separadores

: Palabras Reservadas

: Operadores

: Literales

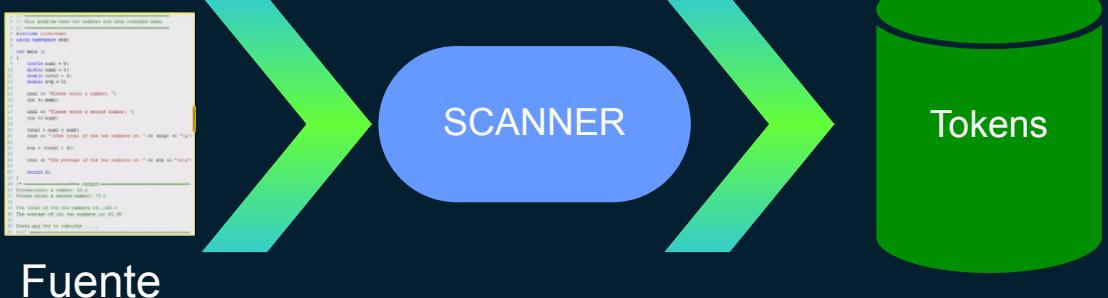
: Identificadores

```
void Burbuja (n_entradas, vector<int>) {
    int indice, j, aux;
    for (indice = 0; indice < n_entradas; indice++){
        for (j = indice + 1; j < n; j++){
            if (vector[j] < vector[indice]) {
                aux = vector[j];
                vector[j] = vector[indice];
                vector[indice] = aux;
            }
        }
    }
}
```

Ejemplo de como trabaja el Scanner

Entrada y Salida de Análisis Léxico

- Se podría decir que el Scanner es un chunche mágico que convierte el fuente en un montón de tokens



Acordemonos que esto son mentiras



Parser o Analizador Sintáctico

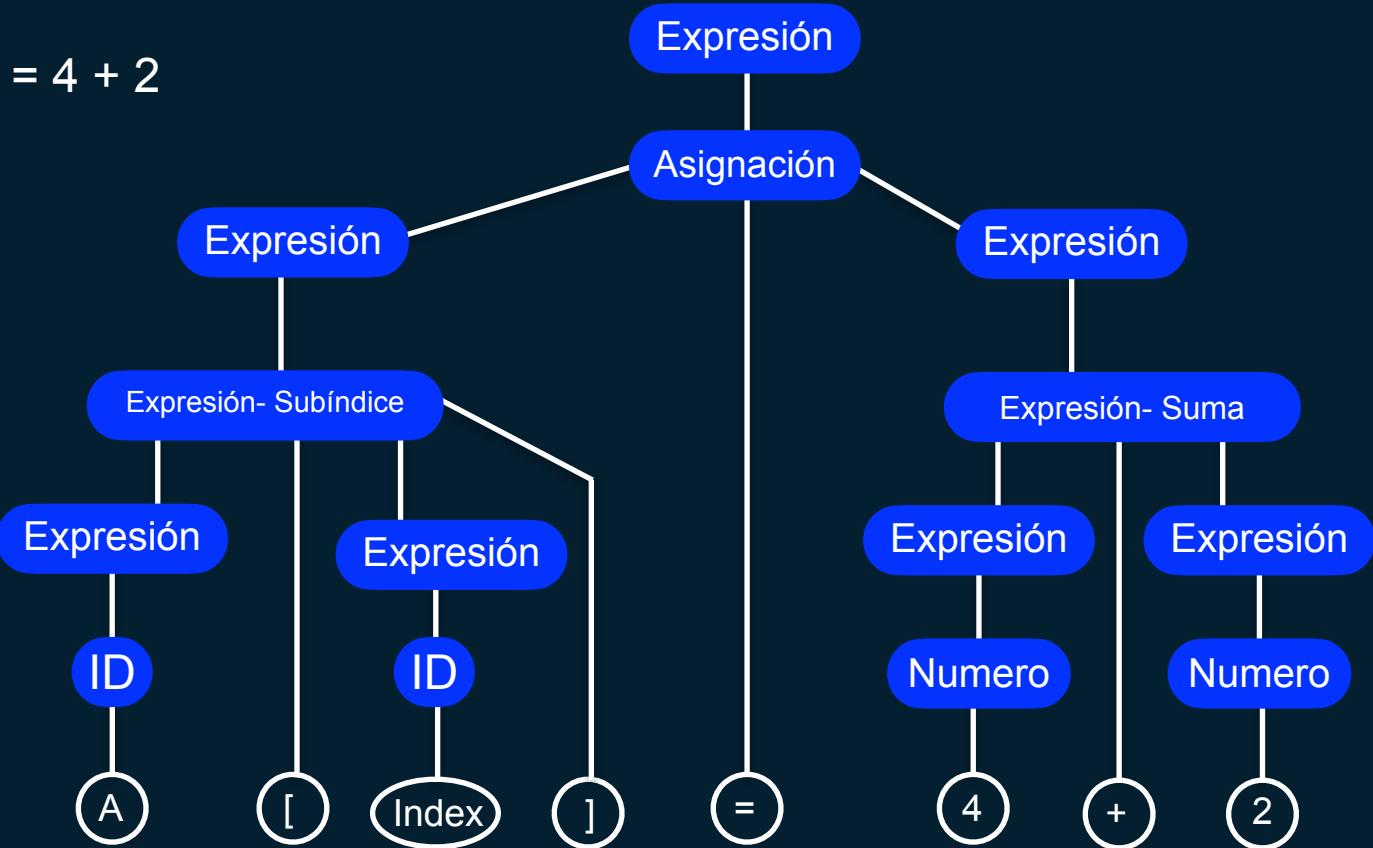
- Va después del Scanner
- Revisa que la Sintaxis del programa sea correcta
- Agarra todos los tokens y verifica que tengan sentido -> **Gramática**
- Hace uso de los "PushDown Automata"
- Hay herramientas para generar un parser
- Micro usa "Descenso Recursivo"
- El parser genera un árbol con los tokens que creó el scanner para ver si cumple la Gramatica.

To Parse:
Entender la estructura de una frase

PushDown Automata:
Es un automata que reconoce si una "cadena" es parte de una lenguaje dado un alfabeto

Parser o Analizador Sintáctico - Ejemplo

A [index] = 4 + 2



Entrada y Salida de Análisis Sintáctico

- Se podría decir que un Scanner es un chunche mágico que le entran todos los tokens que genera el scanner y genera el árbol sintáctico.



- Si hubo un error sintáctico entonces no generaría el árbol sino que tiraría error.



Análisis Semántica

- La **semántica** de algo es el **significado**
- El sentido de las instrucciones
- Hay que ver si en tiempo de corrida si lo que esta escrito está bien o está mal -> **Semántica Estática**
- Revisa las declaraciones, revisa tipos, si las variables están declaradas
- Si se hacen revoltijos de tipos, esto seria un error semántico, usar un arreglo como un int
- Construye la **Tabla de Símbolos** -> **Árbol de Atributos o Árbol Anotado**

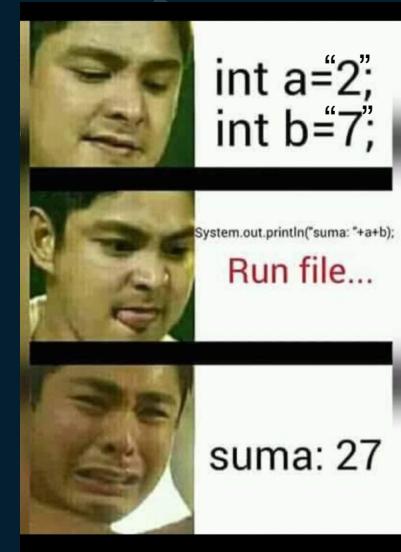
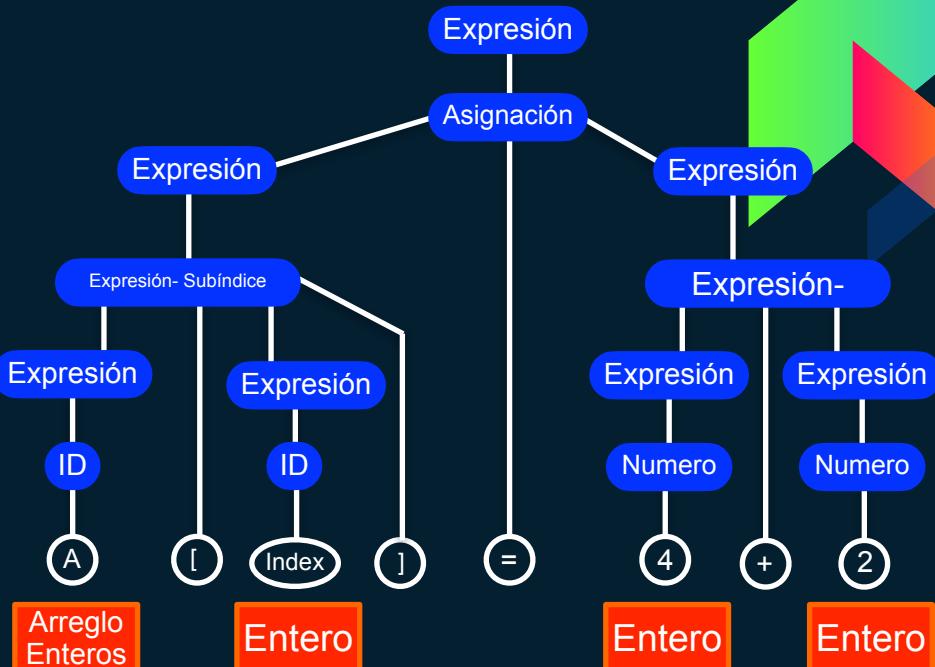


Tabla de Símbolos:
Asocia cada símbolo con los atributos.
Ej : variables locales

Analizador Semántico - Ejemplo

- Que sea sintácticamente correcto no significa que sea semánticamente correcto
- Revisa que las variables coincidan con su declaración
- Revisa que el código tenga sentido
- Algo puede ser sintácticamente correcto pero sin sentido.



Compilador cuando ve un error semántico

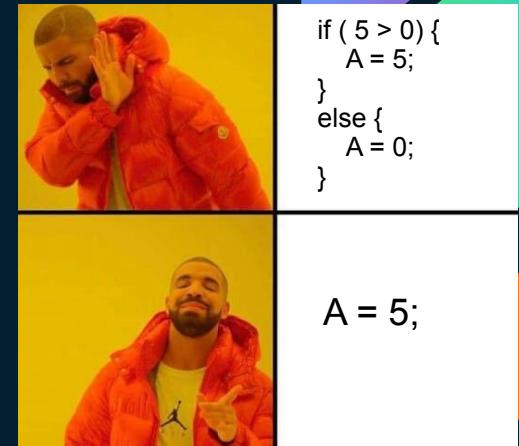
Entrada y Salida de Análisis Semántico

- Se podría decir que un Scanner es un chunche mágico que le entra un árbol sintáctico y genera un árbol decorado y una tabla de símbolos



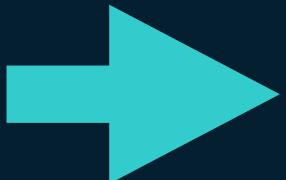
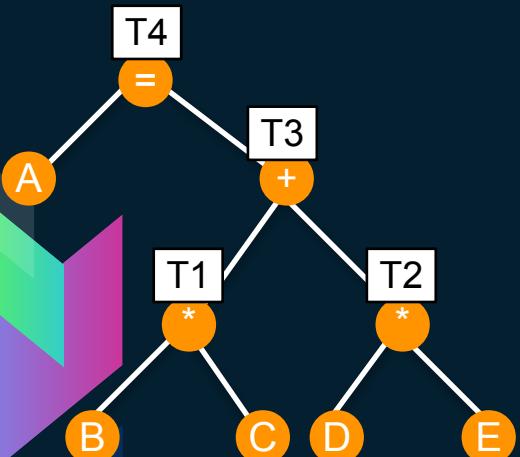
Optimizador de Código Fuente

- El compilador hace el cálculo de constantes, para que llegue un código más limpio a la representación intermedia y genere un código más compacto.
- Ejemplos:
 - $A = 10 + 7 \Rightarrow A = 17$
 - $B = 0 * 3 \Rightarrow B = 0$
 - $C = 0 + 5 \Rightarrow C = 5$
- También si ve que hay código que nunca se ejecuta lo omite
- Esto lo genera a partir del Árbol Anotado
- Agarra lo que está en el lenguaje y lo va convirtiendo en un ensamblador de 3 direcciones -> **Código intermedio**



Código Intermedio / Representación Intermedia - Ejemplo

$$A = B * C + D * E$$



$$T1 = B * C$$

$$T2 = D * E$$

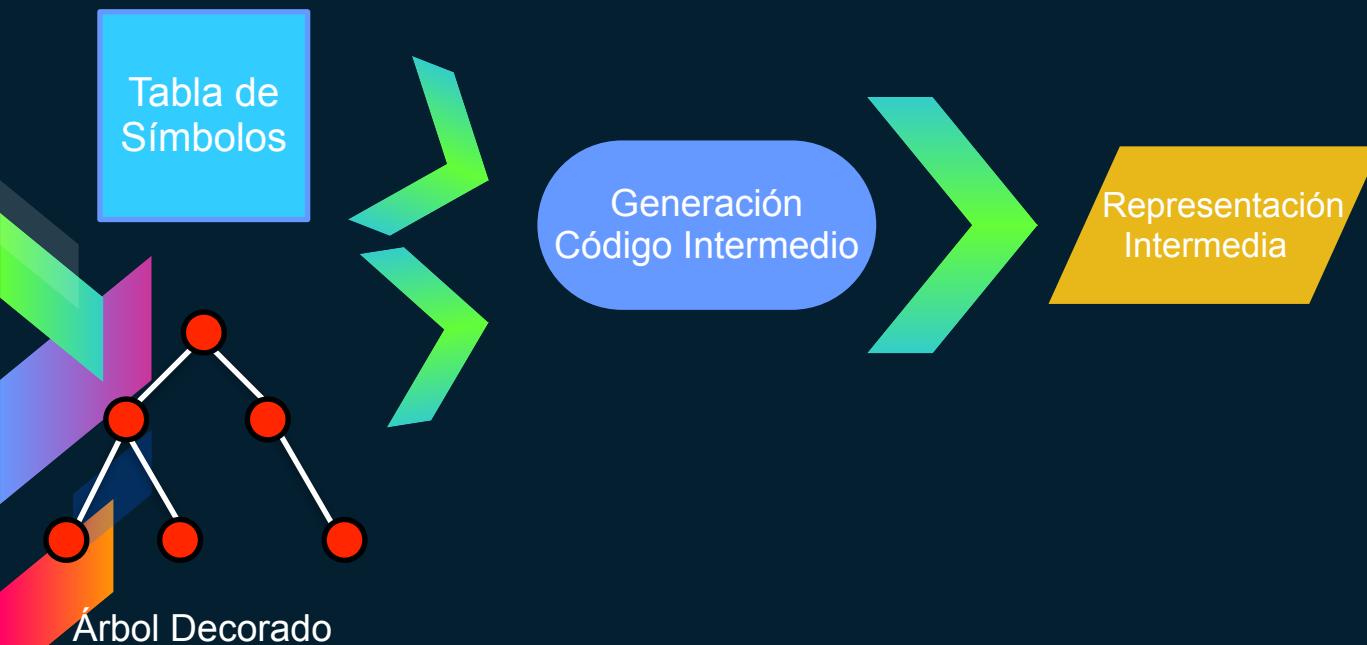
$$T3 = T1 + T2$$

$$T4 = A + T3$$

- Analiza el árbol y crea un código intermedio de 3 direcciones
- Creando variables temporales para cada “arbolito” del árbol grande, de cada operación que se ocupa ir haciendo
- Esto pasa en los finales del Front End

Entrada y Salida de Generación Código Intermedio

- Se podría decir que un Scanner es un chunche mágico que le entra un árbol decorado y una tabla de símbolos y le genera una representación intermedia



Generador de Código

- Aquí ya se empieza en el Back End
- Aquí ya me empieza a interesar la arquitectura
- Ya para este punto ya no importa en lo que este escrito el programa fuente ya que el código ahorita está en código intermedio
- Ya se convierte de Código intermedio a Código Real



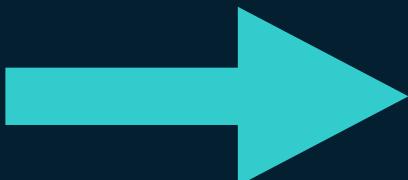
- Se puede generar un ensamblador base ("cochino")
- La representación de Datos es muy importante:
 - Hay que saber de la Arquitectura que se quiere llegar y sus capacidades
 - La cantidad de bytes que se van a reservar, algunos van de 4 en 4

PRO TIP:
Tener un template para convertir código intermedio a código ensamblador

Generador de Código - Ejemplo

$$A = B * C + D * E$$

```
T1 = B * C  
T2 = D * E  
T3 = T1 + T2  
T4 = A + T3
```

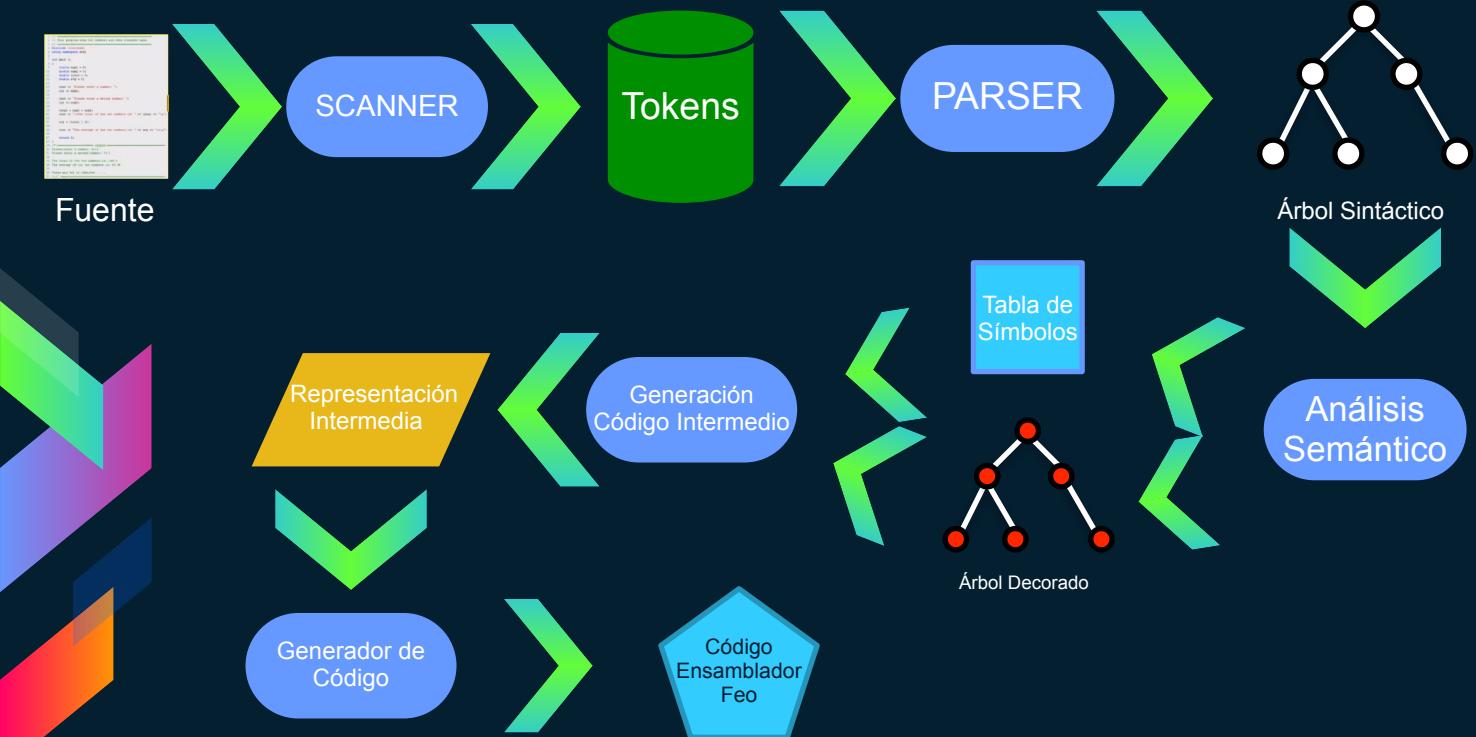


```
mov AX, C  
mul B  
mov T1, AX  
  
mov AX, E  
mul D  
mov T2, AX  
  
mov AX, T2  
add AX, T2  
mov T3, AX  
  
mov AX, T3  
mov A, AX  
mov T3, AX
```

- Se va agarrando cada línea del código intermedio y se va convirtiendo en ensamblador, por eso es útil el template
- Ya aquí queda un código ensamblador funcional aunque tal vez un poco feo.

Fases de Compilación

- El proceso de las fases de compilación hasta el momento va así:



Compiladores e intérpretes

Apuntes: 23/09/2020

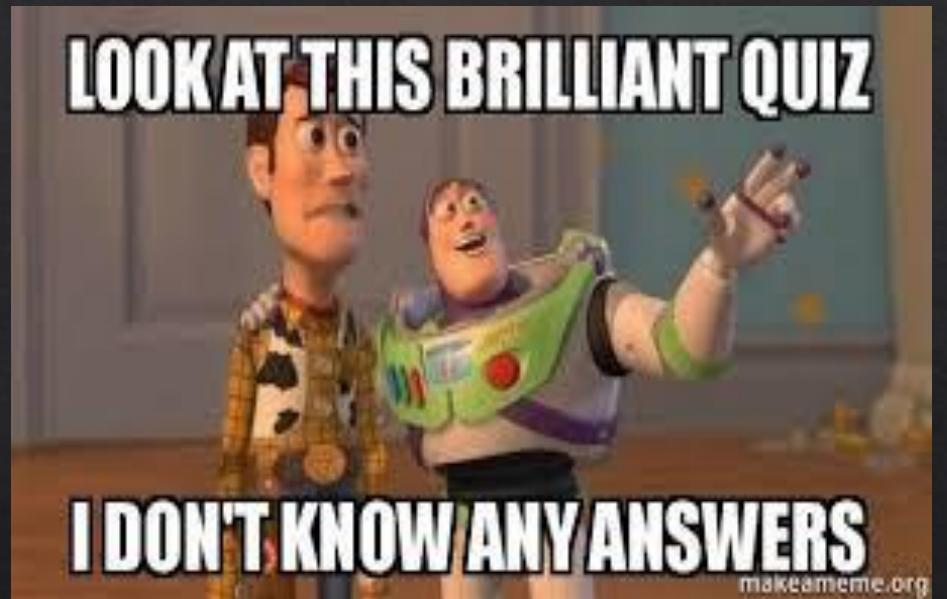
Steven Retana Cedeño

Quiz 2

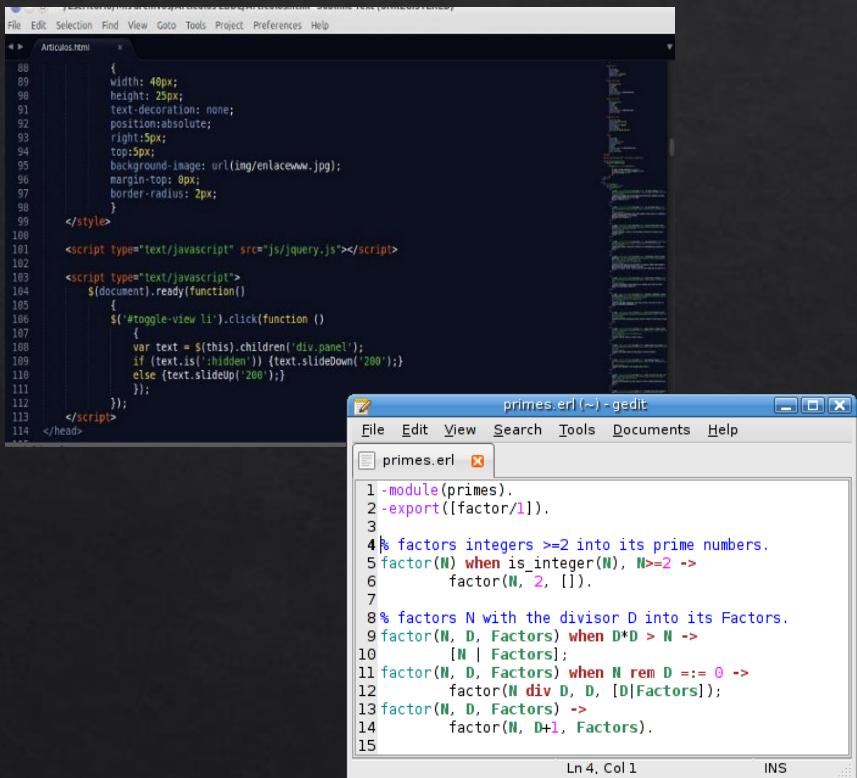
1. Defina detalladamente los siguientes conceptos
 - ◊ Preprocesador
 - ◊ Linking dinámico
 - ◊ Relocalización
 - ◊ Backpatching
 - ◊ Front end

2. Suponga que su sistema Linux tiene un compilador de C, versión 7. Usted recibe por correo los siguientes archivos:
 - ◊ El programa Fuente de la versión 10 del compilador de C, escrito en C.
 - ◊ El Fuente de un compilador de Fortran que genera C, escrito en C.
 - ◊ Un compilador de Cobol, escrito en Fortran, que genera lenguaje máquina.

Usando diagramas T y explicaciones detalladas, diga lo mejor que se puede hacer con esto



Editor de texto



```
File Edit Selection Find View Goto Tools Project Preferences Help
Articulos.html
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
<style>
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
</style>
<script type="text/javascript" src="js/jquery.js"></script>
<script type="text/javascript">
$(document).ready(function()
{
  $('#toggle-view li').click(function ()
  {
    var text = $(this).children('div.panel');
    if (text.is(':hidden')) {text.slideDown('200');}
    else {text.slideUp('200');}
  });
});
</script>
</head>
<body>
```

```
primes.erl (~) - gedit
File Edit View Search Tools Documents Help
primes.erl
1 -module(primes).
2 -export([factor/1]).
3
4 % factors integers >=2 into its prime numbers.
5 factor(N) when is_integer(N), N>=2 >->
6     factor(N, 2, []).
7
8 % factors N with the divisor D into its Factors.
9 factor(N, D, Factors) when D*D > N >->
10    [N | Factors];
11 factor(N, D, Factors) when N rem D == 0 >->
12    factor(N div D, D, [D|Factors]);
13 factor(N, D, Factors) >->
14    factor(N, D+1, Factors).
15
```

Ln 4, Col 1 INS



Usan técnicas de compiladores para sus funciones



Expresiones regulares para búsquedas



Poseen un código cargado para alterar el comportamiento (Ejemplo, autocompletado en un lenguaje)



Existen editores de propósito general u orientados a un lenguaje en específico

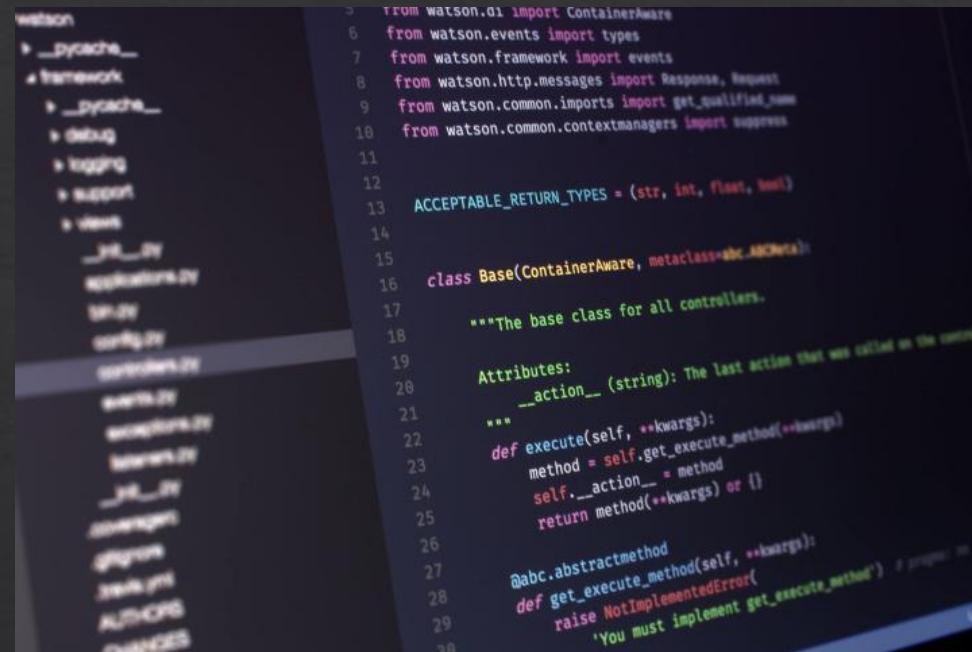
Características de los editores de texto

Conocen la sintaxis del lenguaje

Genera partes del código para autocompletar y facilitar la programación

Marca con colores o tipos de letra especial según corresponda al lenguaje

El editor informa de errores sintácticos



```
from watson.di import ContainerAware
from watson.events import types
from watson.framework import events
from watson.http.messages import Response, Request
from watson.common.imports import get_qualified_name
from watson.common.contextmanagers import suppress

ACCEPTABLE_RETURN_TYPES = (str, int, float, bool)

class BaseController(ContainerAware, metaclass=abc.ABCMeta):
    """The base class for all controllers.

    Attributes:
        __action__ (string): The last action that was called on the controller.

    def execute(self, **kwargs):
        method = self.get_execute_method(**kwargs)
        self.__action__ = method
        return method(**kwargs) or {}

    abc.abstractmethod
    def get_execute_method(self, **kwargs):
        raise NotImplementedError(
            'You must implement get_execute_method'
        )
```

Junto con el compilador, el profiler y debugger forman un entorno de desarrollo integrado (Integrated Development Environment)
I.D.E

Manejadores de versiones

Varias personas trabajando sobre los mismos fuentes o partes de ellos.

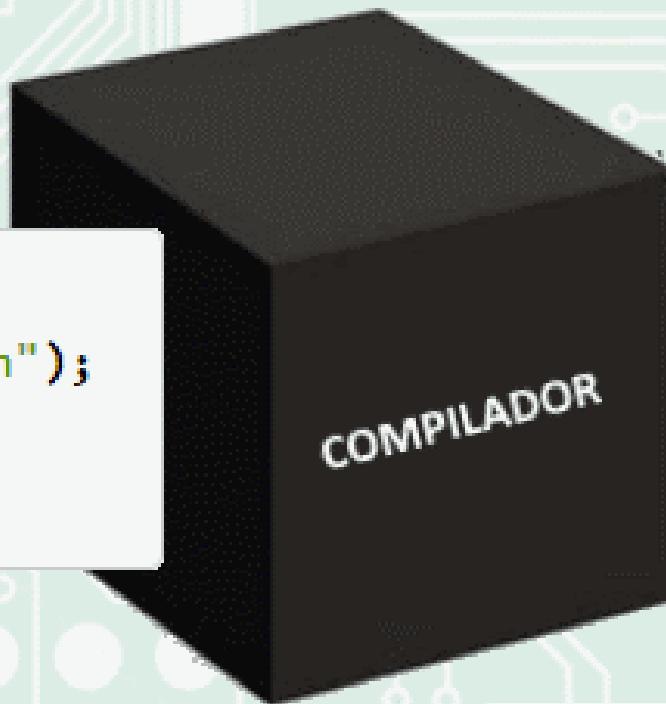
Ayudan a coordinar las versiones de los códigos.

Documentan quién hizo algo y en qué momento.

Permite regresar a puntos previos del Desarrollo.



```
int main() {  
    printf("¡Hola compilador!\n");  
    return 0;  
}
```



```
111011011100010100000011100001100011  
1110110111000101000000111000101000000  
1000011011000111101101100010100000011100  
10110001111011011101100010100000011100  
0110001111011011101100010100000011100  
0110001111011011101100010100000011100  
00011110110111000101000000111000011  
11011000111101101100010100000011100  
00011011000111101101110110001010000001110  
0110001111011011101100010100000011100  
11101101110110001010000001110000111000  
00011110110111011000101000000111000011  
11101101110110001010000001110000111000  
00011110110111011000101000000111000011
```

Estructura básica de los compiladores

Entrada: Lenguaje fuente a ser traducido por el compilador.

```
1 // Programa de búsqueda lineal codificado en C++
2
3 #include <iostream.h>
4 void main()
5 {
6     int LISTA[10] = {11, 9, 27, 32, 52, 46, 8, 94, 88, 26};
7     int INICIO = 0;
8     int FINAL = 9;
9     int VALOR, band, ap, i;
10    char mas = 's';
11
12    while ((mas == 's') || (mas == 'S')){
13        cout << "\nLa lista es: ";
14        for(i = INICIO; i <= FINAL; i++)
15            cout << LISTA[i] << " ";
16        cout << endl;
17        band = 0;
18        ap = INICIO;
19        cout << "Dame el valor a buscar: ";
20        cin >> VALOR;
21
22        while ((ap <= FINAL) && (band == 0))
23            if (LISTA[ap] == VALOR)
24                band = 1;
25            else
26                ap++;
27
28        if (band == 1)
29            cout << "Encontre el numero " << VALOR << " en la localidad" << "No. " << ap+1 << endl;
30        else
31            cout << "El numero " << VALOR << "No esta en la lista\n";
32
33        cout << "Desea Continuar? (s/n): ";
34        cin >> mas;
35    }
36    cout << "\nAdios\n";
37 }
```

Salida: Lenguaje objeto al que llega el compilador.

```
1 section .text
2     global _start
3
4 _start:
5     mov edx, len
6     mov ecx, msg
7     mov ebx, 1
8     mov eax, 4
9     int 0x80
10
11    mov eax, 1
12    int 0x80
13
14 section .data
15 msg db 'Hola, mundo!', 0xa
16 len equ $ - msg
```

Representación intermedia: Es creada en los procesos de compilación y sirve como un intermedio entre el back end y front end para luego convertirse en el lenguaje objeto.

El proceso de compilación se puede dividir en 2 fases:

Fase 1

Lenguaje fuente

```
1 #include <iostream>
2 int FINAL = 5;
3 int VALOR, band, ap;
4 char msg;
5 const int LISTA[5] = {1, 2, 3, 4, 5};
6
7 int main()
8 {
9     cout << "Introduzca el valor a buscar: ";
10    cin >> VALOR;
11
12    while ((exp <= FINAL) && (band == 0))
13    {
14        if (LISTA[ap] == VALOR)
15            band = 1;
16        else
17            ap++;
18
19        if (band == 1)
20            cout << "Encontro el numero " << VALOR << " en la localidad" << "No. " << ap << endl;
21        else
22            cout << "El numero " << VALOR << "No esta en la lista\n";
23
24        cout << "Desea continuar? s/n: ";
25    }
26 }
```

Front End

Representación intermedia

Fase 2

Representación intermedia

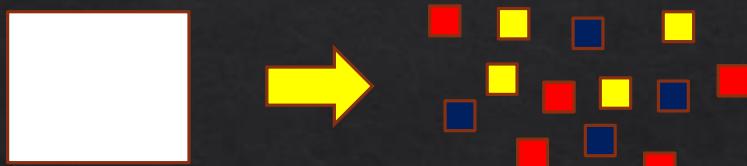
Back End

Lenguaje fuente

```
4 _start:
5     mov    edx, len
6     mov    ecx, msg
7     mov    ebx, l
8     mov    eax, 4
9     int    0x80
10
11    mov    eax, 1
12    int    0x80
```

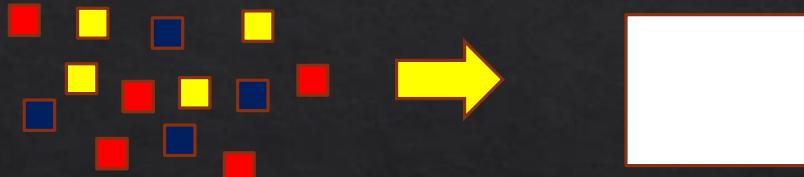
Recordadando...

Analizar: Tomar algo y descomponerlo en partes para lograr entenderlo.



Front End: Tiene que ver con el código fuente, el código que recibe para procesar. (**Análisis**)

Sintetizar: Construir algo a partir de varias piezas sueltas.



Back End: Es la parte que trabaja en la arquitectura de destino, en el Código que se genera. (**Síntesis**)

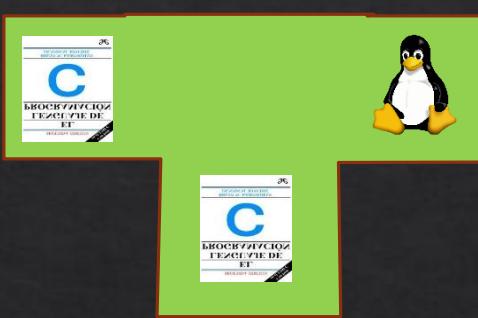
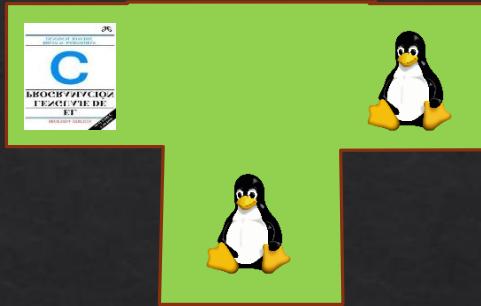
Ejemplo

- ❖ Debemos desarrollar en Linux un compilador de C
- ❖ Este compilador va a ejecutar en Windows
- ❖ Va a generar Código que corra en Windows

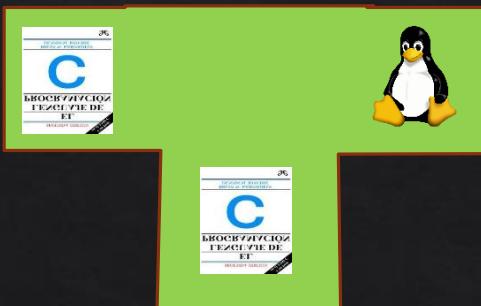


Tenemos disponible lo siguiente:

- ❖ Un compilador de **gcc** que ejecuta **Linux** y genera Código para **Linux**.
- ❖ Poseemos el fuente del compilador **gcc**



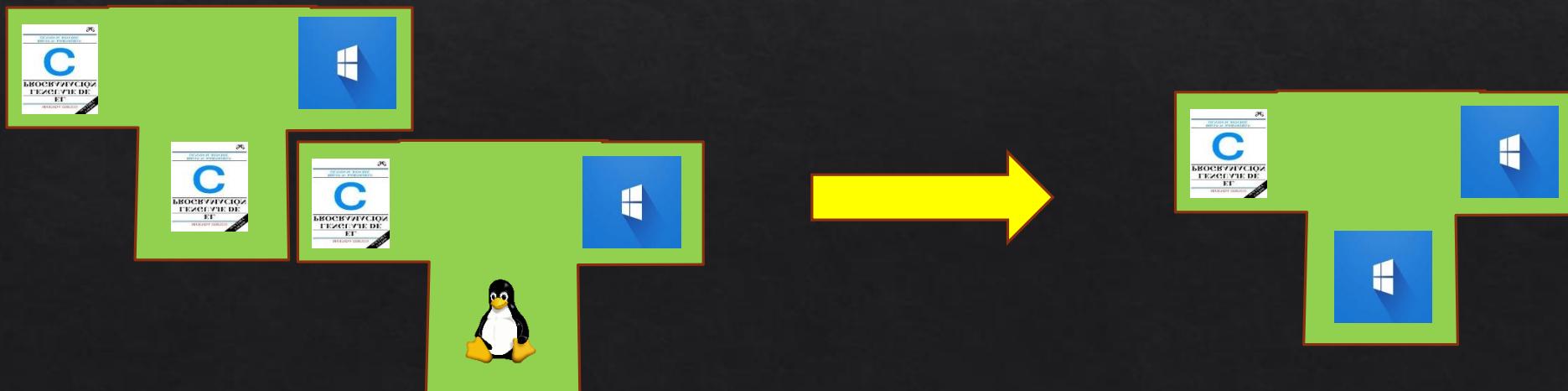
Podemos estudiar el fuente e identificar el back end, luego modificarlo para que genere código **Windows**.

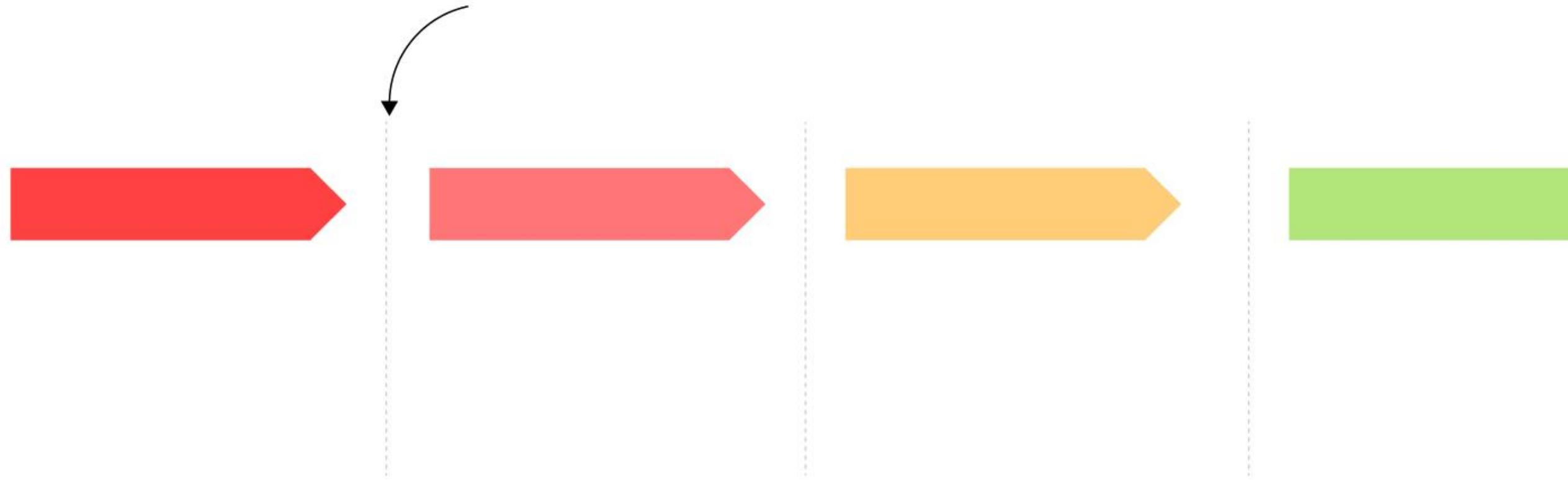


Compilamos con gcc en Linux



Compilamos con nuevo compilador en Linux





Fases de compilación



Pasadas

Pasada: Lectura completa del fuente de inicio a fin.

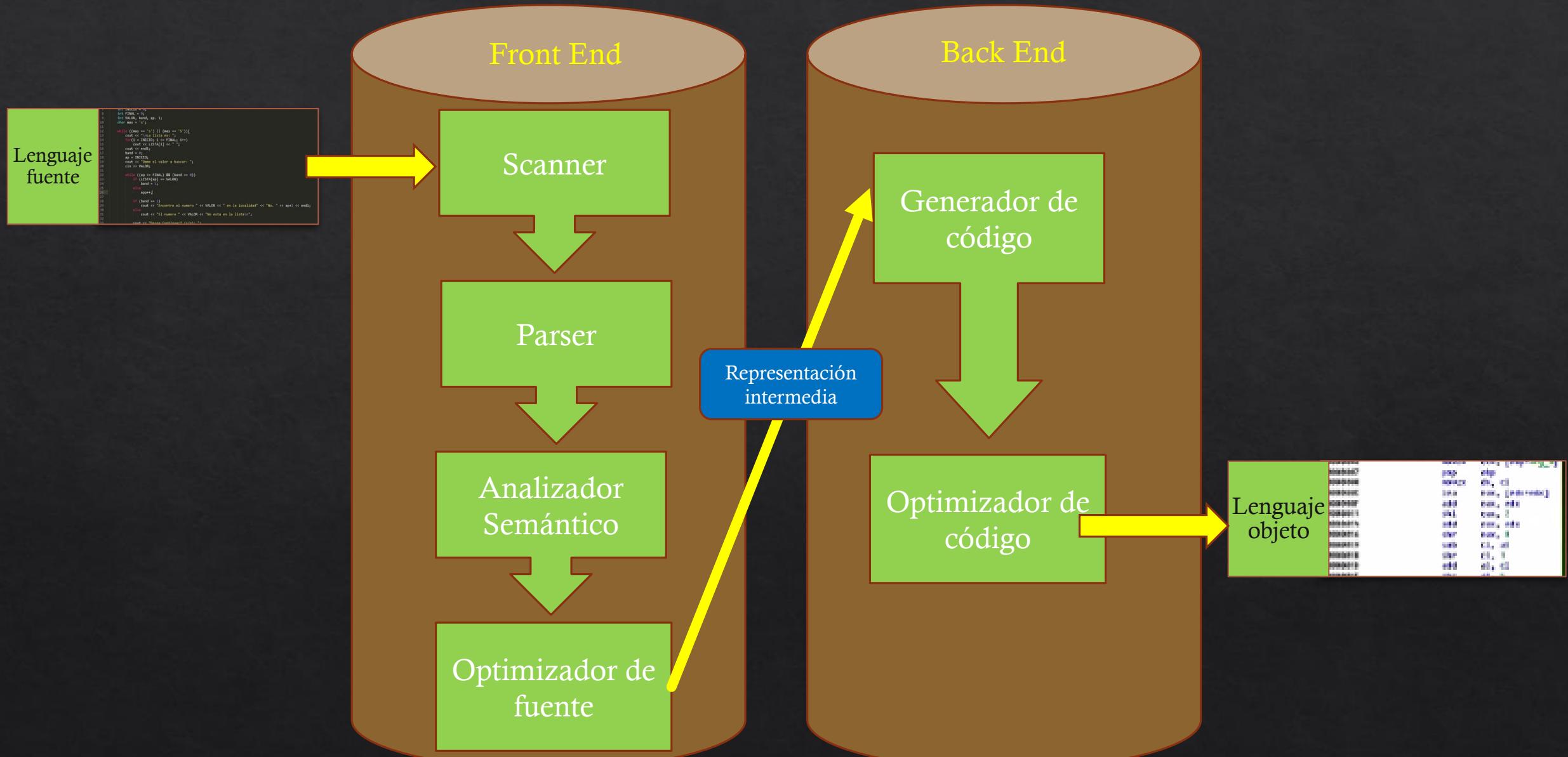
“N pasadas”: Leer **N** veces el Fuente de inicio a fin.

Algunas definiciones pueden estar físicamente después de su uso.

Más pasadas implican más flexibilidad

Más pasadas también implican más tiempo de compilación.

Fases de compilación



Análisis Léxico

- ❖ Léxico: Vocabulario
- ❖ Es conocido como “scanner”
- ❖ Toma el Fuente de entrada y lo descompone en unidades léxicas mínimas llamadas “tokens”
- ❖ Implementados con autómatas de estados finitos

```
int SumaVariables (int v1, int v2)
```

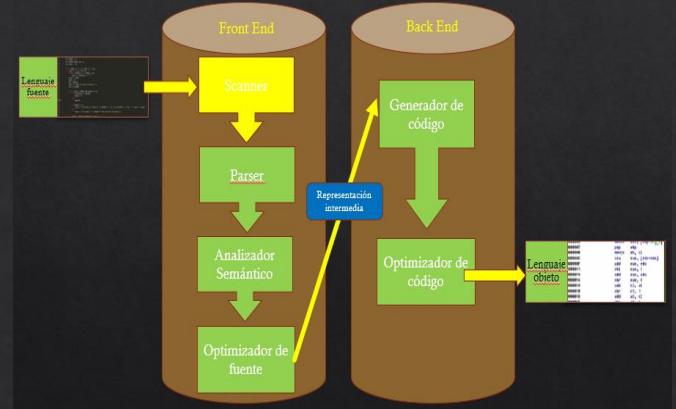
```
{
```

```
    return v1 + v2;
```

```
}
```



Distintos tokens del fuente de entrada



Análisis Sintáctico

Revisa que la **sintaxis** de un programa esté correcta

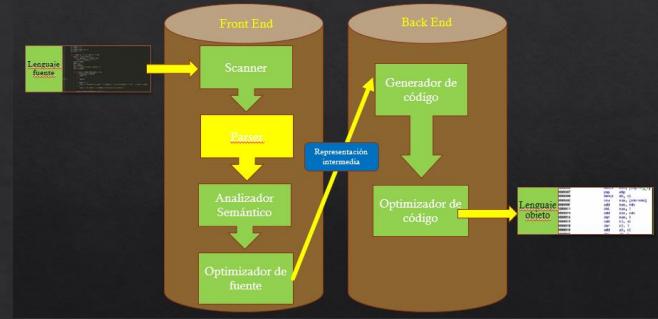
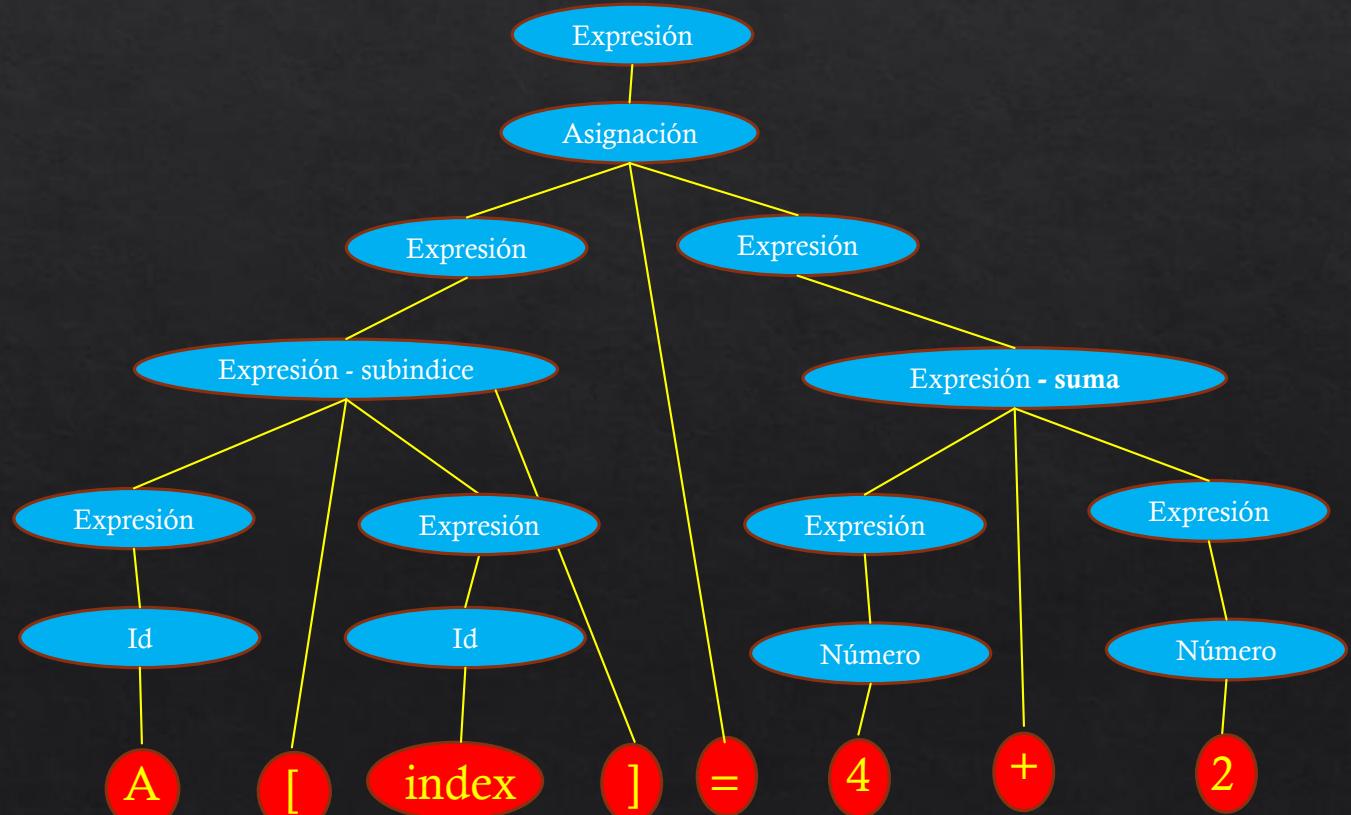
Es conocido como “**parser**”

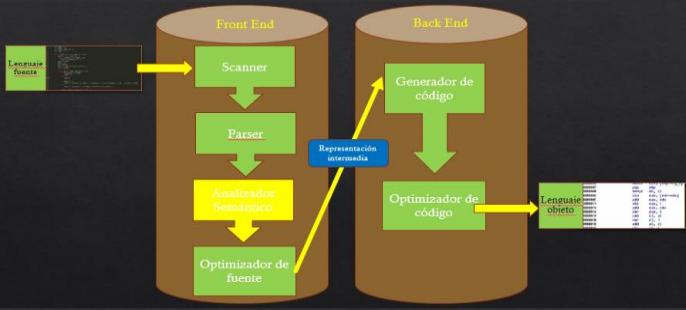
Toma la secuencia de **tokens** y verifica que correspondan a construcciones válidas.

Requieren de una **gramática**

Construye el **árbol sintáctico**

$A [\text{index}] = 4 + 2$





Análisis Semántico

$$A [\text{index}] = 4 + 2$$

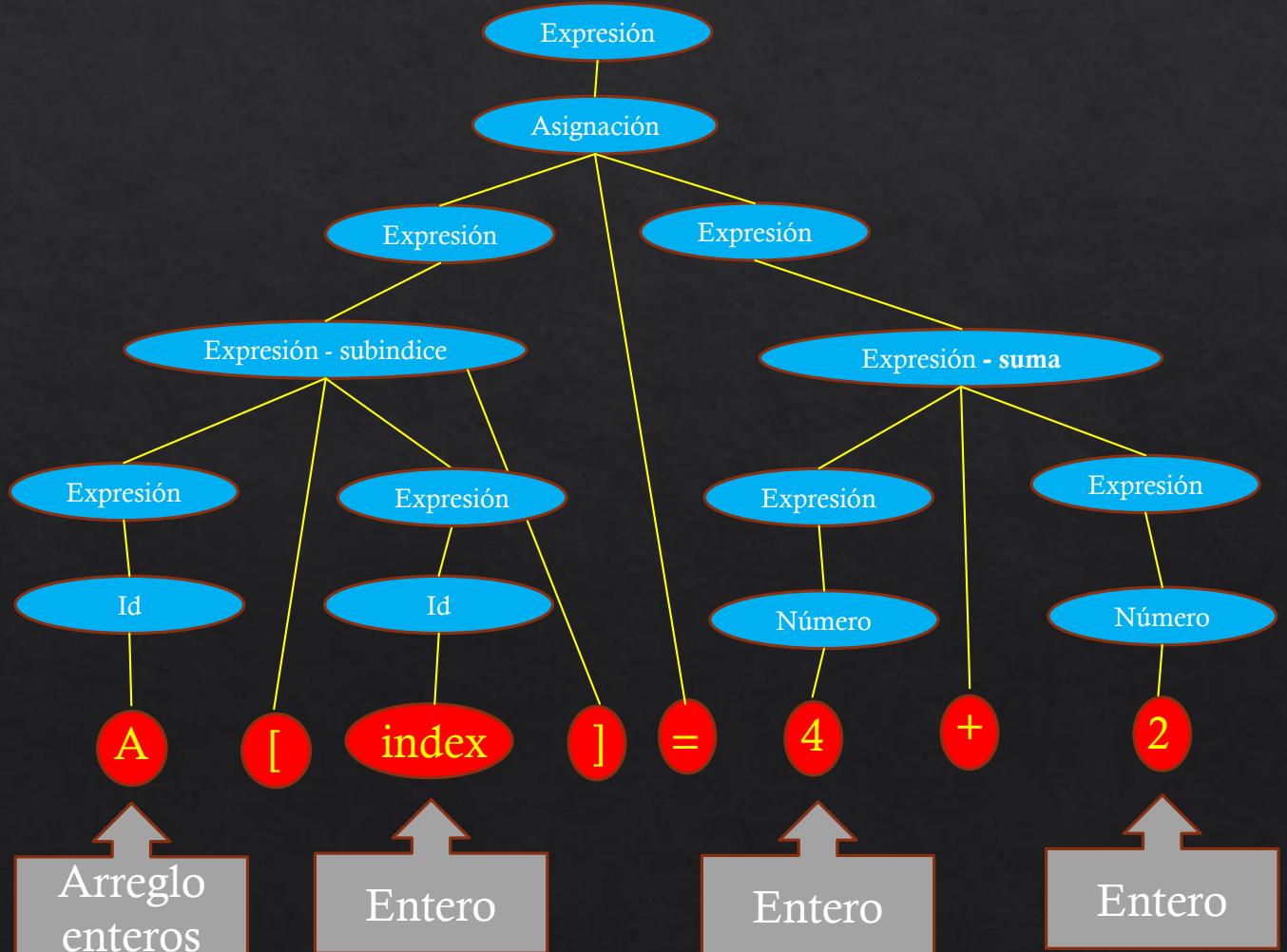
Revisar si tiene sentido lo que se pide que se haga

Determina su comportamiento en tiempo de corrida

Semántica estática

Declaraciones y revisión de tipos

Construye la “tabla de símbolos” y el “árbol de atributos” o “árbol decorado”



Optimizador de código fuente

Introduce optimizaciones

Cálculo y propagación de constantes

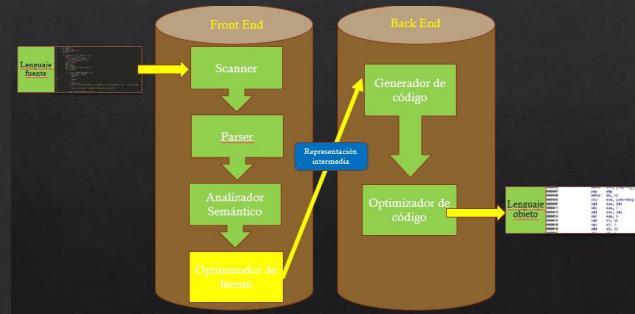
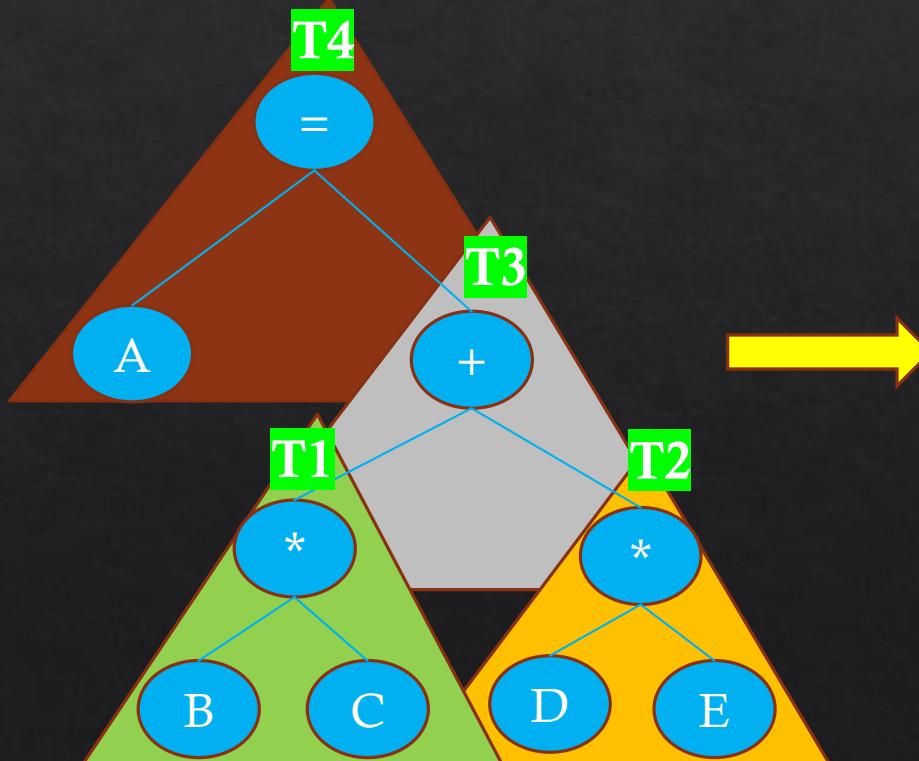
Eliminación de código innecesario

Similar a un ensamblador de 3 direcciones

Genera el “código intermedio”

Código intermedio

$$A = B * C + D * E$$



$$T1 = B * C$$

$$T2 = D * E$$

$$T3 = T1 + T2$$

$$T4 = A = T3$$

Generador de código

Back end del compilador

Convierte el código intermedio en código real

Representación de datos es muy importante, cantidad de bytes, restricciones de colocación.

Se requiere conocer y explotar las posibilidades de la arquitectura

Código intermedio

$$A = B * C + D * E$$

$$T1 = B * C$$

$$T2 = D * E$$

$$T3 = T1 + T2$$

$$T4 = A = T3$$



Código real

mov ax, C

mul B

mov T1, ax

mov ax, E

mul D

mov T2, ax

mov ax, T2

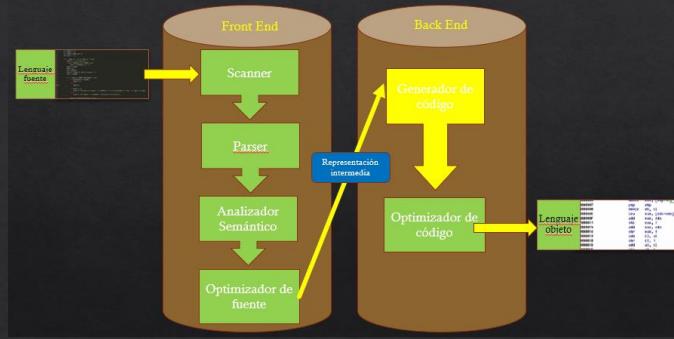
Add ax, T1

mov T3, ax

mov ax, T3

add A , ax

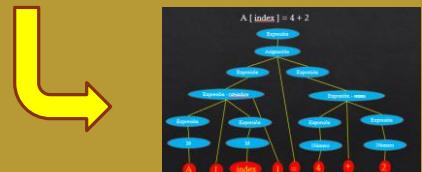
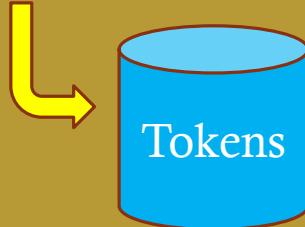
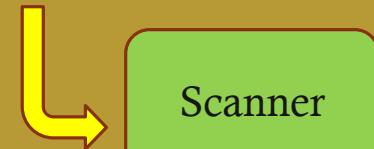
mov T4, ax



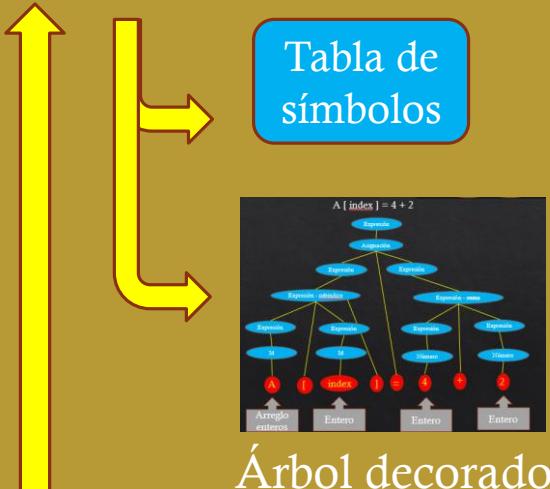
Hasta el momento tenemos los siguientes elementos:

Lenguaje fuente

```
int main() {  
    int VALOR, band, ap; //  
    char c;  
    cout << "Introduzca el valor: ";  
    cin >> VALOR;  
    cout << "Introduzca la bandera: ";  
    cin >> band;  
    cout << "Introduzca el apuntador: ";  
    cin >> ap;  
    cout << endl;  
    cout << "El resultado es: ";  
    cout << ap + VALOR * band;  
    cout << endl;  
    cout << endl;  
    cout << endl;
```



Análisis Semántico



Front End

Generación de representación intermedia

Representación intermedia

Back End

Generador de código

Lenguaje Máquina



Continuará...

Compiladores e Intérpretes Q40

Apuntes viernes 25 de septiembre

RONALD HERRERA GÁMEZ

**“
GREAT THINGS NEVER COME
FROM COMFORT ZONES.**



Comentarios varios

- Al enviar cualquier documento escrito adjuntar PDF y archivo .tex de LaTeX
- Se crearon personajes ficticios en las notas para ir viendo como vamos

Significado

Perfecto: ha sacado 100 en todos los trabajos realizados hasta el momento.

Pasando: ha sacado 67.5 en todos los trabajos.

Promedio: sus notas se determinan promediando las notas de cada trabajo.

Último: ha sacado 0 en todos los trabajos.

Optimizador de código

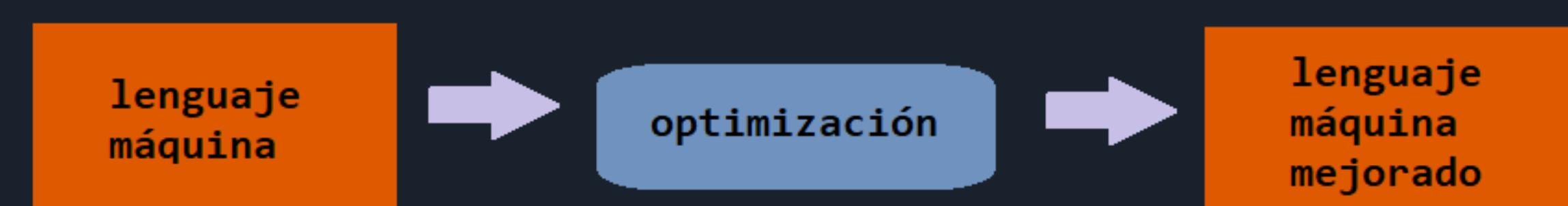
- Mejora el código generado en el paso previo (en generación de código)
- Genera instrucciones **equivalentes**, pero más rápidas. Ejm:
 - INC es más rápida que ADD
- Modos de direccionamiento más apropiados
 - Es mejor guardar los datos más usados en registros que memoria
- Elimina código innecesario o redundante
- Se le debe indicar al compilador si queremos optimizar el fuente para:



Tiempo: Reduce el tiempo de ejecución



Espacio: optimiza el tamaño del ejecutable

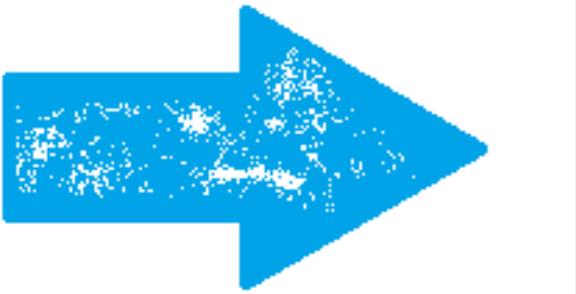


Código generado por el generador de código

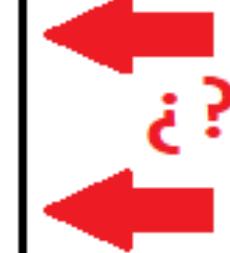
$$A = B*C + D*E$$



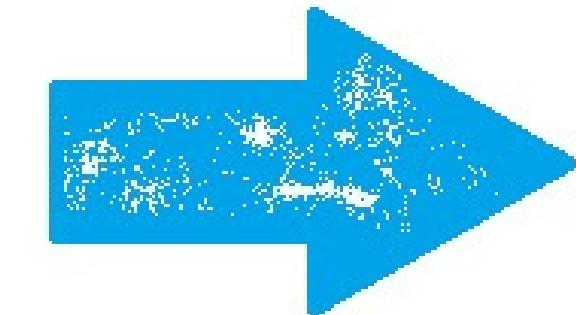
¡Redundancia!



```
mov ax, C  
mul B  
mov T1, ax  
-----  
mov ax, E  
mul D  
mov T2, ax  
-----  
mov ax, T2  
add ax, T1  
mov T3, ax  
-----  
mov ax, T3  
mov A, ax  
mov T4, ax
```



Código ensamblador optimizado



```
mov ax, C  
mul B  
mov dx, ax  
mov ax, D  
mul E  
add ax, dx  
mov A, ax
```



*Las técnicas de optimización son agresivas,
se debe analizar bloques completos para ver qué se puede optimizar.*

Ejercicio moral

1) Pedirle al compilador que genere el código ensamblador sin optimizar.

2) Luego, pedirle el código optimizado (ya sea para espacio o para tiempo).

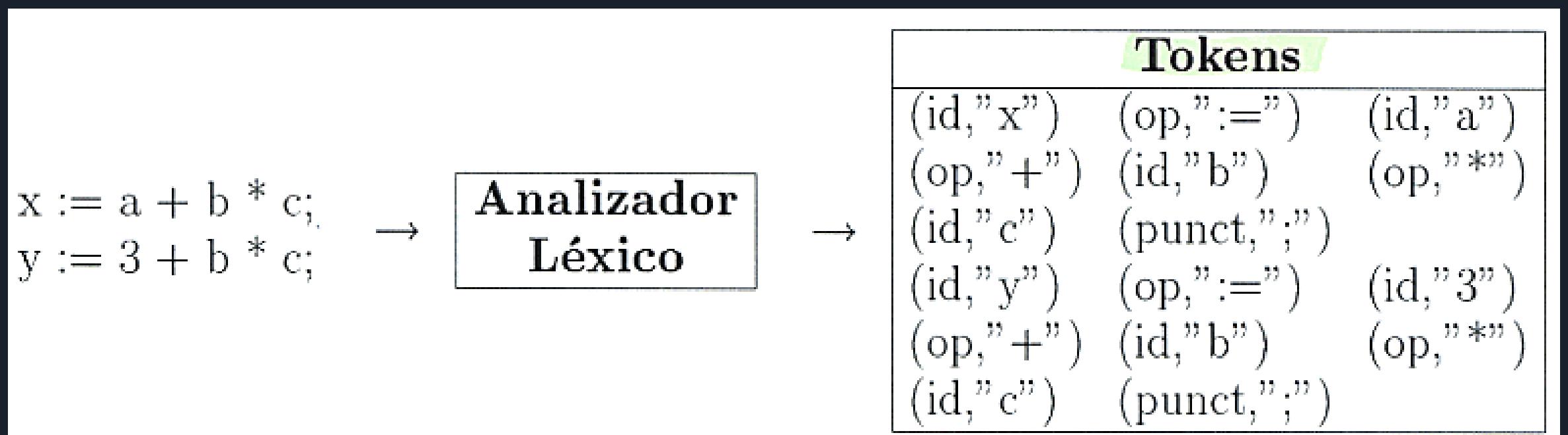
3) Finalmente, comparar.

PD: Para nuestro proyecto no es necesario hacer la optimización.



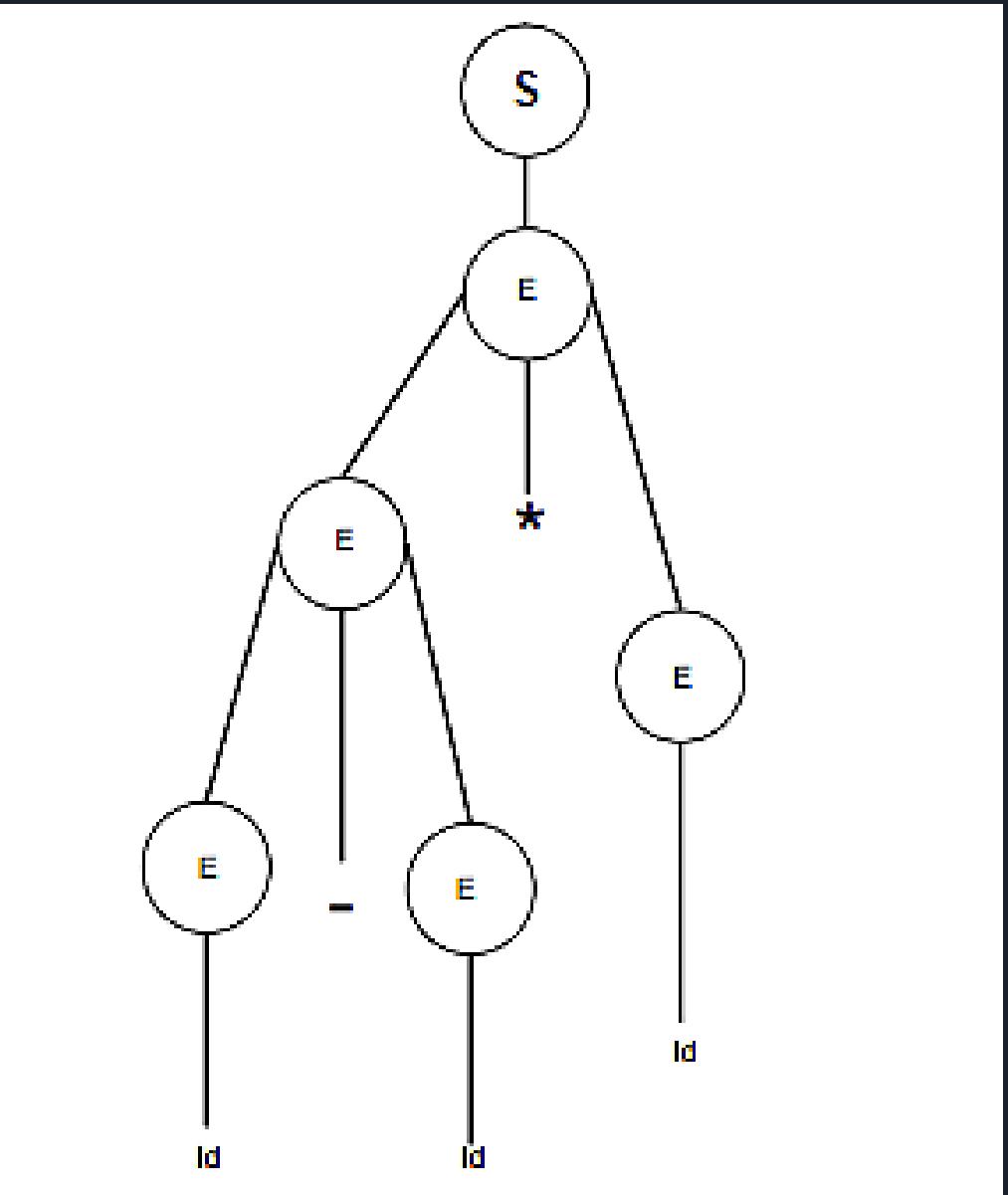
Tokens

- Regresados por el scanner
- Es usualmente una estructura con dos campos:
 - **Código de token**: tipo al que pertenece (es un número internamente)
 - {VARIABLE, INTEGER-CONSTANT, COMMA, LEFT-PARENTHESES}
 - **Hilera específica (lexema)**: secuencia de caracteres concreta encontrada en el código fuente



Árbol Sintáctico

- Creado por el **parser**
- Decorado por analizador semántico (le mete información extra)
- Puede ser un árbol hecho en memoria dinámica donde hay una variable que señala la raíz del árbol.
- Cada una de las partes del árbol puede ser una estructura, con una serie de campos que van siendo llenados por el parser y el analizador semántico
- Podrían ser punteros a otras estructuras
- A veces, no existe explícitamente como una estructura específica.
 - Ejemplo, micro no crea un árbol sintáctico, pero si está ahí.



Árbol Sintáctico

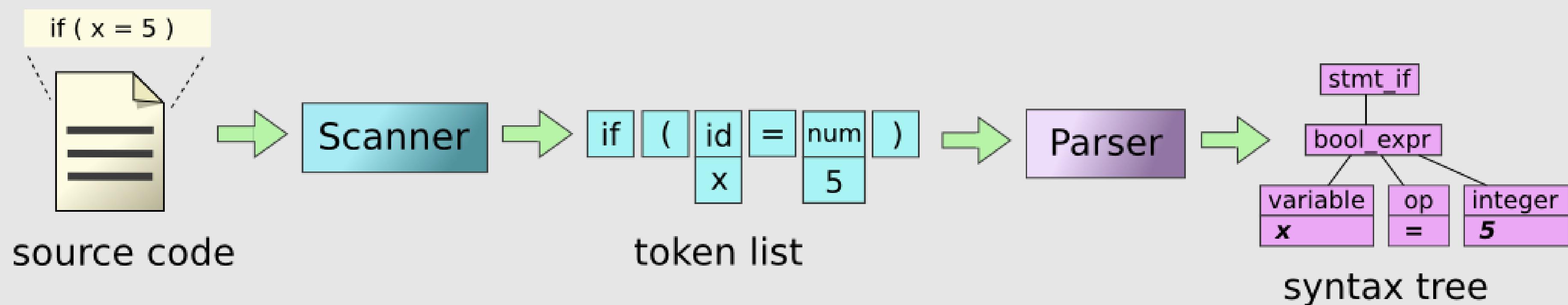


Tabla de símbolos

- Conserva la información asociada a todos los identificadores
 - Variables, constantes, tipos de datos, procedimientos y funciones
- Interactúa con el scanner, parser, analizador semántico, generadores, etc.
- Es la estructura de datos más usada en el compilador.
- Al ser la más usada es conveniente que sea **eficiente**, para que el compilador sea más rápido
- Ejemplo, usar tablas de hash, para hacer más rápido el acceso a la tabla de símbolos
- Puede estar formada por varias tablas

Tabla de literales

- Distintas a los símbolos en que son constantes (números, hileras)
- Es decir, su valor **no cambia** en la ejecución
- Deben residir en memoria.
- Pueden venir punteros de la Tabla de Símbolos.
- Ayuda a reducir tamaño final del código

"En un programa es habitual que haya constantes, todas esas constantes podrían estar en una sola tabla para hacer que el código sea más eficiente"

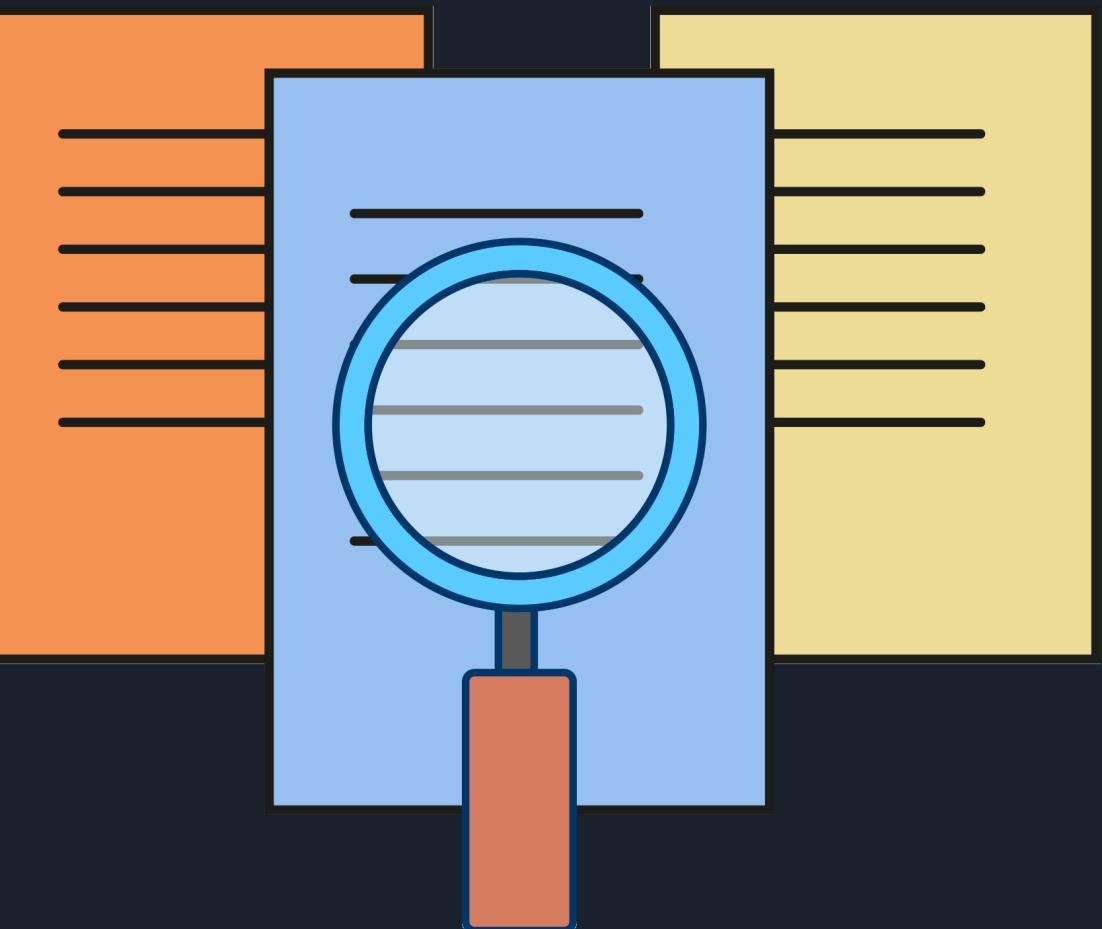
Código intermedio

- Algunas posibilidades para representar el código intermedio:
 - **Arreglo de punteros a hileras**
 - O más sencillo, **archivo de texto**
 - **Lista enlazada de hileras**, etc.
- Puede ser que en la optimización se necesite reorganizar el código intermedio:
 - Manipularla: **reemplazar**, **eliminar** o **reordenar**



Archivos Temporales

- Archivos de vida corta
 - **Creación - escritura - lectura - borrado**
- Típicos en muchos compiladores o en Sistemas Operativos
- Viven segundos
- Son nuestros amigos, resuelven problemas complejos fácilmente

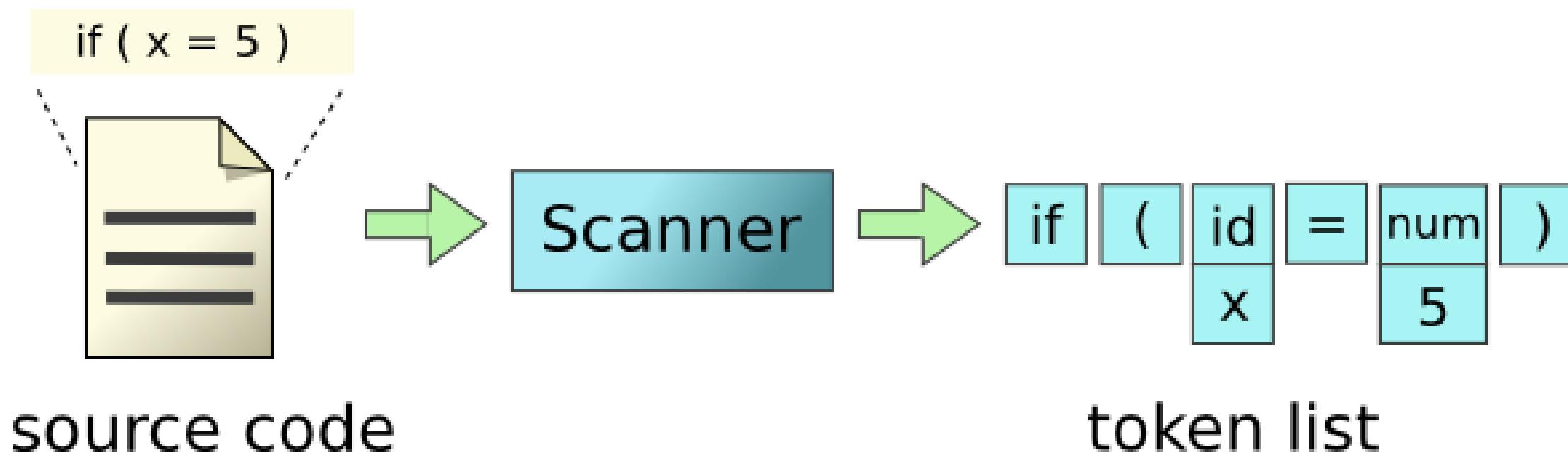




Recordando la clase anterior

Análisis léxico (**scanner**)

- Léxico: **vocabulario**
- Un **token** equivale a una palabra en lenguaje natural
- Pensémoslo de manera abstracta como que toma el fuente de entrada y lo descompone en unidades léxicas mínimas (tokens)



Ejemplo: Tokens encontrados por el scanner

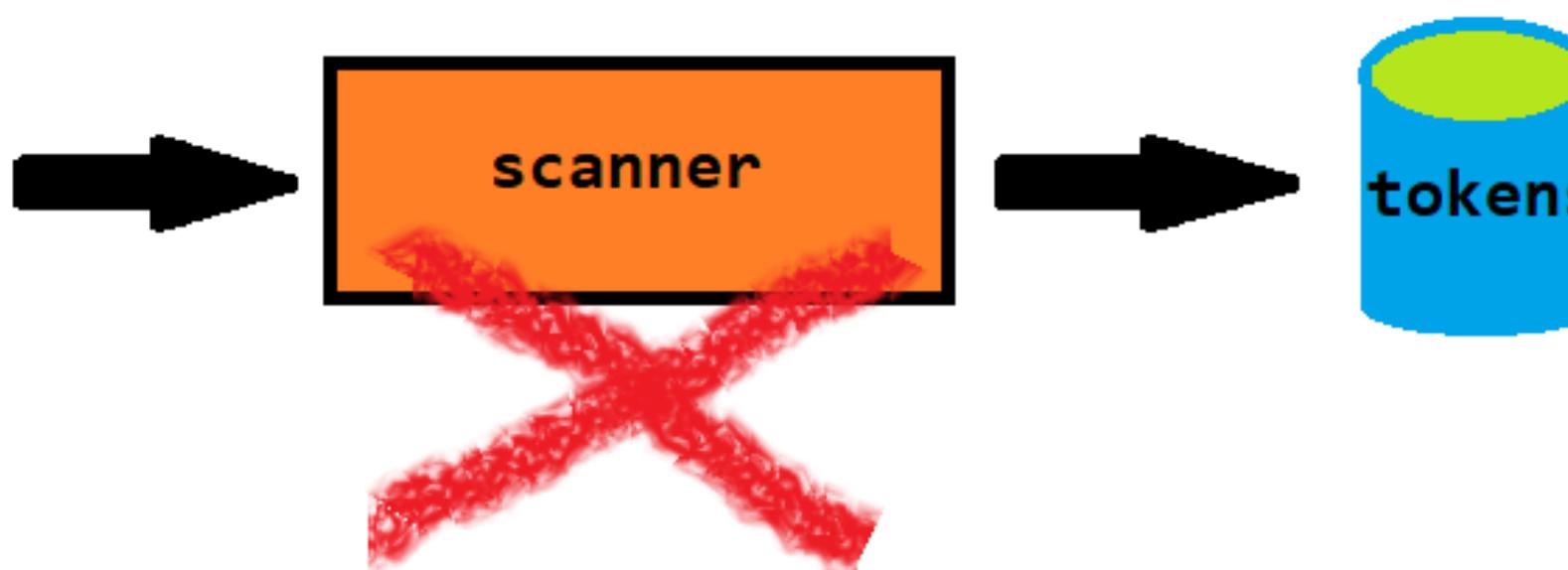
```
#include <stdio.h>
int maximum(int x, int y) {
    // This will compare 2 numbers
    if (x > y)
        return x;
    else {
        return y;
    }
}
```

	Lexeme	Token
1	int	Keyword
2	maximum	Identifier
3	(Operator
4	int	Keyword
5	x	Identifier
6	,	Operator
7	int	Keyword
8	y	Identifier
9)	Operator
10	{	Operator
11	If	Keyword

Nota

*Hasta ahora hemos visto que el scanner toma el fuente y lo convierte en tokens. Sin embargo, **es mentira**. Sí es cierto que encuentra token por token, pero no que corre completamente hasta terminar.*

```
/**  
 * Simple HelloButton() method.  
 * @version 1.0  
 * @author john doe <doe.j@example.com>  
 */  
HelloButton()  
{  
    JButton hello = new JButton( "Hello, wor  
hello.addActionListener( new HelloBtnList  
    // use the JFrame type until support for t  
    // new component is finished  
    JFrame frame = new JFrame( "Hello Button"  
    Container pane = frame.getContentPane();  
    pane.add( hello );  
    frame.pack();  
    frame.show();           // display the fra  
}
```

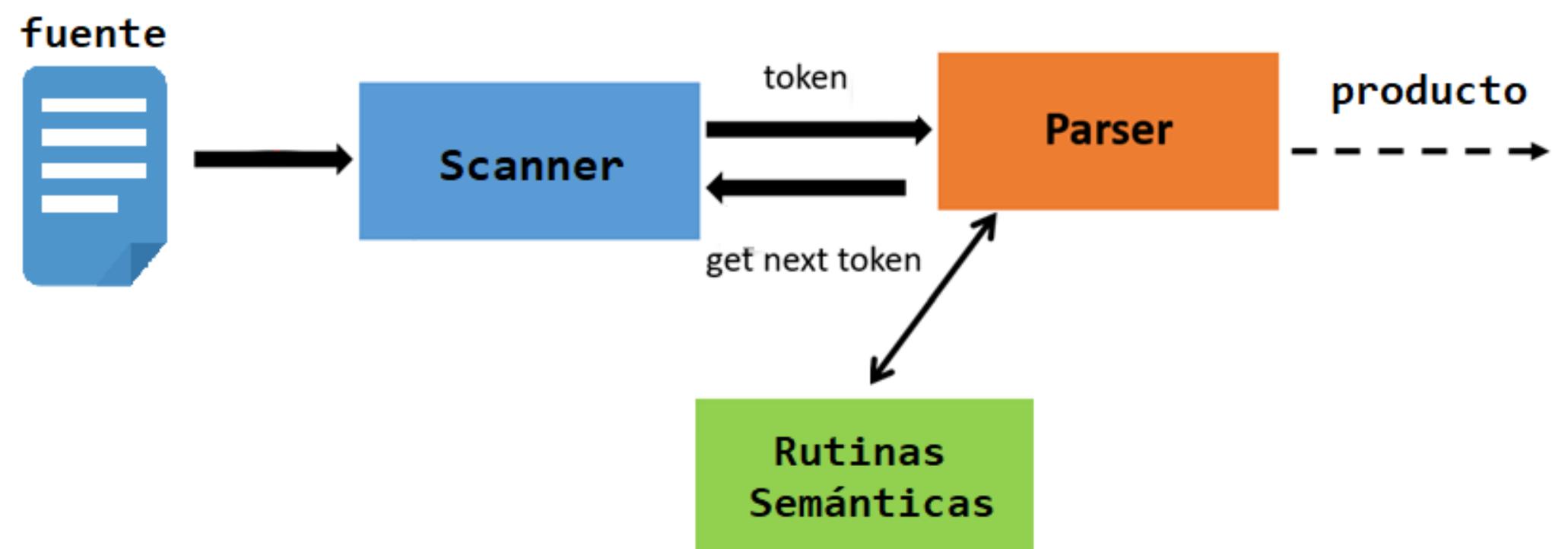


Entonces, ¿Cómo funciona realmente?

Se usa una estructura llamada **Traducción Dirigida por Sintaxis**

- Es la organización más usual en compiladores
- El **parser** dirige todo el proceso (validando la gramática)
 - es el parser que invoca al scanner cada vez que necesita un token
 - Además, invoca rutinas semánticas en los puntos apropiados

- Su trabajo podría terminar al generar el código intermedio o inclusive podría generar lenguaje de máquina
- Es común que haya una función:
 - Token getToken (void);
 - O en Micro “nextToken ()”

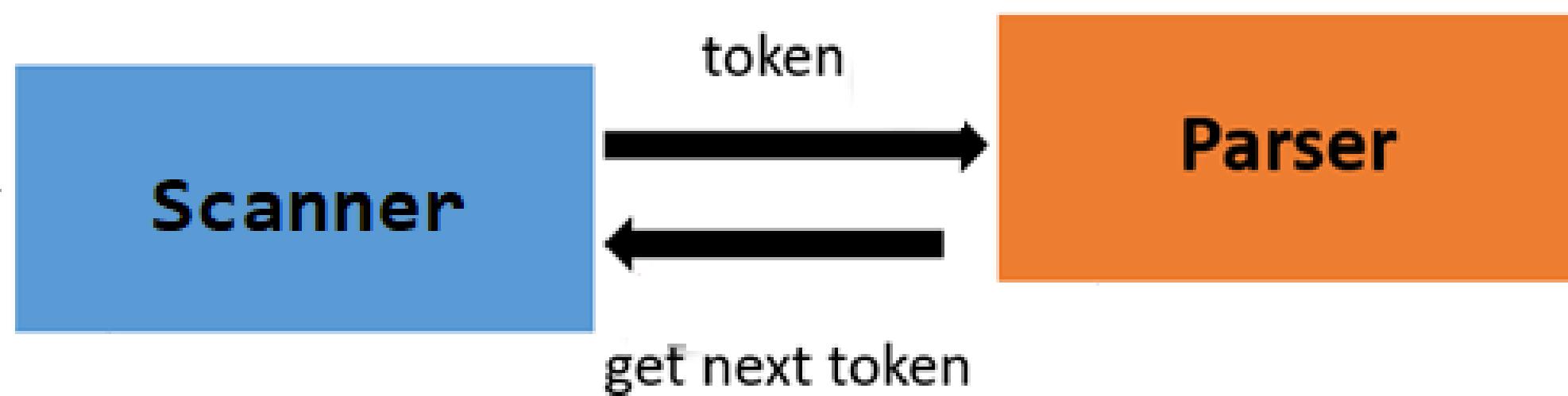


GetToken ()

- Regresa el siguiente token
- Un **token** representa un conjunto de hileras equivalentes sintácticamente
- Puede ser que sean diferente léxica y semánticamente, pero para el parser son equivalentes

Los tokens pueden ser un conjunto:

- De un solo elemento:
 - LEFT_PARENTHESIS { () }
- De una cantidad pequeña de elementos:
 - TYPE { int, float, char }
- Potencialmente infinito:
 - números, hileras, etc.



Recuerden: Token y Lexema son cosas diferentes

- Un **token** es una categoría léxica (nombre de un conjunto)
- Un **lexema** es la hilera particular en el programa que corresponde al token reportado por el scanner.

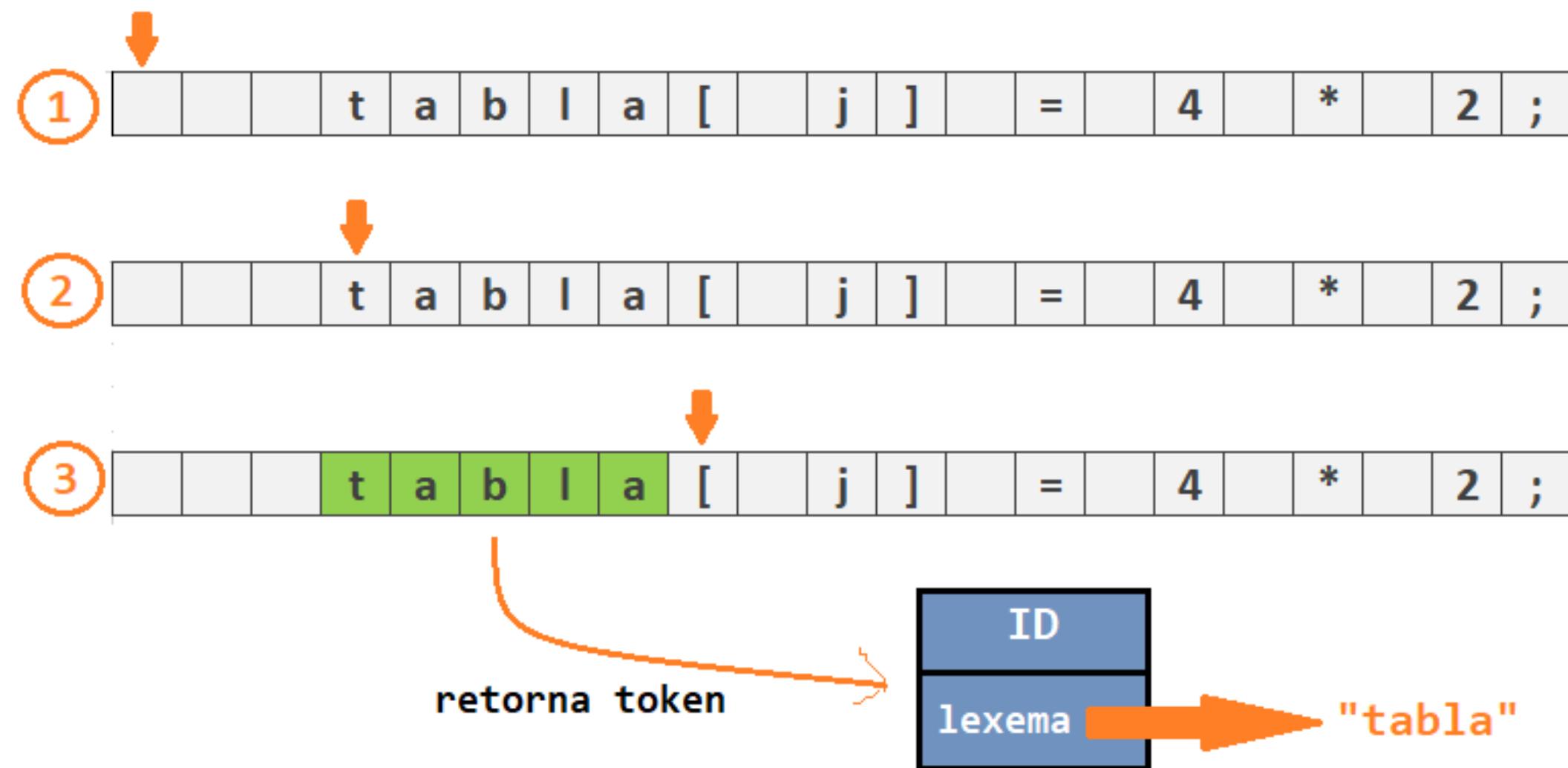


Relación de 1 a N, “un Token tiene muchos lexemas”



GetToken() en ejecución

- El programa completo es una **hilera** en memoria.
- El scanner mantiene una posición actual
- Se salta los Whitespace y comentarios
 - espacios, saltos de línea, tabs, etc.
- Avanza hasta reconocer un token completo



Scanner y “espacios”

- Usualmente no hay un token que represente “espacio”
- En casi todos los lenguajes se ignoran (en Python sí son importantes)

La idea de Backus al hacer Fortran

- Backus intentó **eliminar** los espacios del fuente antes de compilar pero, ¿fue buena idea?



Instrucción DO en FORTRAN

- Permite hacer ciclos sobre una secuencia de instrucciones
- Altera una variable desde un valor inicial hasta un valor final sumando un incremento en cada iteración

Ejemplo

```
K = 0
```

```
DO 10 J = 1,10,1
```

```
K = K + J
```

```
10 CONTINUE
```

- **10** es una etiqueta que debe estar abajo del ciclo
- Vemos que el bloque del **DO** tiene una única instrucción, pero puede tener más
- Realiza las instrucciones y cuando encuentra la etiqueta 10, se regresa al DO
- **J**: Comienza en 1, hasta 10, incrementando de uno en uno

Veamos lo que intentó Backus...

Instrucción DO en FORTRAN

Backus hizo que el compilador eliminara todos los blancos previamente, quedando algo así:

DO 10 J = 1,10,1



D | O | 1 | 0 | J | = | 1 | , | 10 | , | 1 |

¿Qué pasa si le pedimos el siguiente token al scanner?

D | 0 | 1 | 0 | J | = | 1 | , | 10 | , | 1 |

D | 0 | 1 | 0 | J | = | 1 | , | 10 | , | 1 |

¿es un DO?

D | 0 | 1 | 0 | J | = | 1 | , | 10 | , | 1 |

¿o es la variable
DO10J?

**No se sabe. Hay que avanzar
hasta que se
elimine la ambigüedad...**

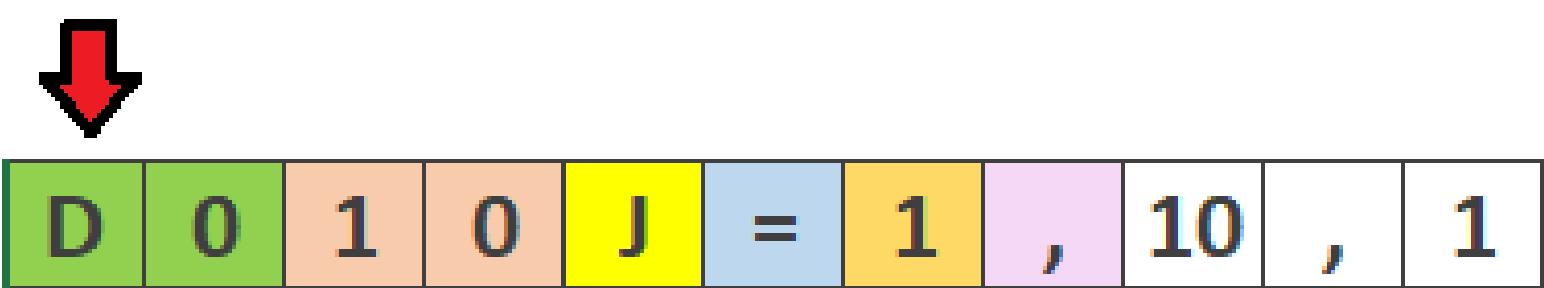


Instrucción DO en FORTRAN

Al llegar al **=**, el compilador aún no sabe si se trata de un **DO** o es una asignación a la variable **DO10J**



Es hasta que llega a la primera coma que se da cuenta que es **DO**, por lo que se tiene que devolver a reportar los tokens anteriores, que son **6 en total** hasta ese punto.

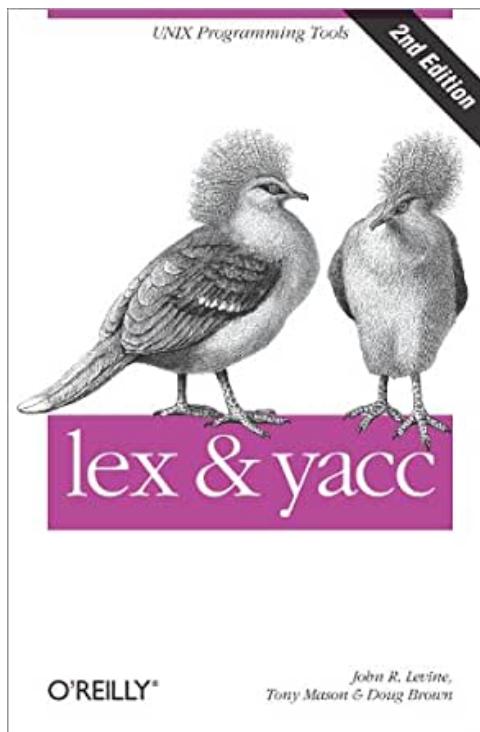


Entonces, aprendimos que eliminarlos fue una mala idea. Aunque no sean tokens, son buenos separadores. Los espacios son amigos del compilador.

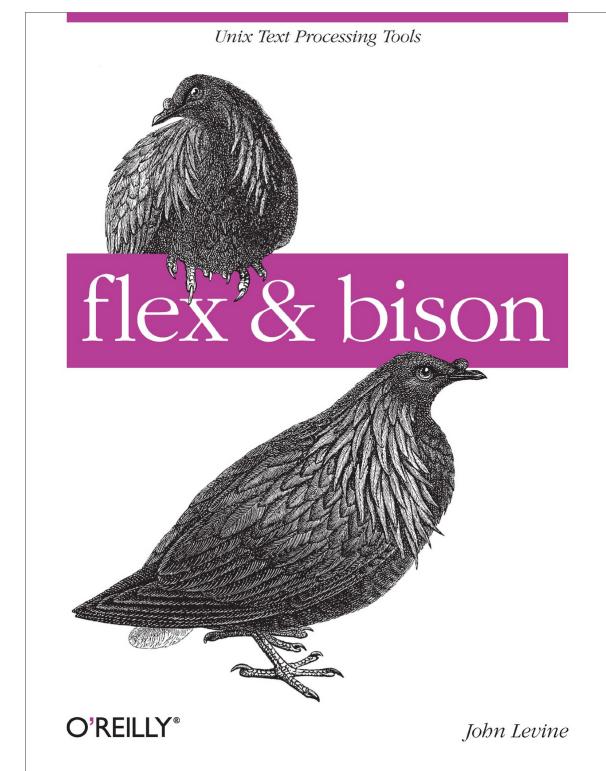


Generadores de Scanners

- Hay mucha teoría y experiencia acumulada respecto al Análisis Léxico
- La mayor parte se puede automatizar
- Existe software que genera casi un scanner completo
 - Ejemplo: lex, flex
- Generar un scanner es relativamente fácil con estas herramientas
- Siempre es posible (o necesario) agregar un poco de código a mano



*** *lex fue luego remplazado por flex, la f es de "fast". yacc es un generador de parsers, que luego fue remplazado por bison*



Resumiendo

Detección de Tokens

- Es la función principal del scanner
- Avanzar desde la entrada hasta reconocer el token más grande posible

Entonces, ¿Cómo se hace?

- Son técnicas derivadas de *Lenguajes formales*, Teoría de Autómatas, Autómatas Determinísticos de Estados Finitos



Lenguajes Formales

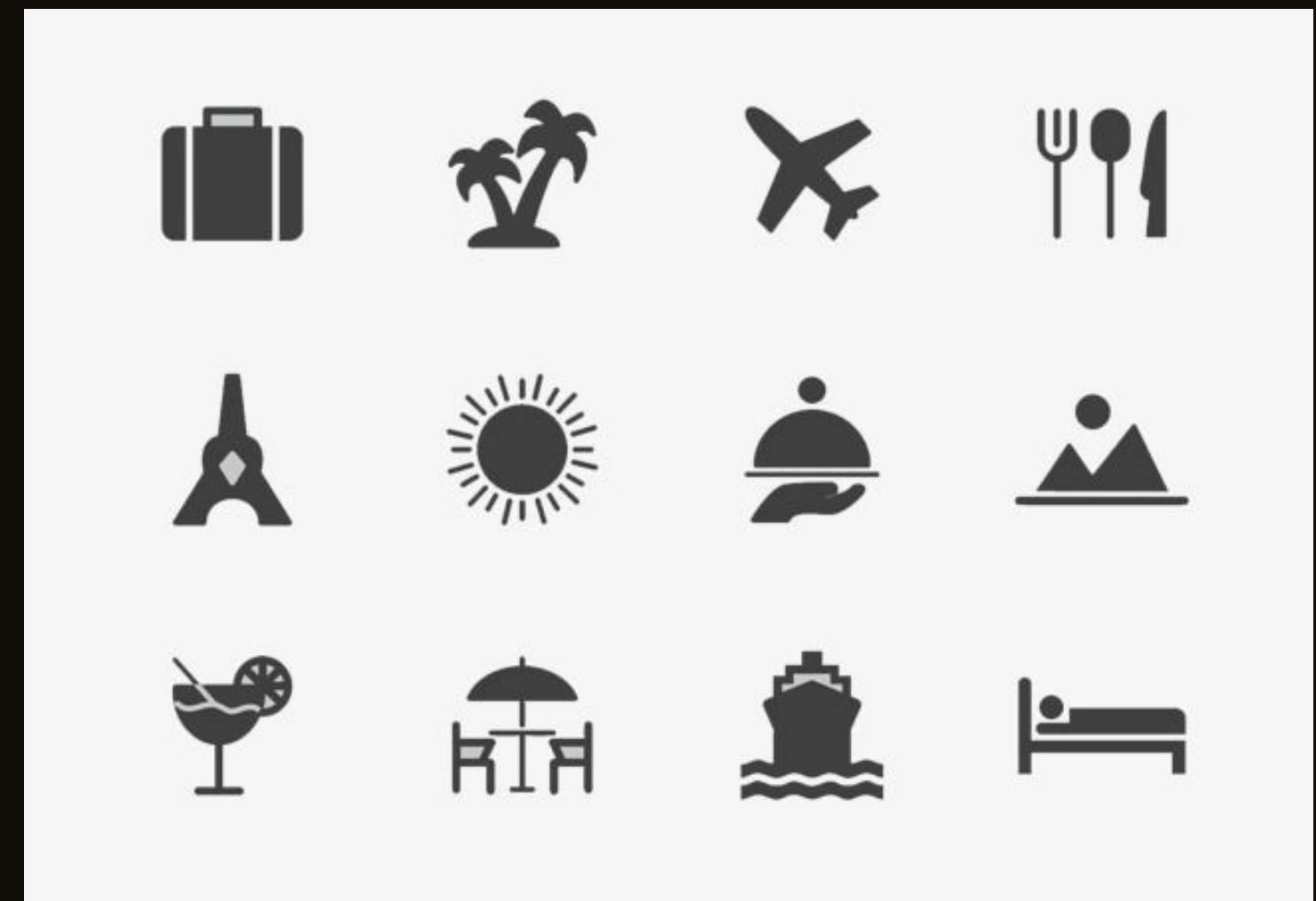
Algunas definiciones de Lenguaje

- Habilidad humana para adquirir y usar complejos sistemas de comunicación. El estudio científico del lenguaje se llama lingüística.
- Conjunto de sonidos articulados con el hombre manifiesta lo que piensa o siente. Manera de expresarse. Conjunto de señales que dan a entender algo.
- Ejemplos:
 - Lenguaje natural: español, inglés, alemán
 - Lenguaje de computadora: C, Java, ensamblador, etc.
 - Lenguaje matemático
 - **Lenguajes Formales**



Símbolo

- Es un concepto **primitivo**
- Representación de un concepto, idea o entidad, con un significado convencional (resultado de un acuerdo)
- No necesariamente lo mismo que un carácter
- En matemáticas frecuentemente denotados como a, b, c



Alfabeto

- Def: Conjunto finito y no vacío de símbolos
- Usualmente denotado como Σ
- Ejemplos:
 - $\Sigma = \{a, b, c, d, \dots, x, y, z\}$
 - $\Sigma = \{0, 1\}$
 - $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
 - $\Sigma = \{A, T, C, G\}$
 - $\Sigma = \{\text{Ala, Arg, Asn, Asp, Cys, Gln, \dots, Trp, Tyr, Val}\}$ *** aminoácidos ***



Hilera sobre Σ

- Def: **Secuencia** de longitud arbitraria formada con símbolos tomados de un alfabeto Σ
- Puede repetirse símbolos
- El orden de la secuencia es importante
- Denotadas usualmente como w, x, y, z
- También conocidas como "palabras" o "frases"

Ejemplo:

Sea $\Sigma = \{0, 1\}$, posibles hileras pueden ser:

- 10101011
-
- 1111
- 1010111101011

Ejemplo:

Sea $\Sigma = \{A, T, C, G\}$

- ATTCAACAGAGA
-
- CGCG
- TTCGA

¡Hileras vacías son válidas!



¿Diferencia entre conjunto y secuencia?

- 1) Una secuencia tiene un orden, un conjunto no.
- 2) En los conjuntos no hay elementos repetidos, en las secuencias sí.

Longitud de una hilera

Def: Sea w una hilera sobre Σ :

- $|w|$ es la longitud de la hilera w (cantidad de símbolos)
 - $|\text{AAAATCATATATA}| = 13$
 - $|00| = 0$
 - $|\text{GUAGUAGUAGUAGUAGUA}| = 21$
- La longitud puede ser 0
 - Hilera vacía
 - Se representa por ε (épsilon) o λ (lambda)
 - $|\varepsilon| = 0$

Prefijos de una hilera

Sea w una hilera sobre Σ :

- Un **prefijo de w** es una hilera formada tomando en el mismo orden los primeros k símbolos desde la izquierda de w donde:
 - $0 \leq k \leq |w|$ (no negativo y menor o igual al tamaño de la hilera)
- Si $k < |w|$ tenemos un **prefijo propio**
- Si $k = |w|$ tenemos un **prefijo no propio**
- Si $k = 0$ tenemos un prefijo vacío
- ϵ es prefijo de cualquier hilera

Ejemplo de prefijos:

Sea $\Sigma = \{A, T, C, G\}$

Sea $w = TTGACACTGCAA$ una hilera sobre Σ :

- TTGACA
- TTGACACTGCAA *(Prefijo no propio!)*
- TTGAC
- T
- ϵ

Sufijos de una hilera

Sea w una hilera sobre Σ :

- Un **sufijo de w** es una hilera formada tomando en el mismo orden los primeros k símbolos desde la derecha de w donde:
 - $0 \leq k \leq |w|$
- Si $k < |w|$ tenemos un **sufijo propio**
- Si $k = |w|$ tenemos un **sufijo no propio**
- Si $k = 0$ tenemos un sufijo vacío
- ϵ es sufijo de cualquier hilera

Ejemplo de sufijos:

Sea $\Sigma = \{A, T, C, G\}$

Sea $w = TTGACACTGCAA$ una hilera sobre Σ :

- GCAA
- TTGACACTGCAA *¡Sufijo no propio!*
- ACTGCAA
- TGCAA
- ϵ

Subhilera de una hilera

Sea w una hilera sobre Σ :

- Una **subhilera de w** es una hilera formada tomando en el mismo orden k símbolos consecutivos de w a partir de una posición j
 - $0 \leq k, j \leq |w|$
 - $0 \leq (k+j) \leq |w|$
- Si $k+j < |w|$ y $j > 0$ tenemos una **subhilera propia**
- Si $k = 0$ tenemos una subhilera vacía
- ϵ es subhilera de cualquier hilera

Ejemplo de subhileras:

Sea $\Sigma = \{A, T, C, G\}$

Sea $w = TTGACACTGCAA$ una hilera sobre Σ :

- GCAA
- TTGACACTGCAA *(Subhilera no propia!)*
- CACT
- A
- ϵ

Concatenación de hileras

Sean v y w dos hilera sobre Σ :

- Si:

$$v = a_1 a_2 a_3 \dots a_k$$

$$w = b_1 b_2 b_3 \dots b_n$$

- Entonces, la concatenación de v y w es la hilera:

$$a_1 a_2 a_3 \dots a_k b_1 b_2 b_3 \dots b_n$$

- Se denota como vw (una después de la otra)

Ejemplo

Sea $\Sigma = \{0, 1\}$

Consideremos las siguientes hileras sobre Σ :

$$v = 11$$

$$w = 01000$$

$$x = 1101$$

$$y = 011101110$$

Entonces:

$$vw = 1101000$$

$$xy = 1101011101110$$

$$xv = 1101$$

$$wx = 010001101$$

$$yy = 011101110011101110$$

Propiedades de la Concatenación de hileras

- Esasociativa
 $\mathbf{vwx} = (\mathbf{v}\mathbf{w})\mathbf{x} = \mathbf{v}(\mathbf{w}\mathbf{x})$
- En general, no commutativa
 $\mathbf{vw} \neq \mathbf{wv}$
- Longitudes se suman
 $|\mathbf{vw}| = |\mathbf{v}| + |\mathbf{w}|$
- La hilera ϵ es el elemento neutro de la concatenación
 $\mathbf{v}\epsilon = \epsilon\mathbf{v} = \mathbf{v}$

Potencia n-ésima de una hilera

Sea v una hilera sobre Σ :

- La **potencia n-ésima** de v es la hilera resultado de concatenar n copias de v
- Se denota como v^n
- Casos especiales:

$$v^1 = v$$

$$v^0 = \epsilon$$



Ejemplos de Potencia n-ésima de una hilera

Sea $v = \{\text{ATTAC}\}$

$$v^1 = \text{ATTAC}$$

$$v^2 = \text{ATTACATTAC}$$

$$v^5 = \text{ATTACATTACATTACATTACATTAC}$$

1 2 3 4 5

$$v^6 = \text{ATTACATTACATTACATTACATTAC}$$

$$v^0 = \epsilon$$



Apuntes de 30/09 y 02/10

Apuntadora: Jazmine Espinoza Palma

Del miércoles:

Alfabeto: Conjunto finito, no vacío de símbolos

Sea Σ un alfabeto, entonces Σ^k es el conjunto de todas las hileras sobre Σ que tengan longitud k , con $k \geq 0$ y entero



Multiplicación de alfabetos

Para $\Sigma = \{0,1\} \rightarrow \Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

Para $\Sigma = \{0,1\} \rightarrow \Sigma^2 = \{00, 01, 10, 11\}$

Para $\Sigma = \{0,1\} \rightarrow \Sigma^1 = \{0, 1\}$

••• ¿Será Σ igual a Σ^1 ?

Para $\Sigma = \{0,1\} \rightarrow \Sigma^0 = \{\varepsilon\}$

(El conjunto no es vacío, tiene un elemento, que es vacío).

NO.

$\Sigma^1 = \{0, 1\}$ es un conjunto de hileras

$\Sigma = \{0, 1\}$ es un conjunto de símbolos



Conjunto Σ^*

Sea Σ un alfabeto, Σ^* se define recursivamente como:

1. $\epsilon \in \Sigma^*$
2. Si $w \in \Sigma^*$ y $a \in \Sigma$, entonces $wa \in \Sigma^*$
3. $w \in \Sigma^*$ solo si puede ser construida desde ϵ usando el paso 2 repetidamente

$$\Sigma^* = \bigcup_{i=0}^k \Sigma^i \text{ para } k \geq 0 \text{ arbitrariamente grande (cuanto yo quiera)}$$

TODAS LAS HILERAS QUE PUEDO FORMAR CON ESTE ALFABETO

Todavía nos queda un monstruo peludo por conocer:

Stephen Cole Kleene



A Σ^* se le llama Cierre de Kleene

Kleeneamos el alfabeto 😊

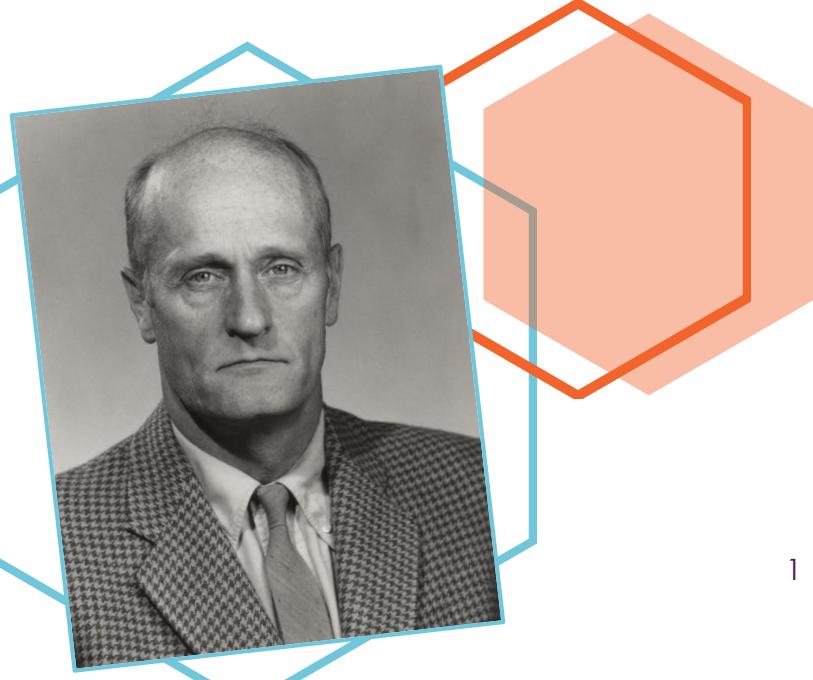
- Matemático de USA (1909 -1994)
- Estudio con Alonzo Church (El del Cálculo lambda)
- Uno de los creadores de “Lenguajes Formales”
- Inventa expresiones regulares
-

Para $\Sigma = \{1,0\}$

→ $\{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, 100000, \dots\}$

Son miembros de Σ^*

Es decir, Σ^* es un conjunto infinito



Conjunto Σ^t

Sea Σ un alfabeto, Σ^t se define recursivamente como:

1. Si $a \in \Sigma^t$ entonces $a \in \Sigma$
2. Si $w \in \Sigma^t$ y $a \in \Sigma$, entonces $wa \in \Sigma^t$
3. $w \in \Sigma^t$ solo si puede ser construida desde algun elemento de Σ usando el paso 2 repetidamente

$$\Sigma^t = \bigcup_{i=1}^k \Sigma^i \text{ para } k \geq 1 \text{ arbitrariamente grande (cuanto yo quiera)}$$

Para $\Sigma = \{1, 0\}$

→ $\{0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, 100000, \dots\}$

Son miembros de Σ^t

Es decir, Σ^t es un conjunto infinito



ES LO MISMO



SOLO QUE DIFERENTE

Reverso de una Hilera

Sea $w \in \Sigma^*$, la reversa de w , denotada como w^R o w^{-1} es la hilera w , pero escrita al revés.

$$w = aaabbac \rightarrow w^R = cabbaaa$$

Teorema:

Sean $u, v \in \Sigma^*$.

Entonces $(uv)^R = v^R u^R$

Ejemplo:

$$u = abc, v = def$$

$$(uv)^R = fedcba \rightarrow v^R u^R = fedcba$$

Σ^* contiene a ϵ
(empieza desde $i=0$)
y Σ^t no lo contiene
(empieza desde $i=1$)

...el profe se crasheo



Y como no volvió... nos pusimos a jugar 😊

PERO... TOCÓ QUIZ EL VIERNES



QUIZ 3

1. Explique las complicaciones que le trajo al compilador de FORTRAN la idea de ignorar los espacios en blanco.
2. Muestre todos los prefijos, sufijos y subhileras de la hilera “itcr”.
3. Defina detalladamente los siguientes conceptos:
 - a. Preprocesador
 - b. Linking dinámico
 - c. Loader
 - d. Profiler
 - e. Debugger
 - f. Análisis Sintáctico
 - g. Front end

Conceptos repetidos del quiz anterior:

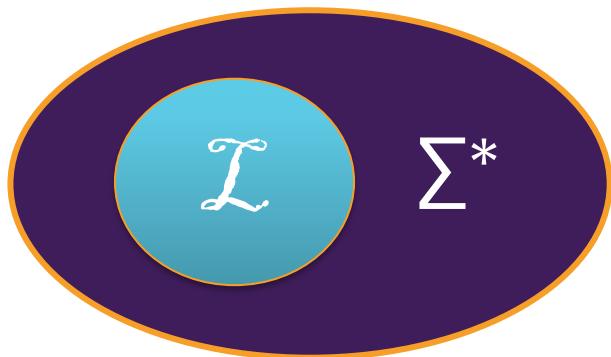


● LENGUAJES FORMALES

(Aún continuamos en el análisis léxico)

Sea Σ un alfabeto, entonces un lenguaje formal \mathcal{L} sobre Σ es:

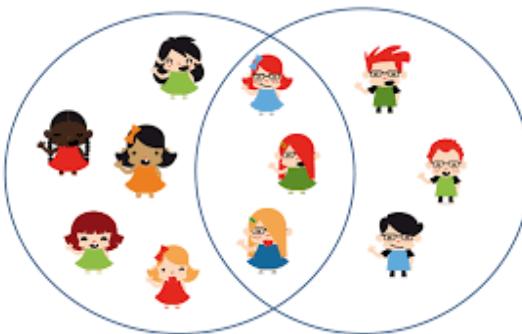
1. Un conjunto de hileras
2. \mathcal{L} es subconjunto de Σ^*
3. Las hileras cumplen cierta propiedad especificada
4. Hay lenguajes finitos, infinitos y vacíos



Ejemplos:

- Sea $\Sigma = \{0, 1\}$
 $\mathcal{L} =$ hileras sobre Σ que terminen en 00
 $\mathcal{L} =$ hileras sobre Σ que tengan longitud impar
- $\Sigma = \{A, U, C, G\}$
 $\mathcal{L} =$ hileras sobre Σ que no incluyan la subhiera UUU
 $\mathcal{L} =$ hileras sobre Σ cuya longitud sea múltiplo de 3

Gran parte de las operaciones sobre lenguajes, vienen heredadas de los conjuntos
(Teoría de conjuntos)



Mejor haciendo
tablita de contenidos
para no perdernos

• • •

TABLA CONTENIDOS:

Lenguaje Formal 

Operaciones sobre
lenguajes formales

- Unión
- Intersección
- Diferencia
- Complemento
- Inverso
- Concatenación
- Multiplicación o potencia

Lenguajes formales y
mecanismos

- Autómatas

Generalidades

• OPERACIONES SOBRE LENGUAJES FORMALES

Unión:

Sean \mathcal{L} y \mathcal{M} dos lenguajes sobre Σ :

La unión de \mathcal{L} y \mathcal{M} , denominada como $\mathcal{L} \cup \mathcal{M}$ es:

$$\mathcal{L} \cup \mathcal{M} = \{x \mid x \in \mathcal{L} \text{ o } x \in \mathcal{M}\}$$

($\mathcal{L} \cup \mathcal{M}$ es el conjunto de las hileras x tales que, x pertenezca a \mathcal{L} , o x pertenezca a \mathcal{M})

Denominado también como $\mathcal{L} + \mathcal{M}$

Ejemplos:

- Sean $\mathcal{L} = \{000, 010, 100, 110\}$ y $\mathcal{M} = \{0, 1, 11\}$

$$\mathcal{L} \cup \mathcal{M} = \{000, 010, 100, 110, 0, 1, 11\}$$

- \mathcal{L} = hileras sobre Σ que terminen en 00

\mathcal{M} = hileras sobre Σ que empiecen con 1

$\mathcal{L} \cup \mathcal{M}$ = hileras sobre Σ que empiecen con 1 o que terminen en 00

$$\mathcal{L} \cup \mathcal{M} = \{1, 00, 100, 000, 1111, \dots\}$$

En los lenguajes no pueden haber repetidos, y el orden no es importante

• TABLA CONTENIDOS:

Lenguaje Formal

Operaciones sobre lenguajes formales

- Unión
- Intersección
- Diferencia
- Complemento
- Inverso
- Concatenación
- Multiplicación o potencia

Lenguajes formales y mecanismos

- Autómatas

Generalidades

• OPERACIONES SOBRE LENGUAJES FORMALES

• • •

Unión:

Propiedades:

- Comutatividad:

$$\mathcal{L} \cup \mathcal{M} = \mathcal{M} \cup \mathcal{L}$$

- Asociatividad:

$$\mathcal{L} \cup \mathcal{M} \cup \mathcal{Q} = (\mathcal{L} \cup \mathcal{M}) \cup \mathcal{Q} = \mathcal{L} \cup (\mathcal{M} \cup \mathcal{Q})$$

- Idempotencia:

$$\mathcal{L} \cup \mathcal{L} = \mathcal{L}$$

- Elemento neutro:

$$\exists \mathcal{M} \text{ t.q. } \forall \mathcal{L}, \mathcal{L} \cup \mathcal{M} = \mathcal{L} \rightarrow \mathcal{M} = \emptyset$$

- Cierre:

Si $\mathcal{L}, \mathcal{M} \subseteq \Sigma^*$ entonces $\mathcal{L} \cup \mathcal{M} \subseteq \Sigma^*$

(La unión de estos lenguajes siempre dará algo que pertenezca a Σ^*)

Intersección:

Sean \mathcal{L} y \mathcal{M} dos lenguajes sobre Σ :

La intersección de \mathcal{L} y \mathcal{M} , denominada como $\mathcal{L} \cap \mathcal{M}$ es:

$$\mathcal{L} \cap \mathcal{M} = \{x \mid x \in \mathcal{L} \text{ y } x \in \mathcal{M}\}$$

($\mathcal{L} \cap \mathcal{M}$ es el conjunto de las hileras x tales que, x pertenezca a \mathcal{L} , y x pertenezca a \mathcal{M})

La diferencia con la unión es que aquí ambos lenguajes deben estar en los conjuntos, en la unión basta con que esté en 1.

• • •

TABLA CONTENIDOS:

Lenguaje Formal

Operaciones sobre lenguajes formales

- Unión 
- Intersección
- Diferencia
- Complemento
- Inverso
- Concatenación
- Multiplicación o potencia

Lenguajes formales y mecanismos

- Autómatas

Generalidades

• • •

• OPERACIONES SOBRE LENGUAJES FORMALES

Ejemplos:

- Sean $\mathcal{L} = \{000, 010, 100, 110\}$ y $\mathcal{M} = \{0, 1, 11\}$
 $\mathcal{L} \cap \mathcal{M} = \emptyset \quad \rightarrow \text{Vacío es un lenguaje}$
- $\mathcal{L} =$ hileras sobre $\Sigma = \{0, 1\}$ que terminen en 00
 $\mathcal{M} =$ hileras sobre $\Sigma = \{0, 1\}$ que empiecen con 1

$\mathcal{L} \cap \mathcal{M} =$ hileras sobre Σ que empiecen con 1 y que terminen en 00
 $\mathcal{L} \cap \mathcal{M} = \{100, 1000, 10000, 11000, 111100, \dots\}$

Propiedades:

- Commutatividad:
 $\mathcal{L} \cap \mathcal{M} = \mathcal{M} \cap \mathcal{L}$
- Asociatividad:
 $\mathcal{L} \cap \mathcal{M} \cap \mathcal{G} = (\mathcal{L} \cap \mathcal{M}) \cap \mathcal{G} = \mathcal{L} \cap (\mathcal{M} \cap \mathcal{G})$

- Idempotencia:
 $\mathcal{L} \cap \mathcal{L} = \mathcal{L}$

- $\mathcal{L} \cap \emptyset = \emptyset$

- Elemento neutro:

$$\exists \mathcal{M} \text{ t.q. } \forall \mathcal{L}, \mathcal{L} \cap \mathcal{M} = \mathcal{L} \quad \rightarrow \mathcal{M} = \Sigma^*$$

- Cierre:

$$\text{Si } \mathcal{L}, \mathcal{M} \subseteq \Sigma^* \text{ entonces } \mathcal{L} \cap \mathcal{M} \subseteq \Sigma^*$$

(La intersección de estos lenguajes siempre dará algo que pertenezca a Σ^*)

La que era para quiz XD

• • •

TABLA CONTENIDOS:

Lenguaje Formal

Operaciones sobre lenguajes formales

- Unión
- Intersección 
- Diferencia
- Complemento
- Inverso
- Concatenación
- Multiplicación o potencia

Lenguajes formales y mecanismos

- Autómatas

Generalidades

• OPERACIONES SOBRE LENGUAJES FORMALES

• • •

Diferencia:

Sean \mathcal{L} y \mathcal{M} dos lenguajes sobre Σ :

La diferencia de \mathcal{L} y \mathcal{M} , denominada como $\mathcal{L} - \mathcal{M}$ es:

$$\mathcal{L} - \mathcal{M} = \{x \mid x \in \mathcal{L} \text{ y } x \notin \mathcal{M}\}$$

($\mathcal{L} - \mathcal{M}$ es el conjunto de los x tales que, x está en \mathcal{L} , pero x no puede estar en \mathcal{M})

Ejemplos:

- Sean $\mathcal{L} = \{1, 11, 111, 1111, 11111\}$ y $\mathcal{M} = \{0, 1, 11\}$

$$\mathcal{L} - \mathcal{M} = \{111, 1111, 11111\}$$

- \mathcal{L} = hileras sobre $\Sigma = \{0, 1\}$ que terminen en 00

\mathcal{M} = hileras sobre $\Sigma = \{0, 1\}$ que empiecen con 1

$\mathcal{L} - \mathcal{M}$ = hileras sobre Σ que terminan en 00 pero que no empiecen con 1

$$\mathcal{L} - \mathcal{M} = \{00, 000, 0100, 0000, 00100, \dots\}$$

$$\mathcal{M} - \mathcal{L} = \{1010, 10001, 101101, 100111, 1001001, \dots\}$$

Es decir, $\mathcal{L} - \mathcal{M}$ no es lo mismo que $\mathcal{M} - \mathcal{L}$

Propiedades:

- NO es commutativa, NO es asociativa, NO es idempotente.

- Elemento neutro:

$$\exists \mathcal{M} \text{ t.q. } \forall \mathcal{L}, \mathcal{L} - \mathcal{M} = \mathcal{L} \rightarrow \mathcal{M} = \emptyset$$

- Cierre:

Si $\mathcal{L}, \mathcal{M} \subseteq \Sigma^*$ entonces $\mathcal{L} - \mathcal{M} \subseteq \Sigma^*$

(Seguimos sin podernos escapar de Σ^*)

• • • TABLA CONTENIDOS:

Lenguaje Formal

Operaciones sobre lenguajes formales

- Unión
- Intersección
- Diferencia** 
- Complemento
- Inverso
- Concatenación
- Multiplicación o potencia

Lenguajes formales y mecanismos

- Autómatas

Generalidades

• • •

• OPERACIONES SOBRE LENGUAJES FORMALES

Complemento:

Sean \mathcal{L} un lenguaje sobre Σ :

El complemento de \mathcal{L} , denominada como $\overline{\mathcal{L}}$ o \mathcal{L}' es:

$$\mathcal{L}' = \{x \mid x \in \Sigma^* \wedge x \notin \mathcal{L}\}$$

(x debe pertenecer a Σ^* y no puede pertenecer a \mathcal{L})

Ejemplos:

- Sean $\mathcal{L} = \{000, 010, 100, 110\}$ un lenguaje sobre $\Sigma = \{0, 1\}$
 $\overline{\mathcal{L}} = \Sigma^* - \{000, 010, 100, 110\}$
- \mathcal{L} = hileras sobre $\Sigma = \{0, 1\}$ que terminan en 00
 $\overline{\mathcal{L}}$ = hileras sobre Σ que **no** terminan en 00

Propiedades:

- $\overline{\Sigma^*} = \emptyset$
- $\overline{\overline{\mathcal{L}}} = \mathcal{L}$
- Cierre:
Si $\mathcal{L} \subseteq \Sigma^*$ entonces $\overline{\mathcal{L}} \subseteq \Sigma^*$

Inverso:

Sean \mathcal{L} un lenguaje sobre Σ :

El inverso o el reflejo de \mathcal{L} , denominada como \mathcal{L}^{-1} o \mathcal{L}^R es:

$$\mathcal{L}^{-1} = \{x^{-1} \mid x \in \mathcal{L}\}$$

(darles vuelta a las hileras que están en \mathcal{L} y se las echamos a \mathcal{L}^{-1})

• • •

TABLA CONTENIDOS:

Lenguaje Formal

Operaciones sobre lenguajes formales

- Unión
- Intersección
- Diferencia
- Complemento** 
- Inverso
- Concatenación
- Multiplicación o potencia

Lenguajes formales y mecanismos

- Autómatas

Generalidades

• • •

• OPERACIONES SOBRE LENGUAJES FORMALES

Ejemplos:

- Sean $\mathcal{L} = \{000, 010, 100, 110\}$ un lenguaje sobre $\Sigma = \{0, 1\}$
 $\mathcal{L}^{-1} = \{000, 010, 001, 011\}$
- $\mathcal{L} = \{\text{esteban, tomas, Alejandro, amanda}\}$
 $\mathcal{L}^{-1} = \{\text{nabetse, samot, ordnajela, adnama}\}$

Propiedades:

- NO es idempotente
- Cierre:
Si $\mathcal{L} \subseteq \Sigma^*$ entonces $\mathcal{L}^{-1} \subseteq \Sigma^*$

Concatenación:

Sean \mathcal{L} y \mathcal{M} dos lenguajes sobre Σ :

La concatenación de \mathcal{L} y \mathcal{M} , denotada como \mathcal{LM} es:

$$\mathcal{LM} = \{xw \mid x \in \mathcal{L} \text{ y } w \in \mathcal{M}\}$$

(Lenguaje formado por hileras creadas al concatenar una hilera de \mathcal{L} con una hilera de \mathcal{M})

Ejemplos:

- Sean $\mathcal{L} = \{000, 010, 100, 110\}$ y $\mathcal{M} = \{0, 1, 11\}$

$$\mathcal{LM} = \{0000, 0001, 00011, 0100, 0101, 01011, 1000, 1001, 10011, 1100, 1101, 11011\}$$

- \mathcal{L} = hileras sobre $\Sigma = \{0, 1\}$ que terminen en 00
 \mathcal{M} = hileras sobre $\Sigma = \{0, 1\}$ que empiecen con 1

\mathcal{LM} = hileras de la forma ...001...
(Cualquier hilera que contenga la subhilera 001)

• • •

TABLA CONTENIDOS:

Lenguaje Formal

Operaciones sobre lenguajes formales

- Unión
- Intersección
- Diferencia
- Complemento
- Inverso
- Concatenación
- Multiplicación o potencia



Lenguajes formales y mecanismos

- Autómatas

Generalidades

• OPERACIONES SOBRE LENGUAJES FORMALES

Propiedades:

- NO es conmutativa, NO es idempotente
- Asociatividad:

$$\mathcal{L}\mathcal{M}\mathcal{Q} = (\mathcal{L}\mathcal{M})\mathcal{Q} = \mathcal{L}(\mathcal{M}\mathcal{Q})$$
- Elemento neutro:

$$\exists \mathcal{M} \text{ t.q. } \forall \mathcal{L}, \mathcal{L}\mathcal{M} = \mathcal{M}\mathcal{L} = \mathcal{L} \quad \rightarrow \mathcal{M} = \{\epsilon\}$$

Multiplicación o Potencia:

Sea L un lenguaje, entonces:

- $L^1 = L$
- $L^2 = LL$
- $L^3 = LLL$
- $L^4 = LLLL$

No puede repetirse ni faltar ninguno



Ejemplos:

- Sea $L = \{000, 010, 100, 110\}$
 $L^2 = \{000000, 000010, 000100, 000110, 010000, 010010, 010100, 010110, 100000, 100010, 100100, 100110, 110000, 110010, 110100, 110110\}$
- Sea $M = \{000, 010, 100, 110, \epsilon\}$
 $M^2 = \{000000, 000010, 000100, 000110, 010000, 010010, 010100, 010110, 100000, 100010, 100100, 100110, 110000, 110010, 110100, 110110, 000, 010, 100, 110, \epsilon\}$

Propiedades:

- $L^K = L^{K-1}L = L L^{K-1}$, con $K \geq 0$ entero
- $L^0 = \{\epsilon\}$
- Cierre:
Si $\mathcal{L} \subseteq \Sigma^*$ entonces $\mathcal{L}^K \subseteq \Sigma^*$

• TABLA CONTENIDOS:

Lenguaje Formal

Operaciones sobre lenguajes formales

- Unión
- Intersección
- Diferencia
- Complemento
- Inverso
- Concatenación
- Multiplicación o potencia

Lenguajes formales y mecanismos

- Autómatas

Generalidades

• • •

• LENGUAJE L*

(Aún continuamos en el análisis léxico)

Sea L un lenguaje, el lenguaje L* se define como:

$$L^* = \bigcup_{i=0}^K L^i \text{ para } k \geq 0 \text{ arbitrariamente grande (cuanto yo quiera)}$$



• LENGUAJE L†

Sea L un lenguaje, el lenguaje L† se define como:

$$L^\dagger = \bigcup_{i=1}^K L^i \text{ para } k \geq 1 \text{ arbitrariamente grande (cuanto yo quiera)}$$

puede haber un ε solo porque L ya lo contenga

• Lenguajes formales y mecanismos

Mecanismos asociados a un lenguaje L, que nos interesan:

- **Generador de L**

- Genera todas las hileras de L.
- No genera hileras que NO pertenezcan a L.



Genera hileras del lenguaje

- **Reconocedor de L**

- Mecanismo binario que acepta o rechaza hileras.
- Solo aceptará hileras de L
- Nunca rechazará hileras que pertenezcan a L



Es hilera del lenguaje
No es hilera del lenguaje

• • •

TABLA CONTENIDOS:

Lenguaje Formal

Operaciones sobre lenguajes formales

- Unión
- Intersección
- Diferencia
- Complemento
- Inverso
- Concatenación
- Multiplicación o potencia

Lenguajes formales y mecanismos



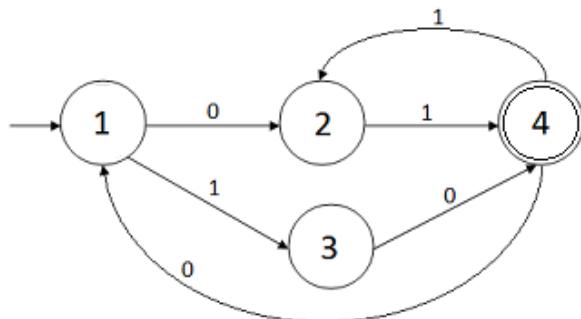
- Autómatas

Generalidades

• AUTÓMATAS DETERMINÍSTICOS DE ESTADOS FINITOS

En el mundo del scanner todavía

- Autómatas son chunches que nos van a servir para hacer scanner



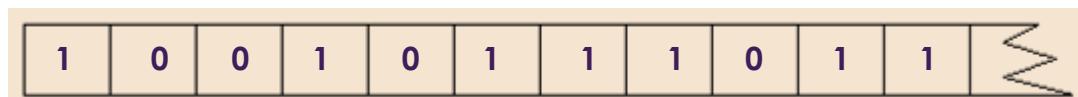
AGAIN... 😞

Estructura:

- Es un grafo
- Los nodos tienen etiqueta (opcional)
- Los arcos están etiquetados
- Los nodos son **estados**
- Los arcos son **transiciones**
- Hay un **estado inicial** (flecha sin origen)

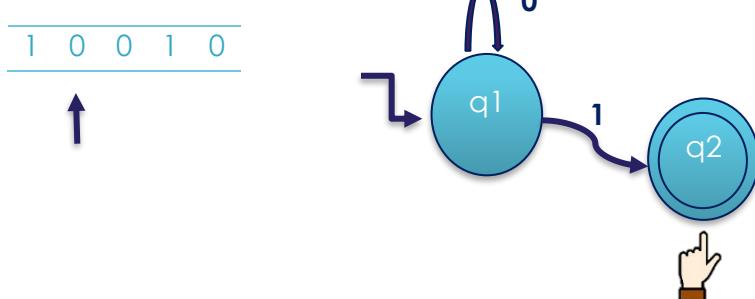
Mecánica:

- Un autómata de este tipo procesa hileras (es un reconocedor)



Resultado: **Aceptado** o **Rechazado**

Siempre hay un **estado actual** del autómata y un **símbolo actual** en la hilera, una vez procesado se toma el **siguiente símbolo**, hasta terminar la hilera, o que en algún punto no haya transición (para lo cual se rechaza la hilera).



• TABLA CONTENIDOS:

Lenguaje Formal

Operaciones sobre lenguajes formales

- Unión
- Intersección
- Diferencia
- Complemento
- Inverso
- Concatenación
- Multiplicación o potencia

Lenguajes formales y mecanismos

- Autómatas

Generalidades

• Generalidades

Sobre el ensayo 04 de Pinker:
Miércoles 07 de octubre

Sobre el ensayo 05 de Pinker:
Miércoles 14 de octubre

Proyecto 1:
Viernes 09 de octubre

Sobre grupos de proyecto:
Enviar grupos de proyecto, datos: nombres integrantes,
correo institucional (estudiantec.cr)



Definición de horarios de consulta (en Teams):

HORAS	LUNES	MARTES	MIÉRCOLES	JUEVES	VIERNES	SÁBADO
7:30-8:00						
8:00-8:30						
8:30-9:00	Preparación de Clases - Revisión de Trabajos - Investigación	Principios de Sistemas Operativos	Compiladores e Intérpretes	Principios de Sistemas Operativos	Compiladores e Intérpretes	Happy Few - Trabajo con testaríos - Investigación
9:00-9:30		Preparación de Clases - Revisión de Trabajos - Investigación	Investigación de Operaciones: SJ06-202	Preparación de Clases - Revisión de Trabajos - Investigación	Investigación de Operaciones: SJ06-202	
9:30-10:00						
10:00-10:30						
10:30-11:00						
11:00-11:30						
11:30-12:00						
12:00-13:00						
13:00-13:30	Consejo de Maestría					
13:30-14:00						
14:00-14:30						
14:30-15:00		CONSULTA		CONSULTA		
15:00-15:30						
15:30-16:00						
16:00-16:30						
16:30-17:00						
17:00-17:30						
17:30-18:00						
18:00-18:30						
18:30-19:00						
19:00-19:30						
19:30-20:00						



14

• TABLA CONTENIDOS:

Lenguaje Formal

Operaciones sobre lenguajes formales

- Unión
- Intersección
- Diferencia
- Complemento
- Inverso
- Concatenación
- Multiplicación o potencia

Lenguajes formales y mecanismos

- Autómatas

Generalidades



Compiladores e intérpretes

Bryan Masís Marín

APUNTES
14/OCTUBRE/2020

CANCELADO EL QUIZ #5

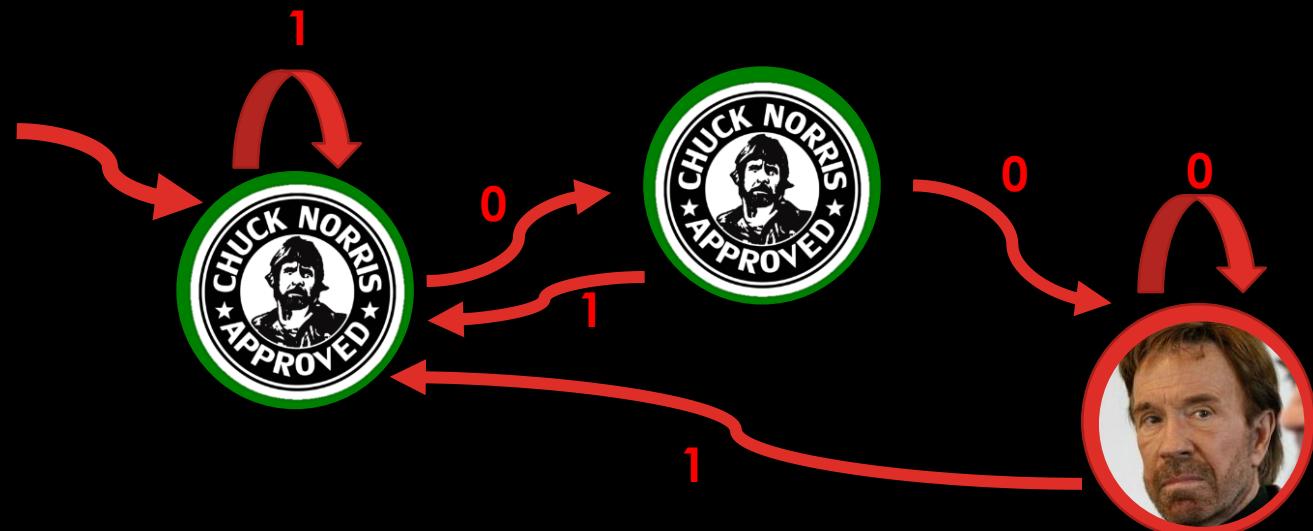
- Envío al correo
torresrojas.cursos.05@gmail.com opinión sobre las trampas y el fraude académico.
- El profe queda en definir nuevo método para aplicar quices.

CONTINUACION EJEMPLOS DE DFA'S

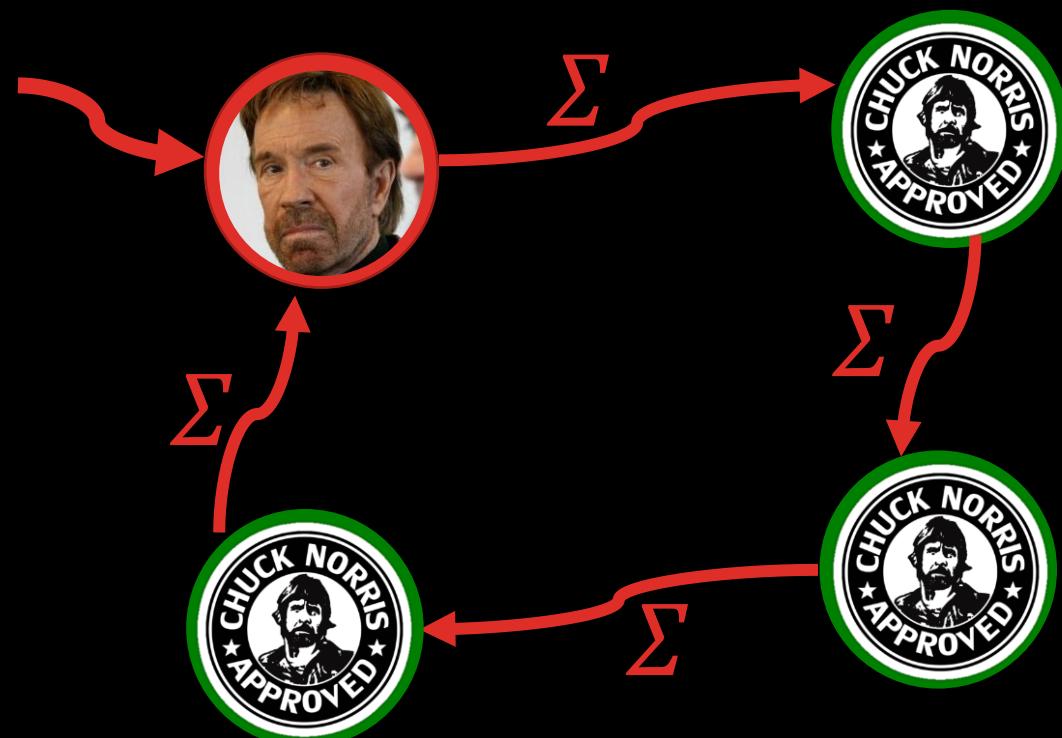
- Sea \mathcal{L} un lenguaje sobre $\Sigma = \{0, 1\}$ de hileras que terminan en 01



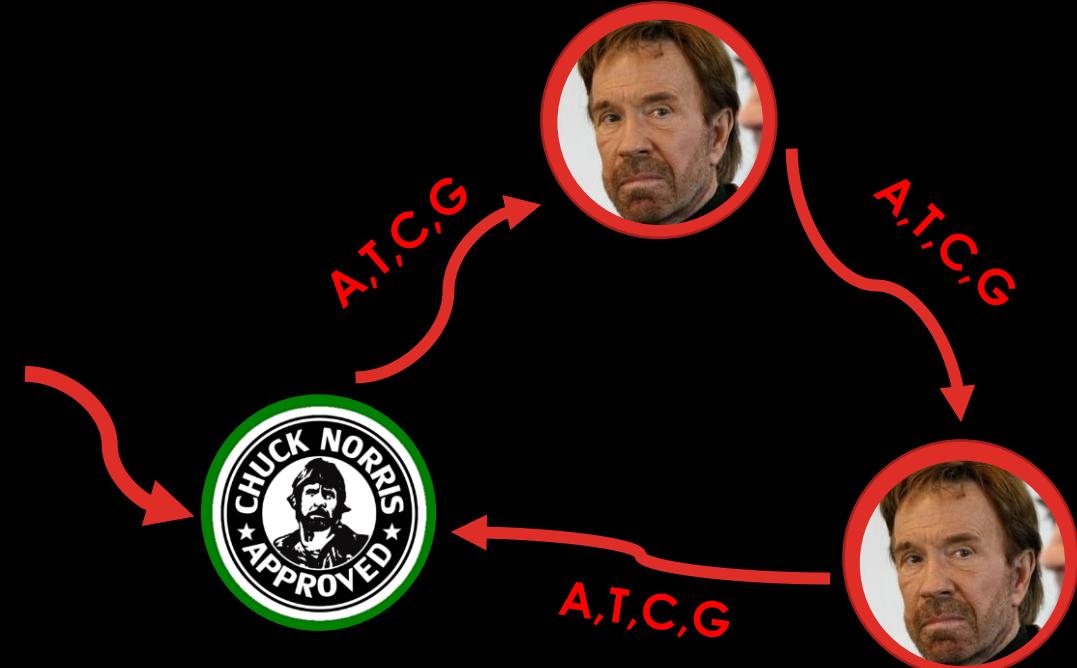
- Sea \mathcal{L} un lenguaje sobre $\Sigma = \{0, 1\}$ de hileras que no terminan en 00



- Sea \mathcal{L} un lenguaje sobre $\Sigma = \{A, T, C, G\}$ de hileras cuya longitud sea múltiplo de 3

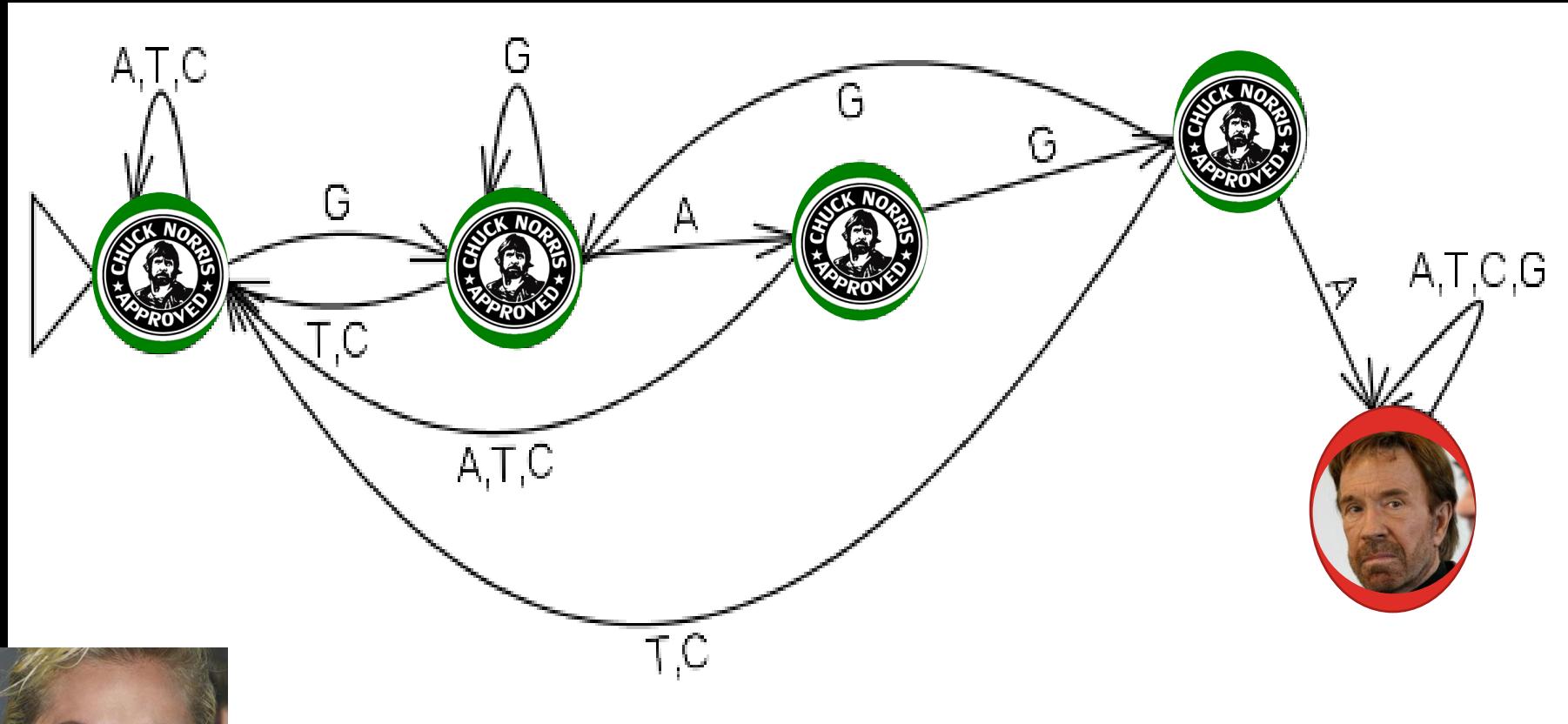


- Sea \mathcal{L} un lenguaje sobre $\Sigma = \{A, T, C, G\}$ de hileras cuya longitud sea múltiplo de 4

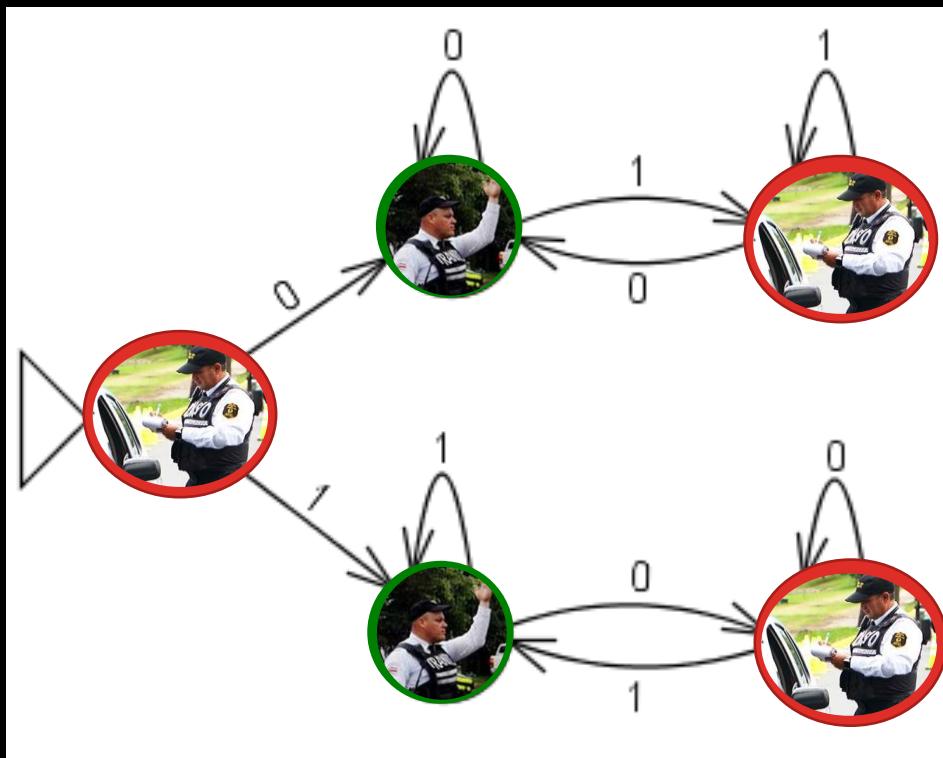


Ahora si se viene lo chido

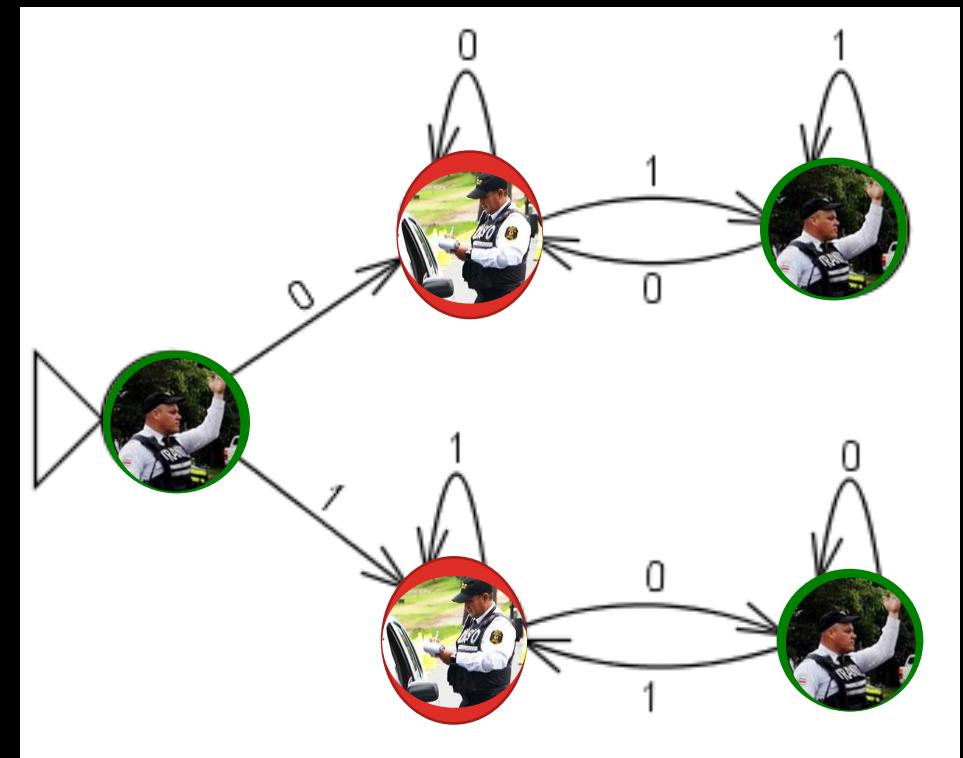
- Sea \mathfrak{L} un lenguaje sobre $\Sigma = \{A, T, C, G\}$ de hileras que **no contengan** la subhilera “**GAGA**”

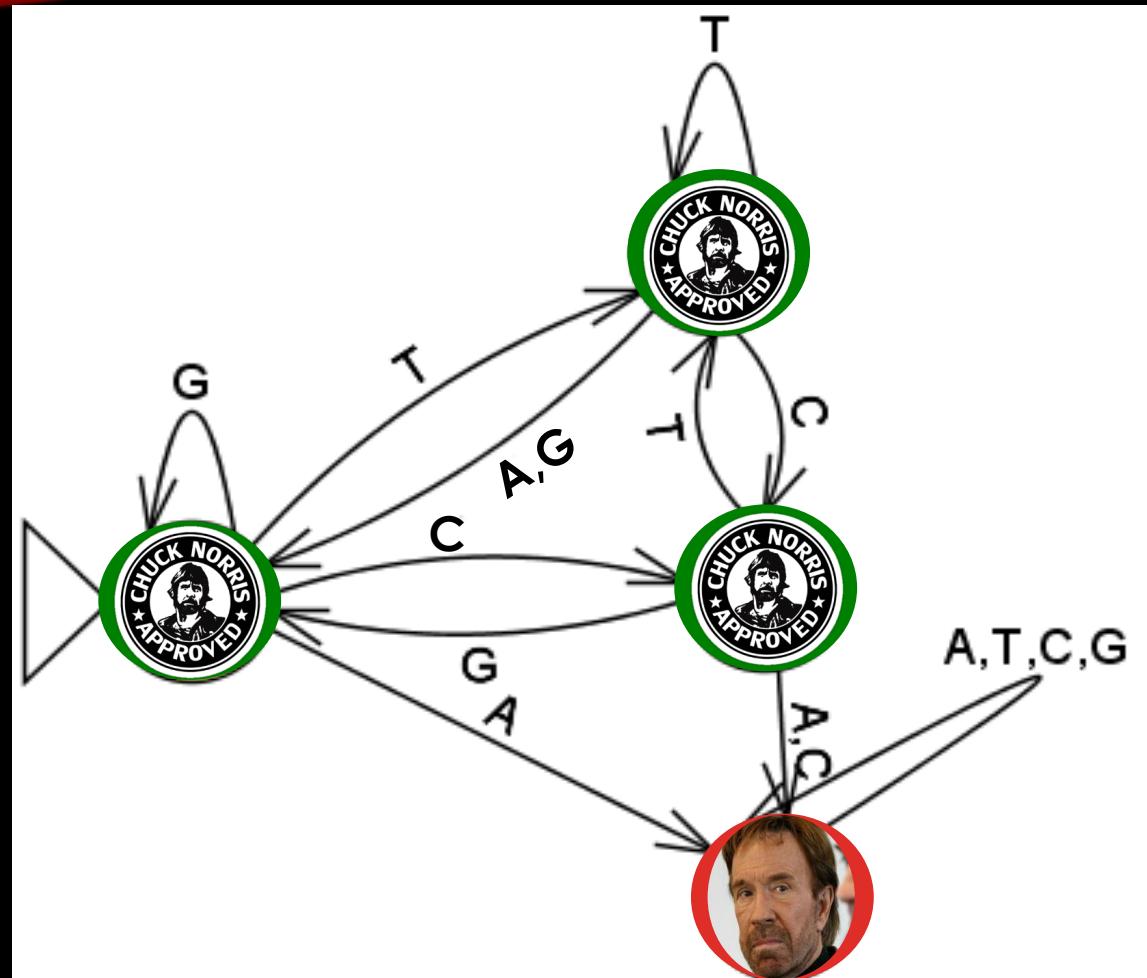


- Sea \mathfrak{L} un lenguaje sobre $\Sigma = \{1,0\}$ de hileras que terminen con el mismo símbolo con el que empezaron



- Sea \mathfrak{L} un lenguaje sobre $\Sigma = \{1,0\}$ de hileras que **no terminen** con el mismo símbolo con el que empezaron

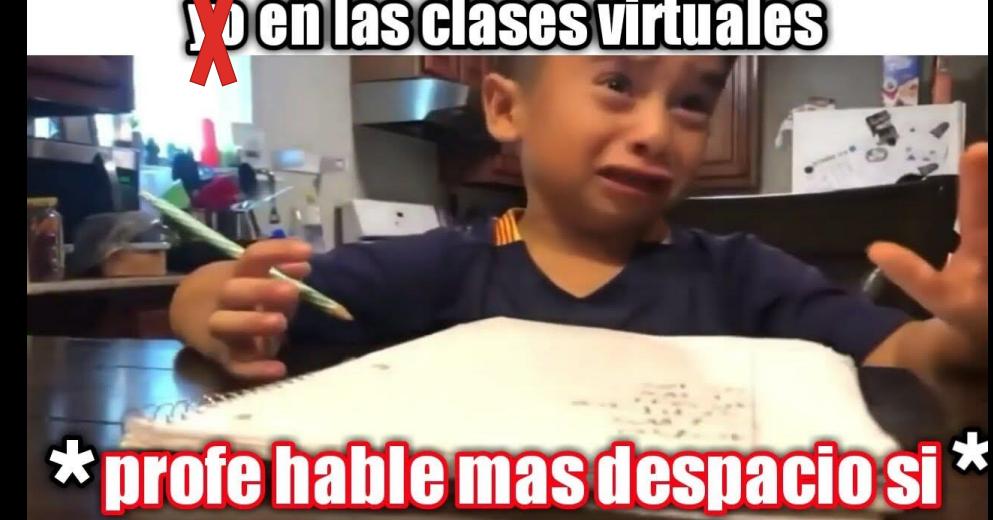




- Sea \mathcal{L} un lenguaje sobre $\Sigma = \{A, T, C, G\}$ de hileras que no contengan la subhilera “CC” y donde toda A es inmediatamente precedida de una T

Los apuntadores

X en las clases virtuales



profe hable mas despacio si

- Sea \mathcal{L} un lenguaje sobre $\Sigma = \{1,2,3,4,5,6,7,8,9\}$ de hileras que representan un número en base 10 divisible entre 3. (La hilera vacía equivale a 0)



Análisis

0,1,2



Posibles residuos al dividir entre 3

Divisible entre tres



Residuo 0
Forma: $3 * k$

Con k entero > 0

Número en base 10



Cada potencia
corresponde a una
potencia de 10

Si n es el número actual

→ Nuevo valor = $(n * 10) + q$

q → Nuevo dígito concatenado

Calculo de nuevo residuo según nuevo símbolo concatenado a la derecha



Forma $(3*k)$

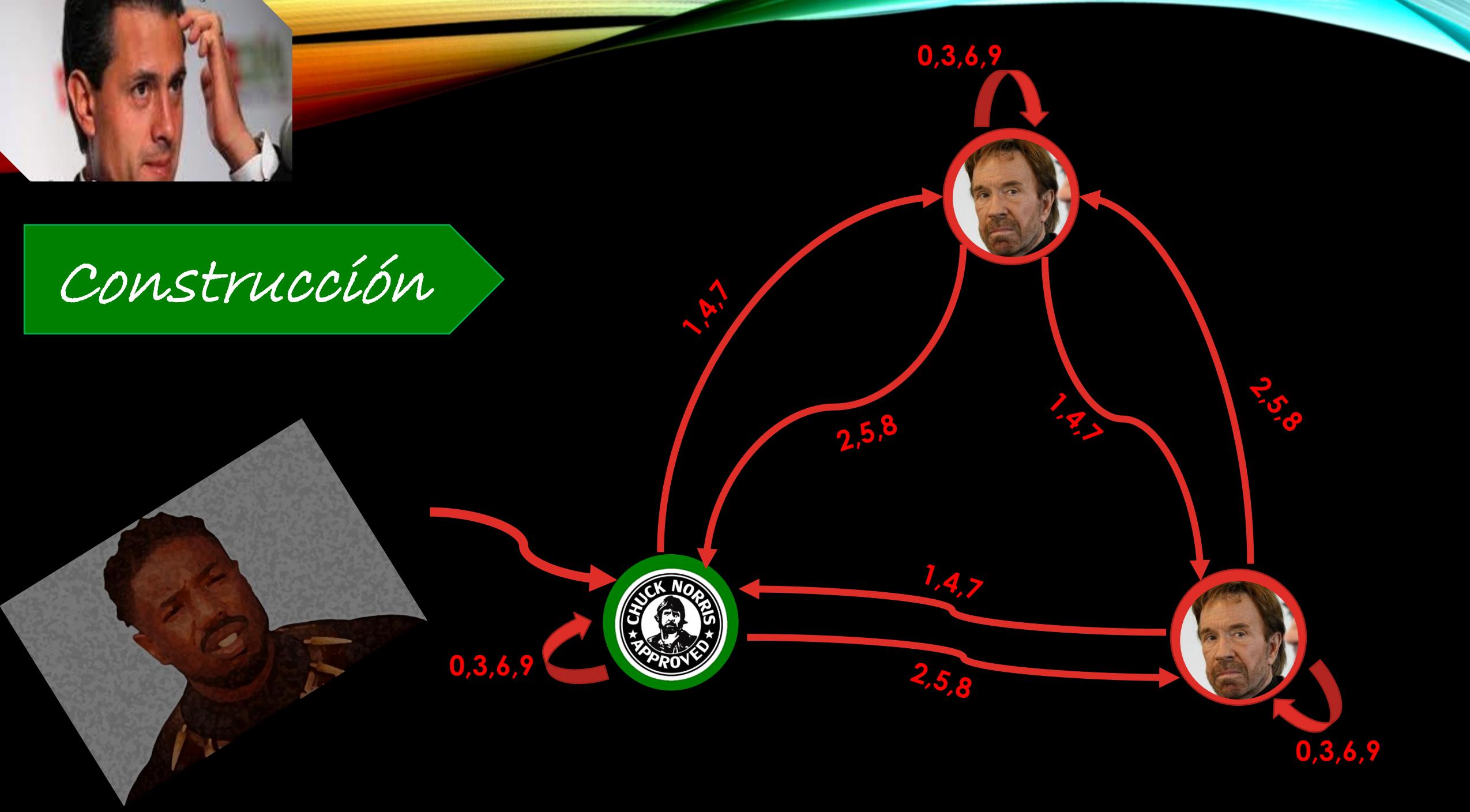
q	Nuevo valor	Residuo
0	$(3*k)*10+0$	0
1	$(3*k)*10+1$	1
2	$(3*k)*10+2$	2
3	$(3*k)*10+3$	0
4	$(3*k)*10+4$	1
5	$(3*k)*10+5$	2
6	$(3*k)*10+6$	0
7	$(3*k)*10+7$	1
8	$(3*k)*10+8$	2
9	$(3*k)*10+9$	0

Forma $(3*k)+1$

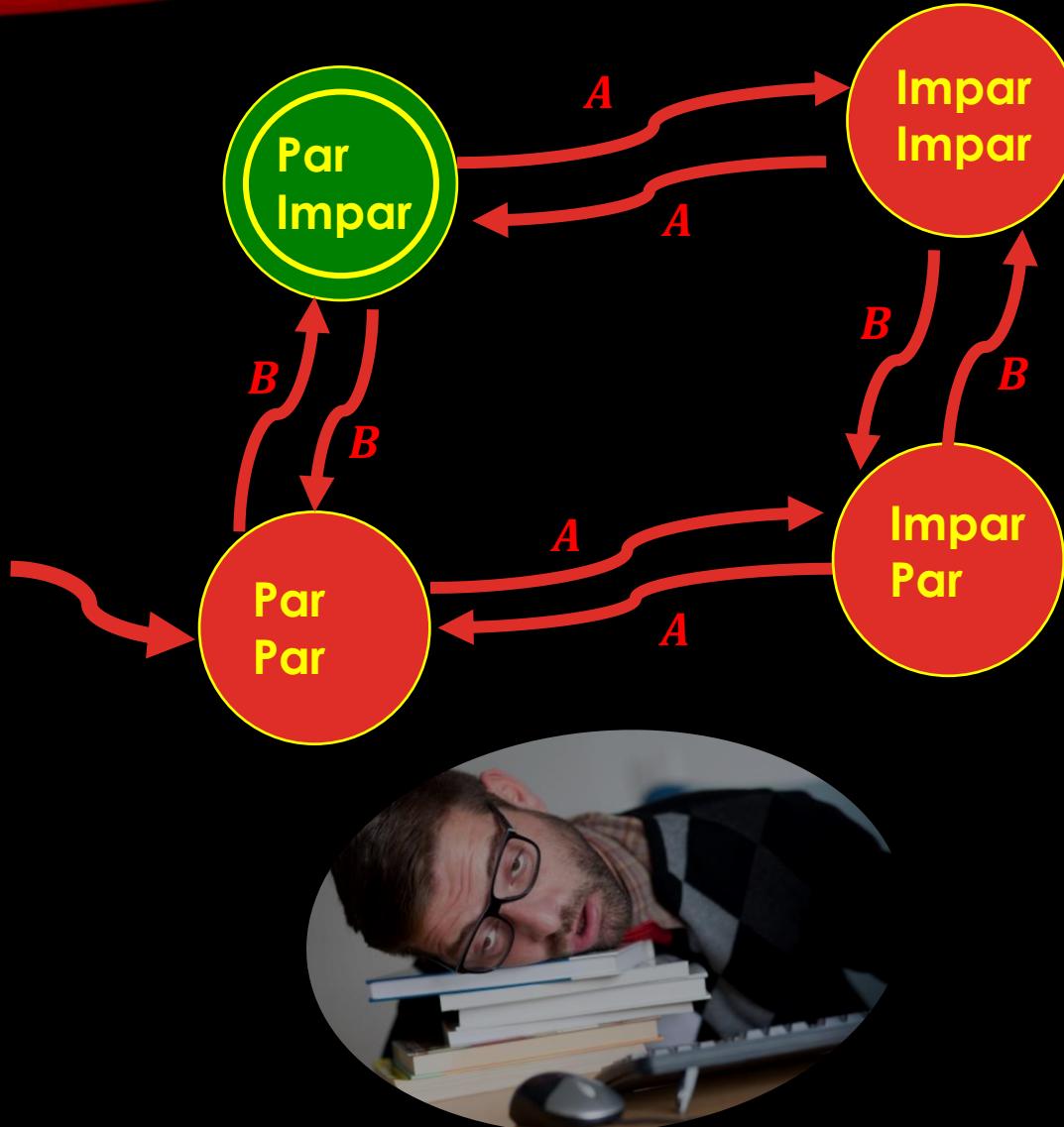
q	Nuevo valor	Residuo
0	$((3*k)+1)*10+0$	1
1	$((3*k)+1)*10+1$	2
2	$((3*k)+1)*10+2$	0
3	$((3*k)+1)*10+3$	1
4	$((3*k)+1)*10+4$	2
5	$((3*k)+1)*10+5$	0
6	$((3*k)+1)*10+6$	1
7	$((3*k)+1)*10+7$	2
8	$((3*k)+1)*10+8$	0
9	$((3*k)+1)*10+9$	1

Forma $(3*k)+2$

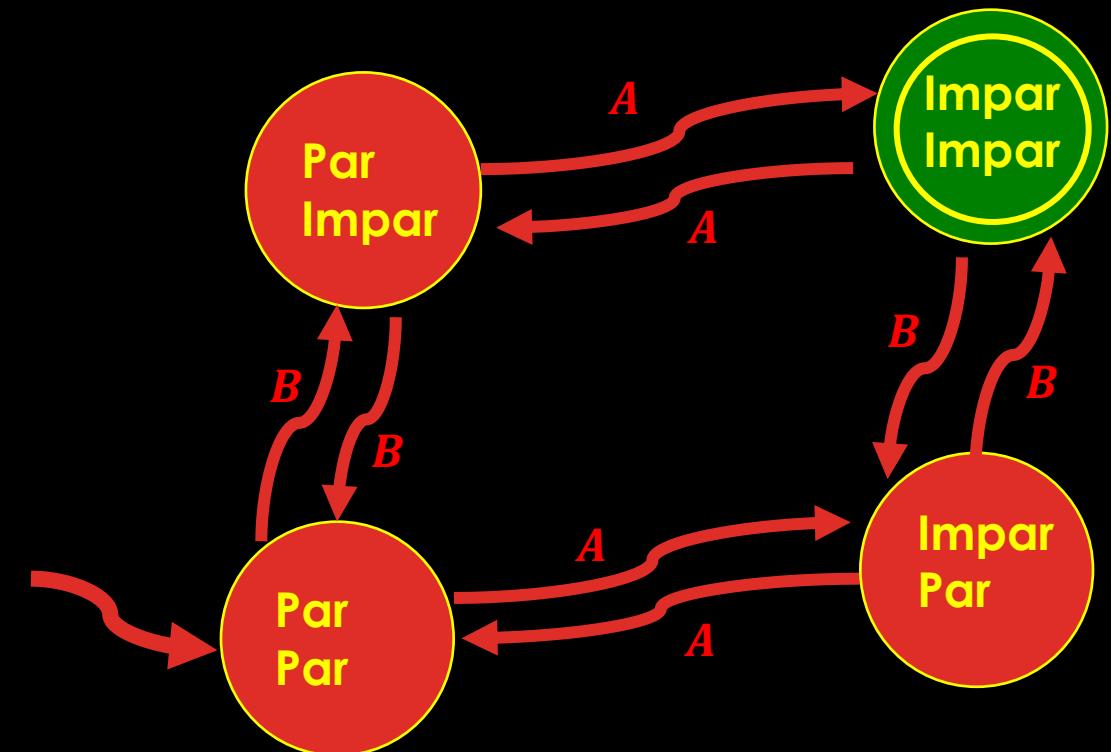
q	Nuevo valor	Residuo
0	$((3*k)+2)*10+0$	2
1	$((3*k)+2)*10+1$	0
2	$((3*k)+2)*10+2$	1
3	$((3*k)+2)*10+3$	2
4	$((3*k)+2)*10+4$	0
5	$((3*k)+2)*10+5$	1
6	$((3*k)+2)*10+6$	2
7	$((3*k)+2)*10+7$	0
8	$((3*k)+2)*10+8$	1
9	$((3*k)+2)*10+9$	2



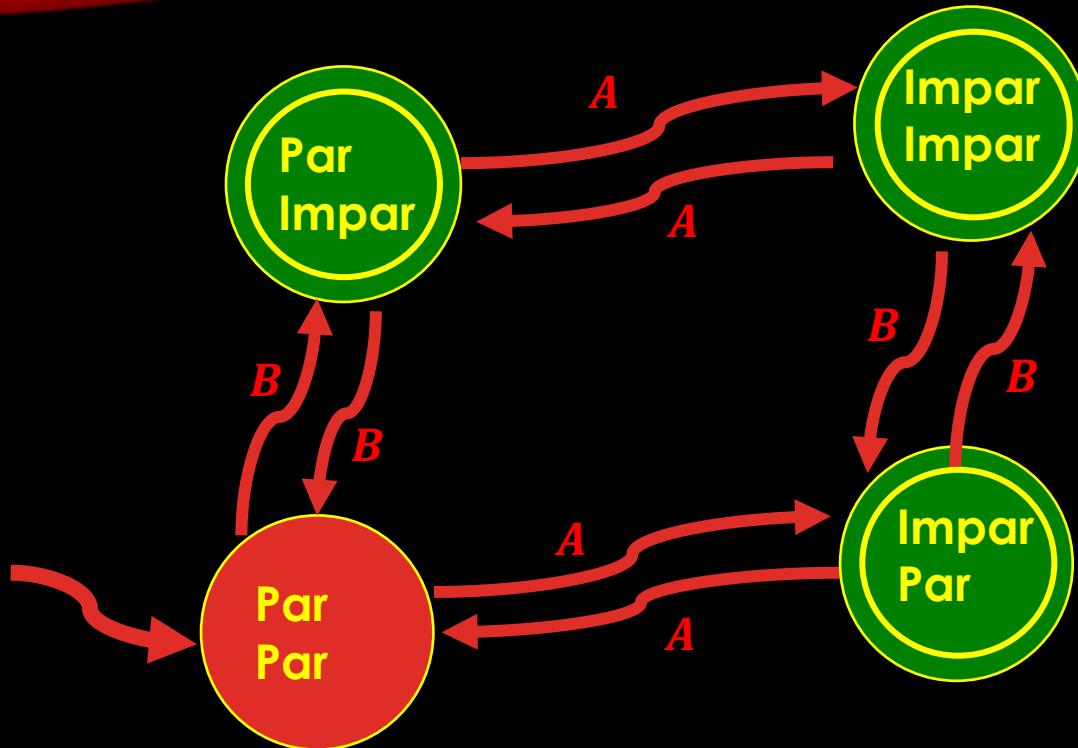
- Sea \mathcal{L} un lenguaje sobre $\Sigma = \{A, B\}$ de hileras que contengan un número par de A's y un número impar de B's



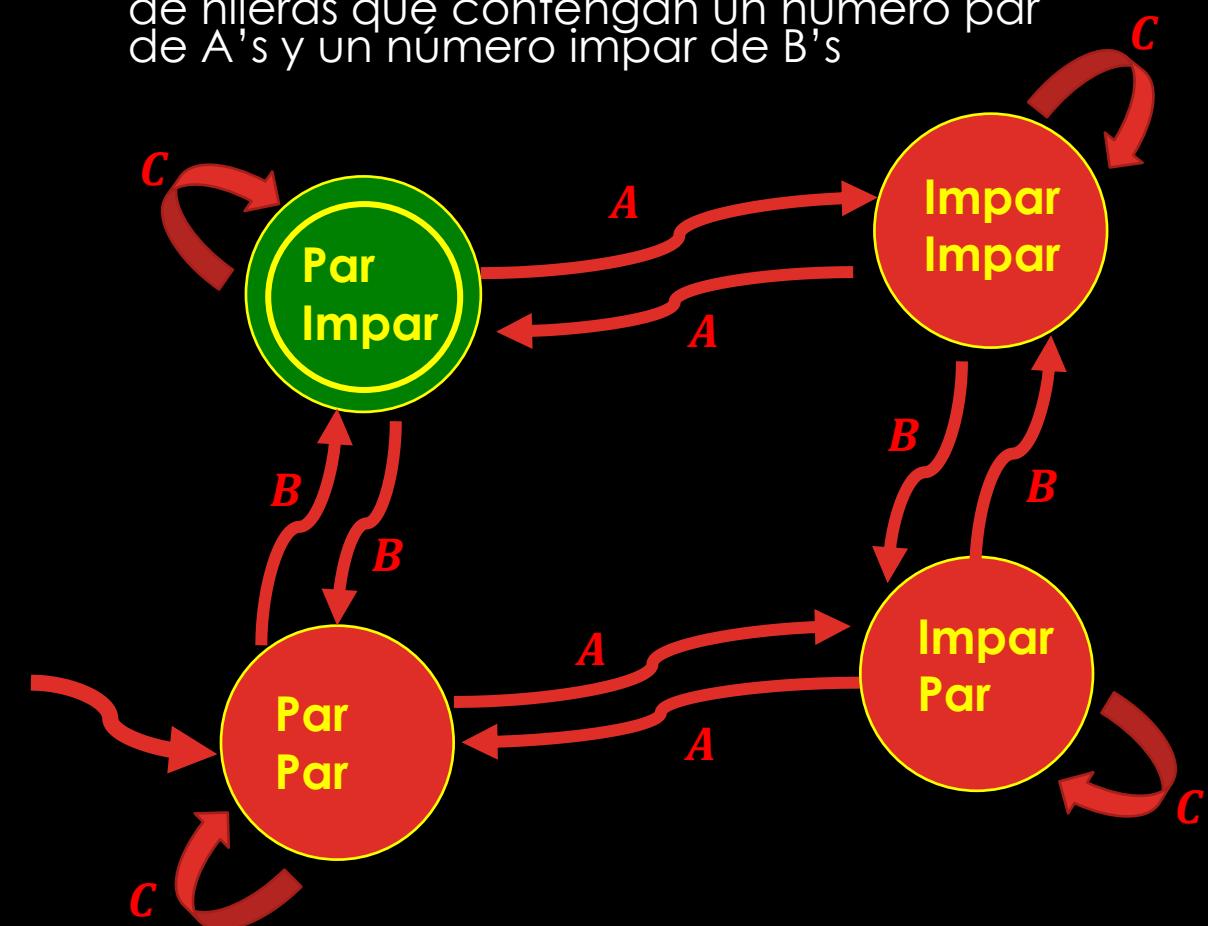
- Sea \mathcal{L} un lenguaje sobre $\Sigma = \{A, B\}$ de hileras que contengan un número impar de A's y un número impar de B's



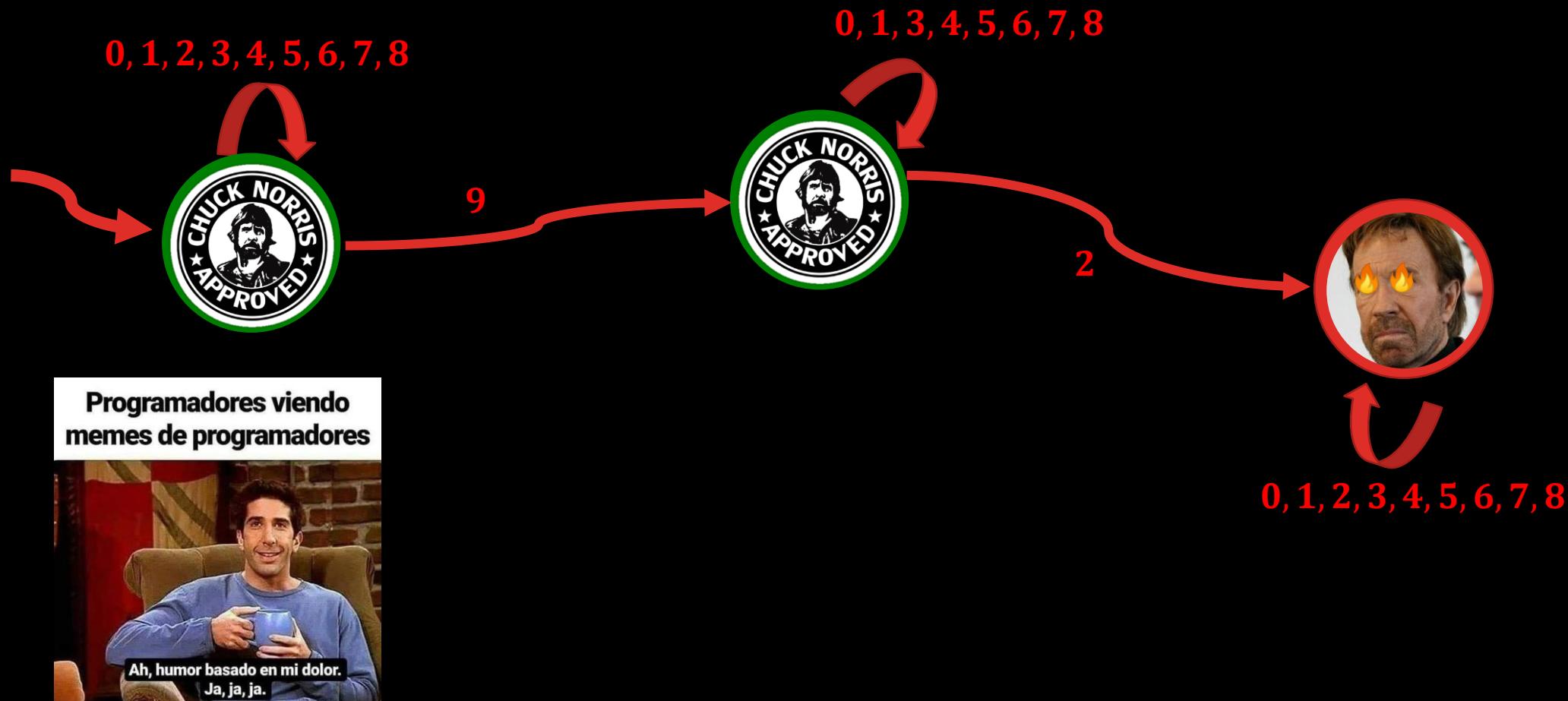
- Sea \mathcal{L} un lenguaje sobre $\Sigma = \{A, B\}$ de hileras que contengan un número impar de A's o un número impar de B's



- Sea \mathcal{L} un lenguaje sobre $\Sigma = \{A, B, C\}$ de hileras que contengan un número par de A's y un número impar de B's



- Sea \mathcal{L} un lenguaje sobre $\Sigma = \{0,1,2,3,4,5,6,7,8,9\}$ de hileras donde todos los 2 aparezcan antes que cualquier 9



APUNTES

07/10/2020

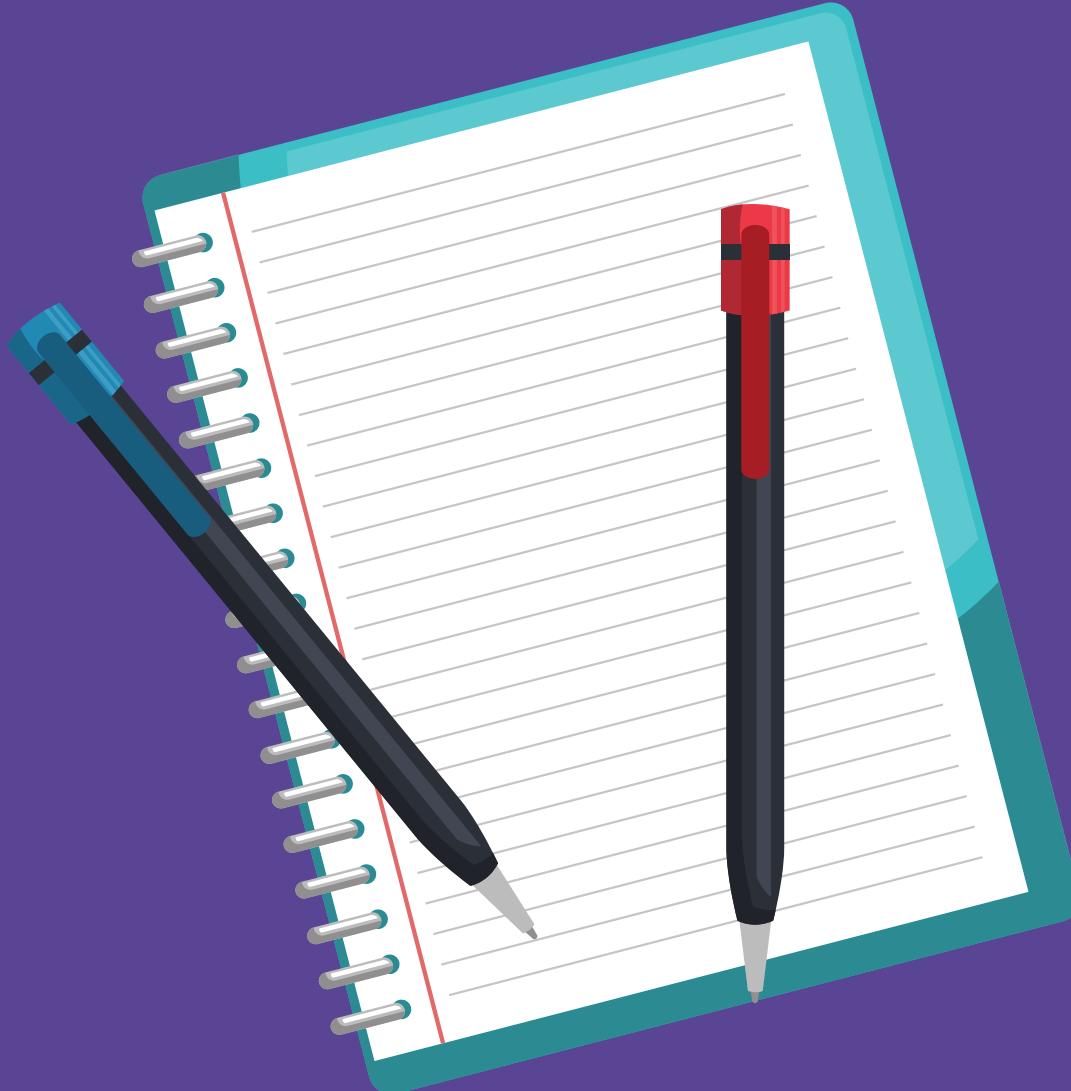
EMMANUEL CASTRO FERNÁNDEZ
2018104486



ÍNDICE

- QUIZ #4
- FECHAS IMPORTANTES
- PROYECTO 1
- AUTOMATA DETERMINISTICO DE ESTADOS FINITOS
- LENGUAJES Y DFA





QUIZ 4

01

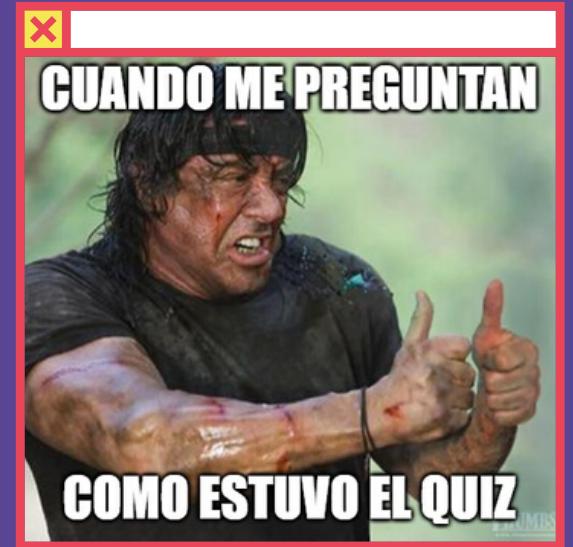
De un ejemplo de dos hileras no vacías $v, w \in \Sigma = \{a, b\}$, tales que cumplan las siguientes características.

1. $v \neq w$
2. $vw = wv$
3. $(vw)^{-1} \neq$

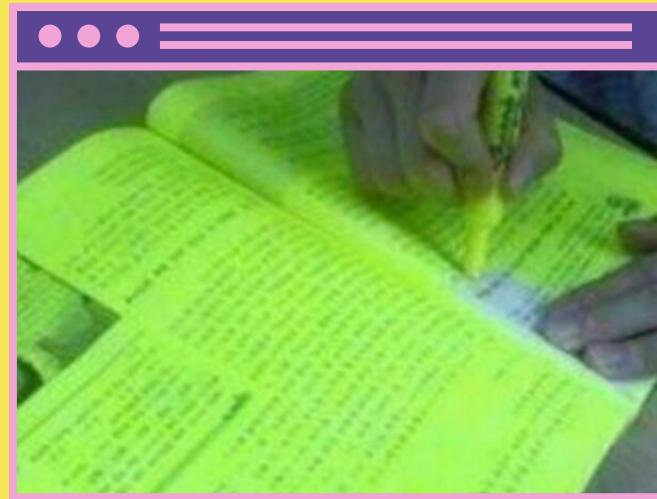
02

Sea $\Sigma = \{A, B\}$, presente los siguientes los siguientes lenguajes sobre Σ :

- a. Hileras de longitud 3
- b. Hileras de longitud 4 donde toda la subhilera de longitud 2 tiene al menos una B.
- c. Intersección entre los 2 lenguajes previos.
- d. Hileras de longitud 4 que terminen en A.
- e. Lenguaje definido en b. menos lenguaje definido en d.
- f. Hileras de longitud 5 que contengan la subhilera AAA en el centro.
- g. Concatenación del lenguaje f. con el lenguaje a.
- h. Concatenación del lenguaje c. con el lenguaje e.
- i. Reverso del lenguaje b.
- j. Lenguaje d. concatenado con lenguaje f. concatenado con lenguaje i.



FECHAS IMPORTANTES



Ensayos

Pinker Capitulo 4 07
de Octubre

Pinker Capitulo 5 14
de Octubre



Tarea Corta 01
11 de Octubre
12:00md



Proyecto 0
09 de Octubre

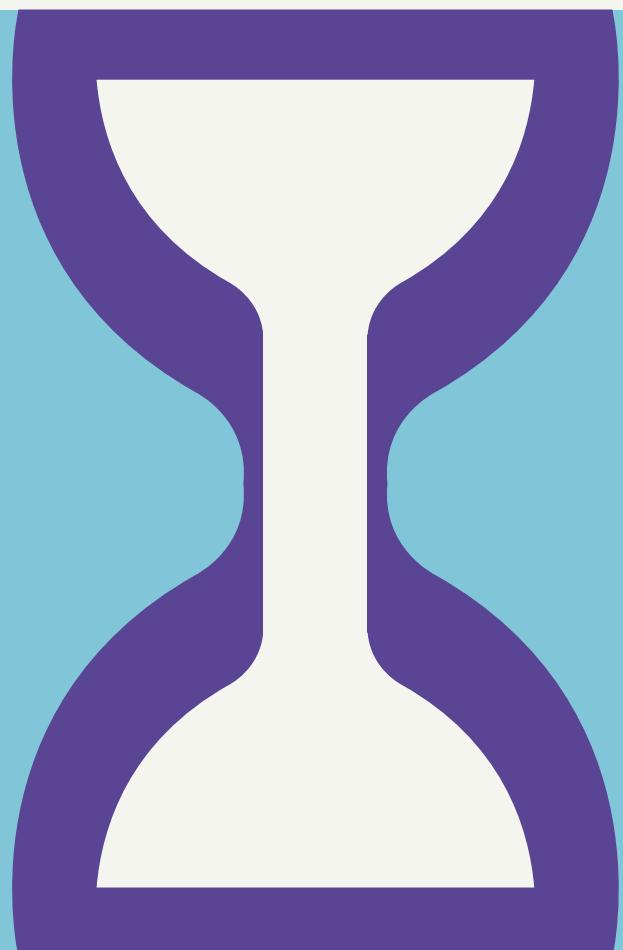


Examen 01
23 de Octubre 7:00am
Forro escrito a mano,
una hoja tamaño carta.

YA CASI PROYECTO 1...

El proyecto 1 es hacer un scanner más robusto.

Se podrá hacer uso de Flex.



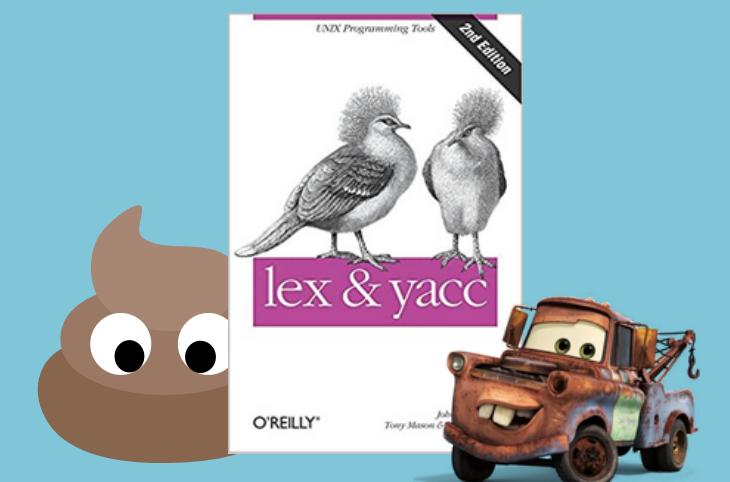
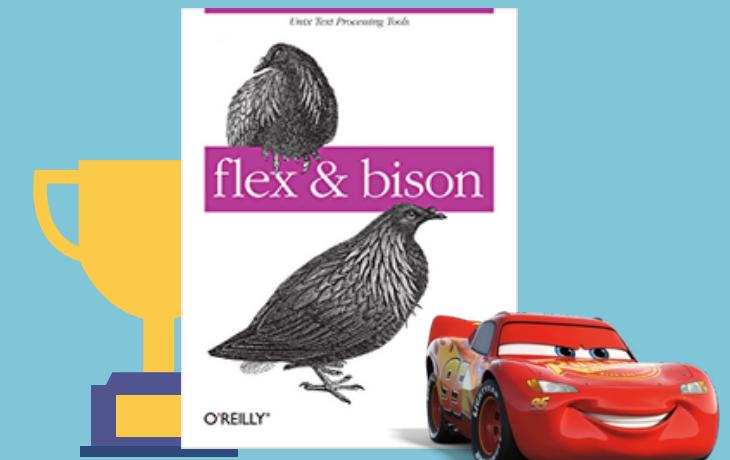
¿QUÉ ES FLEX? HAY QUE IR INVESTIGANDO.

FLEX (fast lexical analyzer generator) es una herramienta / programa de computadora para generar analizadores léxicos (escáneres o lexers) escrito por Vern Paxson en C alrededor de 1987. Se usa junto con el generador de analizador Berkeley Yacc o el generador de analizador GNU Bison.

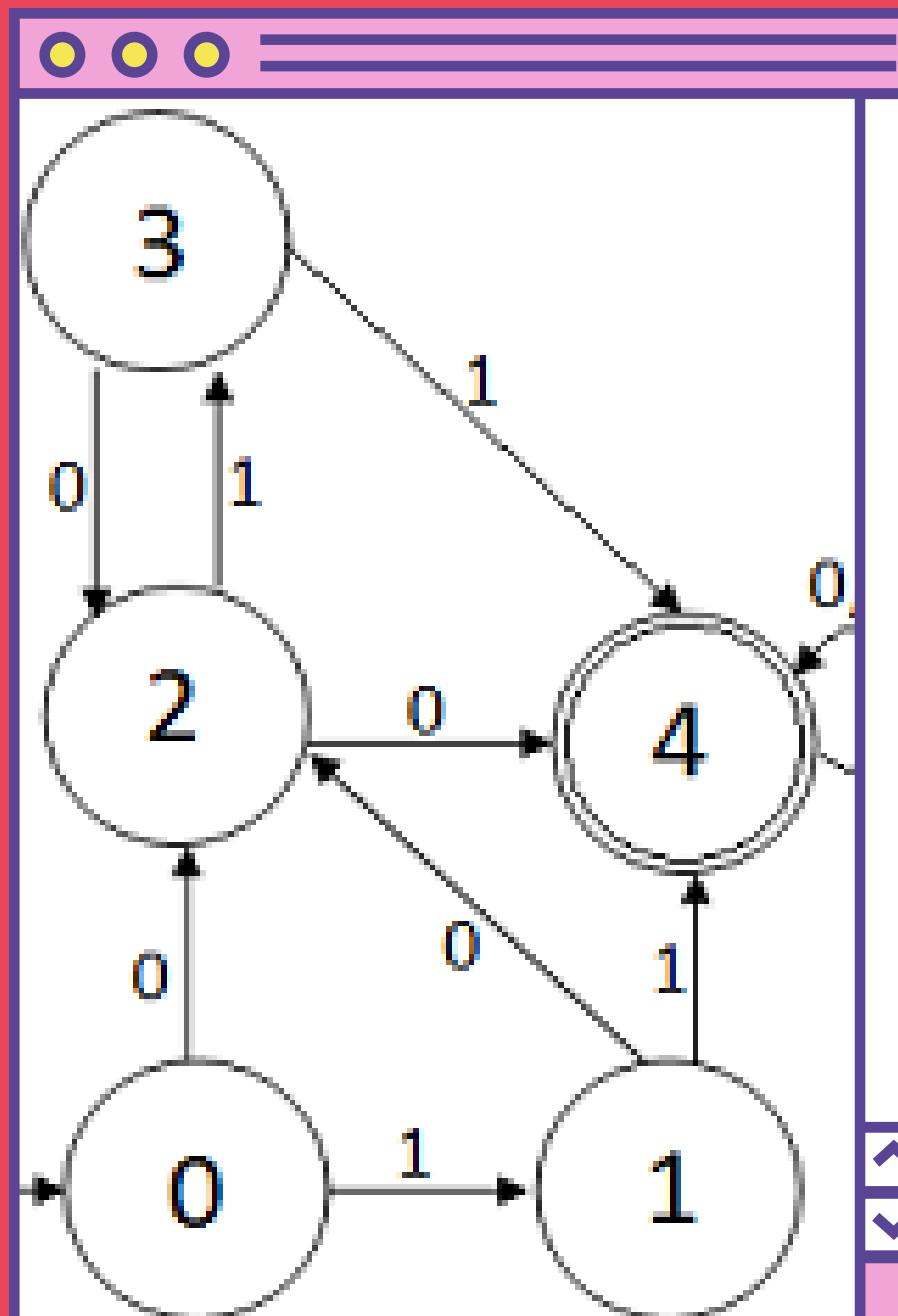
Flex y Bison son más flexibles que Lex y Yacc y producen un código más rápido. Bison produce un analizador a partir del archivo de entrada proporcionado por el usuario.

¿QUÉ ES BISON?

Bison es un generador de analizador sintáctico de propósito general que convierte una gramática libre de contexto anotada en un analizador LR determinista o LR generalizado (GLR) que emplea tablas de analizador LALR (1). Bison produce un analizador a partir del archivo de entrada proporcionado por el usuario.



AUTOMATAS DETERMINISTICOS DE ESTADOS FINITOS



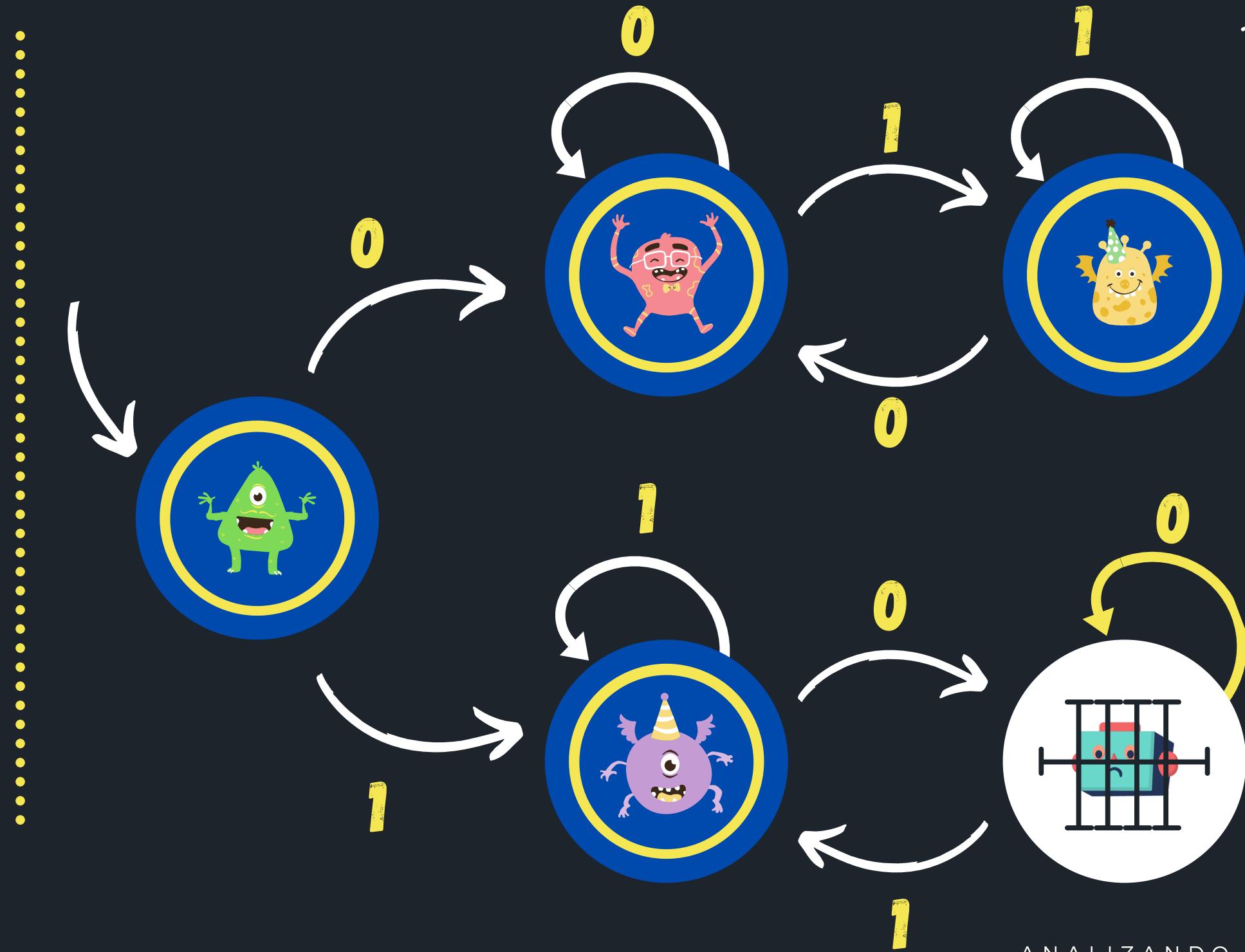
- En inglés Deterministic Finite Automaton (DFA).
- Se le conoce tambien como Maquina de Estados Finitos.
- Es finito, hay un numero fijo y finito de estados posibles.
- Es deterministico. Siempre en la misma entrada da la misma salida.
 - Dado un estado y un simbolo, hay un unico estado siguiente.
 - Dada una entrada siempre podemos predecir a donde vamos a terminar.

EJEMPLO DE AUTOMATA

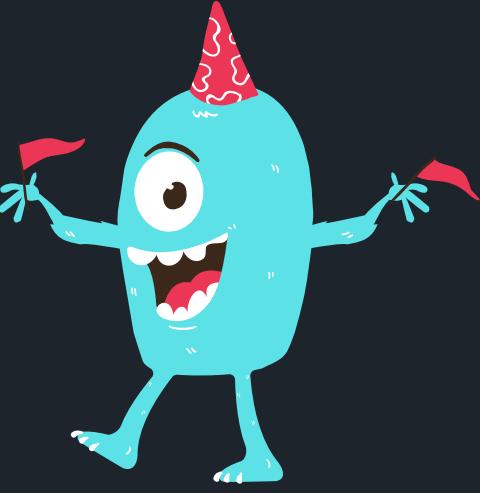
-RECUERDEN PONER EL DEDITO EN EL ESTADO INICIAL Y MOVERLO DE ACUERDO AL SIMBOLO ACTUAL Y HACER LA TRANSICIÓN CORRECTA, PARA QUE CHUCK LO ACEPTE.



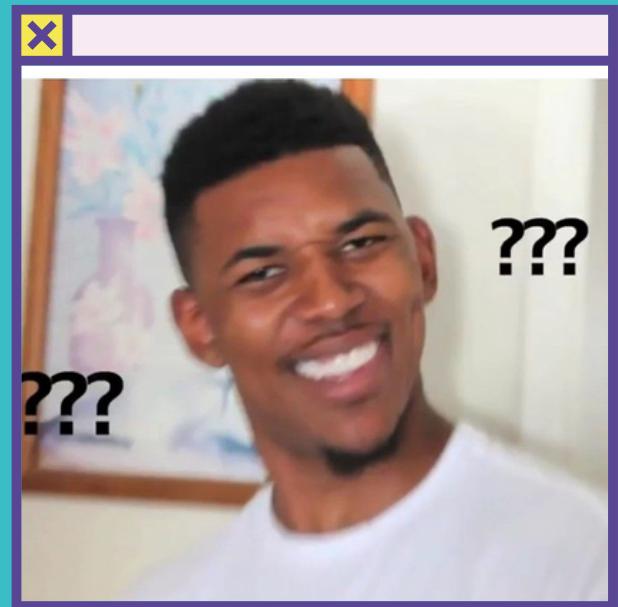
$W = 111101110$



ANALIZANDO LA
HILERA OBTENEMOS
QUE LA HILERA SE
RECHAZA



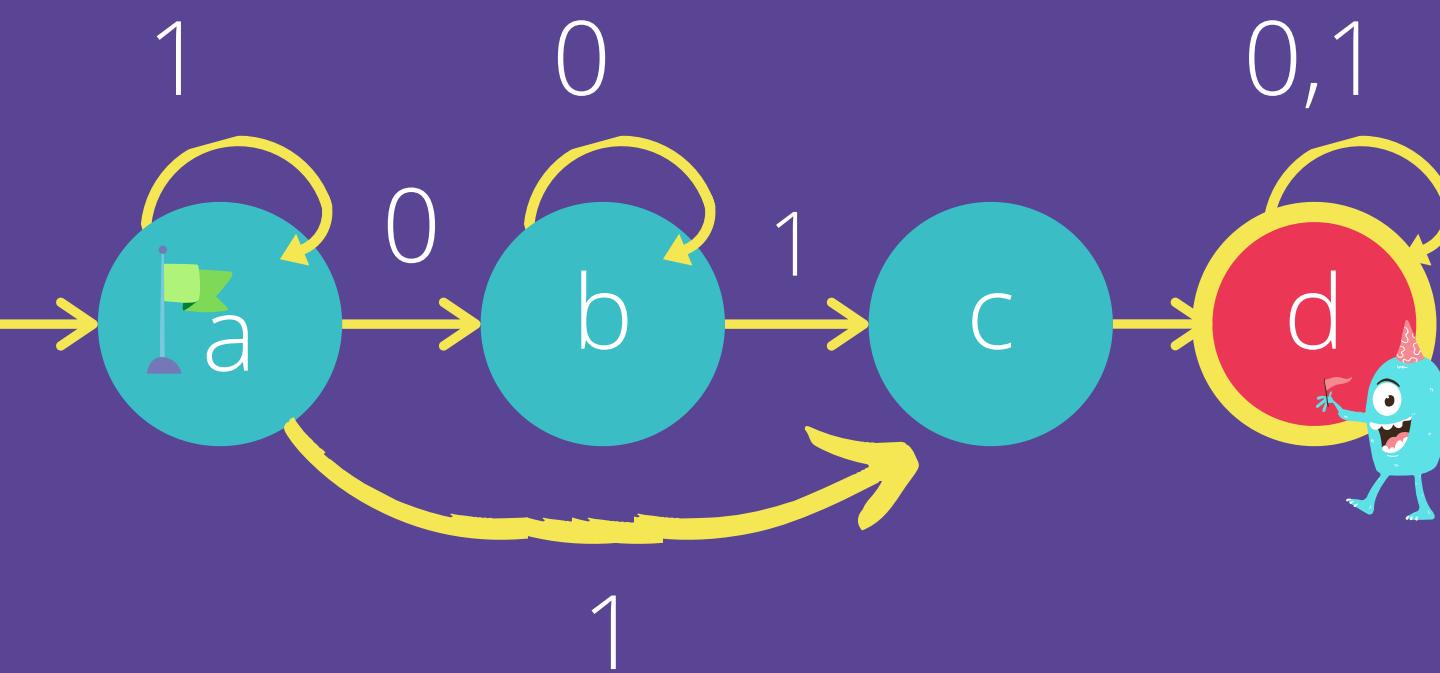
DFA



DEFINICIÓN FORMAL

Un DFA M es un quinto $M = (Q, \Sigma, \delta, q_0, F)$ donde:

- Q es un conjunto finito de estados.
- Σ es un alfabeto.(Etiquetas de las transiciones)
- $\delta: Q \times \Sigma \rightarrow Q$ es la función de transición. De un estado y un simbolo lo mapeo a un estado.
- $q_0 \in Q$, es el estado inicial.
 - Hay una única entrada, un único estado inicial.
- $F \subseteq Q$ conjunto de estados de aceptación o estados finales



EJEMPLO

$$M = (Q, \Sigma, \delta, q_0, F)$$

$Q = \{a, b, c, d\}$ → Conjunto de Estados

$\Sigma = \{0, 1\}$ → Etiquetas de las transiciones

$\delta = \begin{matrix} & 0 & 1 \\ a & b & a \\ b & b & c \\ c & d & a \\ d & d & d \end{matrix}$

Transiciones
Resultado de aplicar
 $Q \times \Sigma$
En palabras sencillas
dado un
estado(primer columna) y una
entrada(primer)
obtengo el estado al
que debo seguir.

$q_0 = a$ → Estado Inicial.
Recuerden solo puede haber uno

$F = d$ → Estados de Aceptación. Donde todo es felicidad.





IMPLEMENTACIÓN

CÓDIGO



¿CÓMO LOS PROGRAMAMOS?

- El algoritmo casi siempre es el mismo.
- Lo que suele cambiar son las tablas.
 - Filas: Estados
 - Columnas: Simbolo
 - Contenido: Siguiente Estado
- Tiene un vector de aceptación o rechazo.



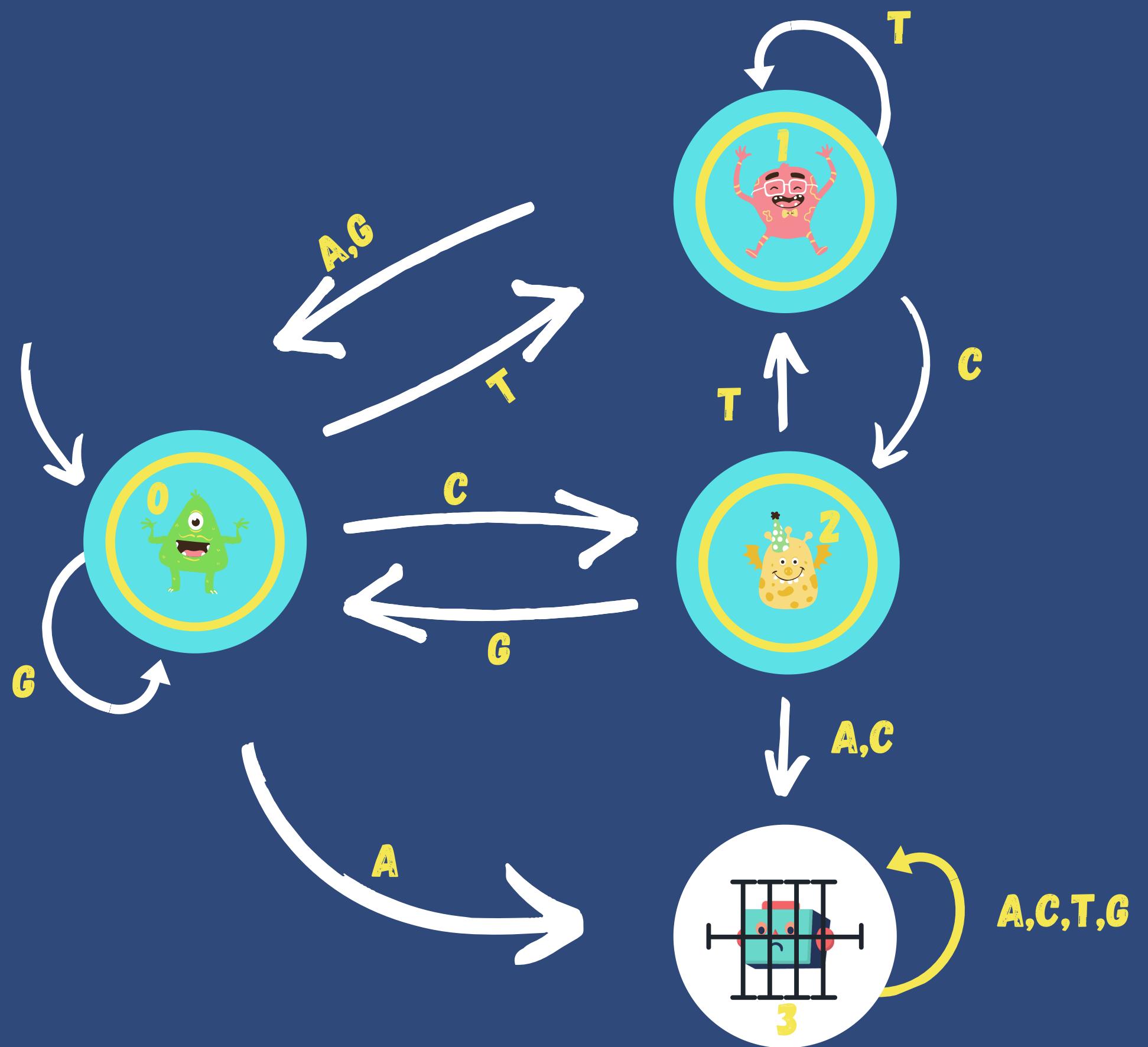
ARGUMENTOS

- Tabla de Transiciones
- Vector de Aceptación/Rechazo
- Puntero a una función de codificación.(Para agarrar los símbolos de la hilera)
- Hilera a ser revisada.
- Estado Inicial.

La función retorna una aceptación o rechazo

```
/*
Driver General para Cualquier Automata
Entrada: Matriz donde estan Las transiciones, Vector
de Aceptación
,puntero a función(Regresa un entero y recibe un cha
r),
La hilera y un estado inicial.
*/
int DFA_driver(int **Table, int *Accept, int (*code
)(char c),
               char * string, int state)
{
    //Para moverme dentro de La hilera;
    char *s;
    int k;
    s = string;
    k = state;
    while(*s){
        //Pasa al siguiente estado
        k = Table[k][code(* s++)]
    }
    return (Accept[k]);
}
```





	A	T	C	G
0	3	1	2	0
1	0	1	2	0
2	3	1	3	0
3	3	3	3	3

	1
0	1
1	1
2	1
3	0

**TODO
IMPLEMENTADO**

```
int tabla[4][4] = {{3,1,2,0},  
{0,1,2,0},  
,{3,1,3,0},  
{3,3,3,3}};
```

```
int accept[4] = {1,1,1,0}
```

```
int code(char c){  
if (c == 'A' ) return 0;  
if (c == 'T' ) return 1;  
if (c == 'C' ) return 2;  
if (c == 'G' ) return 3;  
}
```

DE LA TABLA
CUAL COLUMNA
NECESITO

```
DFA_driver(tabla, accept,code,hilera, 0)
```

DFA Y LENGUAJES

- Sea $M = (Q, \Sigma, \delta, q_0, F)$ un DFA.
- Sea $w = w_1w_2 \dots w_n$ una hilera sobre Σ
- M acepta w si existe una lista de estados r si:
 - a. Todos los elementos de r pertenecen a el conjunto de Estados(Q).
 - b. $r_0 = q_0$ Es decir r_0 es el estado inicial.
 - c. $\delta(r_i, w_{i+1}) = r$ para $i_{i+1} = 0, \dots, n-1$
 - d. $r \in F$

UNA MÁQUINA M RECONOCE AL LENGUAJE L SI Y SOLO SI:

$$L = \{ w \mid M \text{ acepta } w \}$$

Es decir cumple todos los requisitos a,b,c,d.



DISEÑAR DFA

**PARTE DE UN
SCANNER DE
COMPILEADOR
TÍPICO**

**SE PUEDE
AUTOMATIZAR**



MISIÓN

Nos dan un lenguaje L y hacemos un DFA peludo que lo reconozca.

REQUISITOS

El DFA tiene que aceptar todas las hileras de L y rechace todas las que no.

CONCEPTOS BÁSICOS

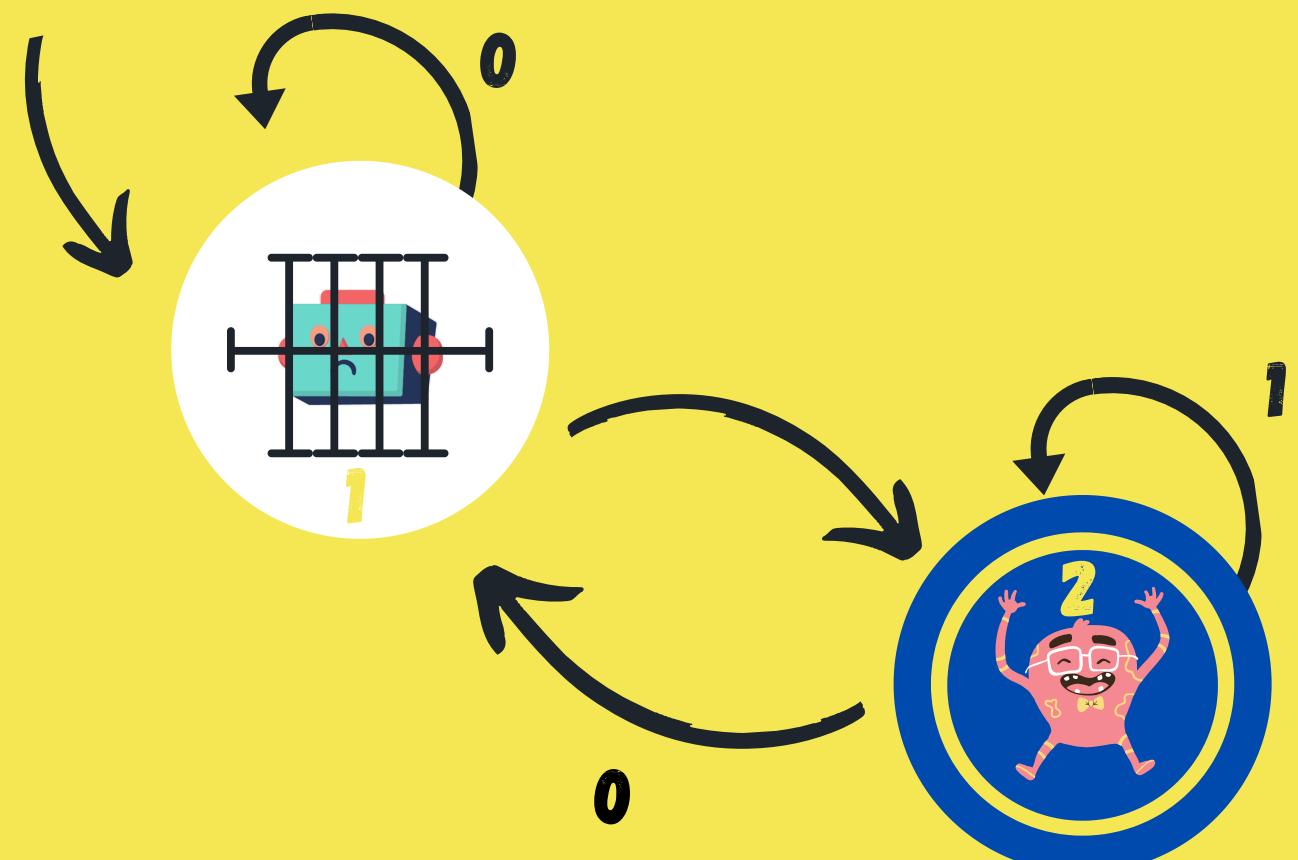
- Número finito de estados.
- Las etiquetas son nuestras amigas.
- De cada estado salen transiciones con todos los elementos de Σ .
- Si ϵ es parte del estado el estado inicial se acepta en caso contrario se rechaza



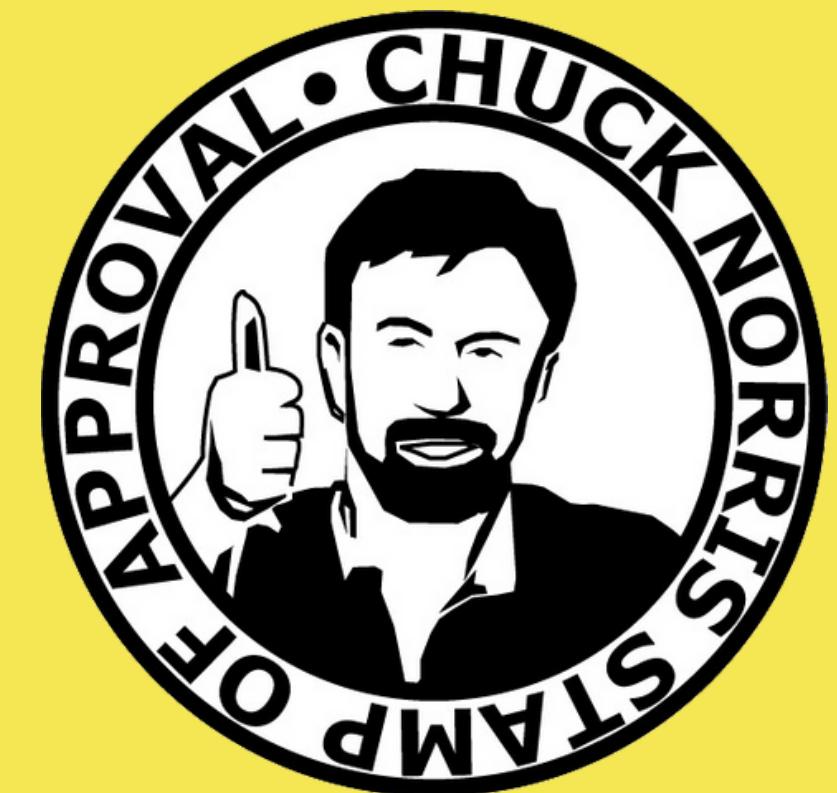
EJEMPLO 1

Sea L el lenguaje sobre $\Sigma = \{0,1\}$
de hileras que terminan en 1.

Diseñemos un DFA que
reconozaca L



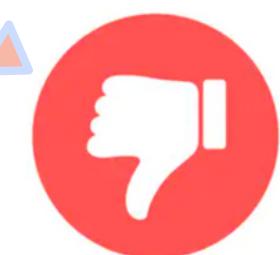
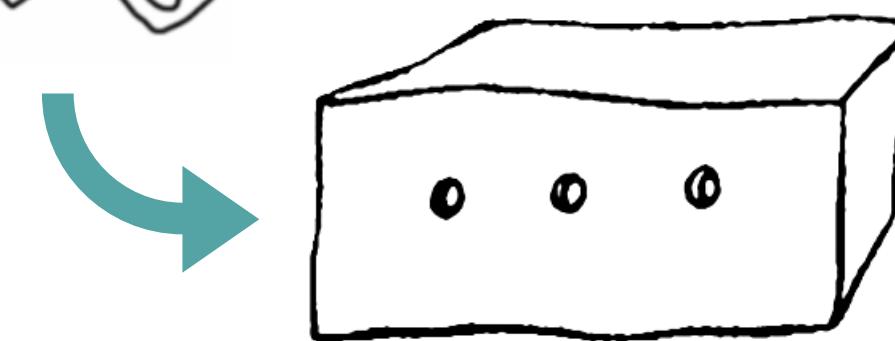
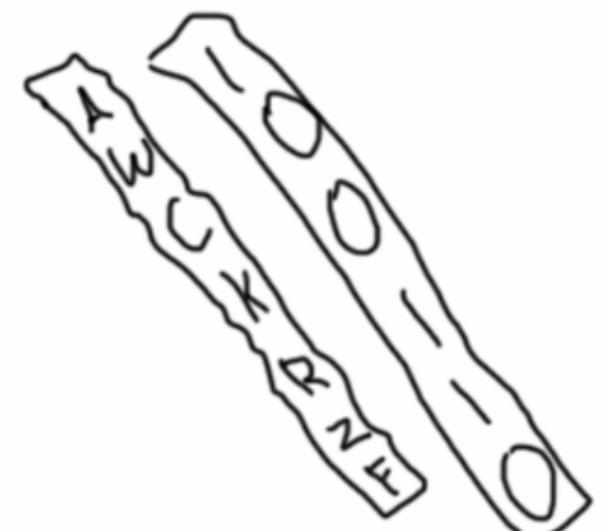
- Se pone el primer estado rechazado al no estar ϵ dentro del alfabeto.
- Cada estado tiene que tener la misma cantidad de transiciones que elementos del alfabeto



Apuntes 07-10

Steven Quesada J.
2018167911

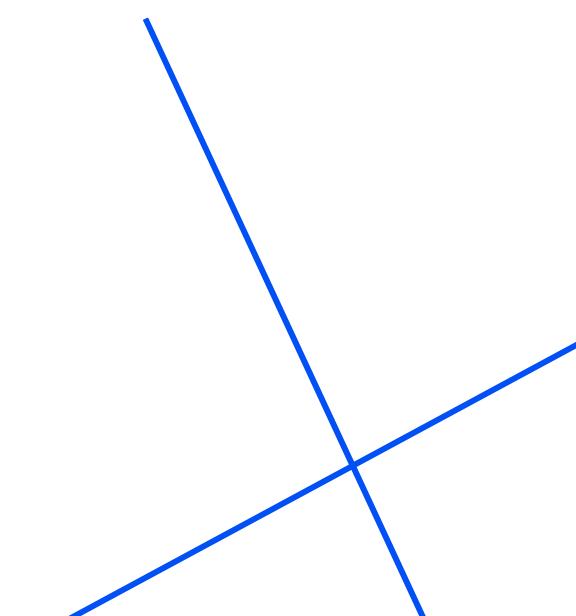
Autómatas



Quiz 04

1. Dé un ejemplo de dos hileras no vacías v, w sobre $\Sigma = \{a, b\}$, tales que cumplan todas las siguientes características al mismo tiempo:
 - a) $v \neq w$
 - b) $vw = wv$
 - c) $(vw)^{-1} \neq vw$

2. Sea $\Sigma = \{A, B\}$, presente los siguientes lenguajes sobre Σ :
 - a) Hileras de longitud 3
 - b) Hileras de longitud 4 donde toda subhilera de longitud 2 tiene al menos una B
 - c) Intersección entre los 2 lenguajes previos
 - d) Hileras de longitud 4 que terminen en A
 - e) Lenguaje definido en b) menos lenguaje definido en d)
 - f) Hileras de longitud 5 que contengan la subhilera AAA en el centro
 - g) Concatenación del lenguaje f) con el lenguaje a)
 - h) Concatenación del lenguaje c) con el lenguaje e)
 - i) Reverso del lenguaje b)
 - j) Lenguaje d) concatenado con lenguaje f) concatenado con lenguaje i)



Examen 1 – Viernes 23/10

Para el examen pueden hacer un forro.

- **Una hoja 8.5x11**
- **Escrito a mano**
- **Pueden anotar lo que quieran**
- **En el examen el profesor puede preguntar lo quiera**
- **Deben tener la cámara encendida**

*Entra toda la materia hasta el viernes 16 de octubre



Características de un DFA:

DFA: Deterministic Finite Automata

- ¿Finito?

Tiene un número fijo y finito de estados

- ¿Determinístico?

Algo con las misma entrada, genera el mismo resultado.

Dado un estado y un símbolo. hay un único estado siguiente.



Definición Formal:

Q: conjunto finito de estados.

Σ : es un alfabeto.

δ : $Q \times \Sigma \rightarrow Q$ es la función de transición (producto cartesiano).

q_0 : pertenece a Q , es el estado inicial (único).

$F \subseteq Q$: conjunto de estados de aceptación.

Ejemplo 1

$$M = (Q, \Sigma, \delta, q_0, F)$$

$$Q = \{a, b, c, d\}$$

$$\Sigma = \{0, 1\}$$

$$\delta = \underline{0} | \underline{1}$$

$$\begin{array}{c|c|c} a & b & a \end{array}$$

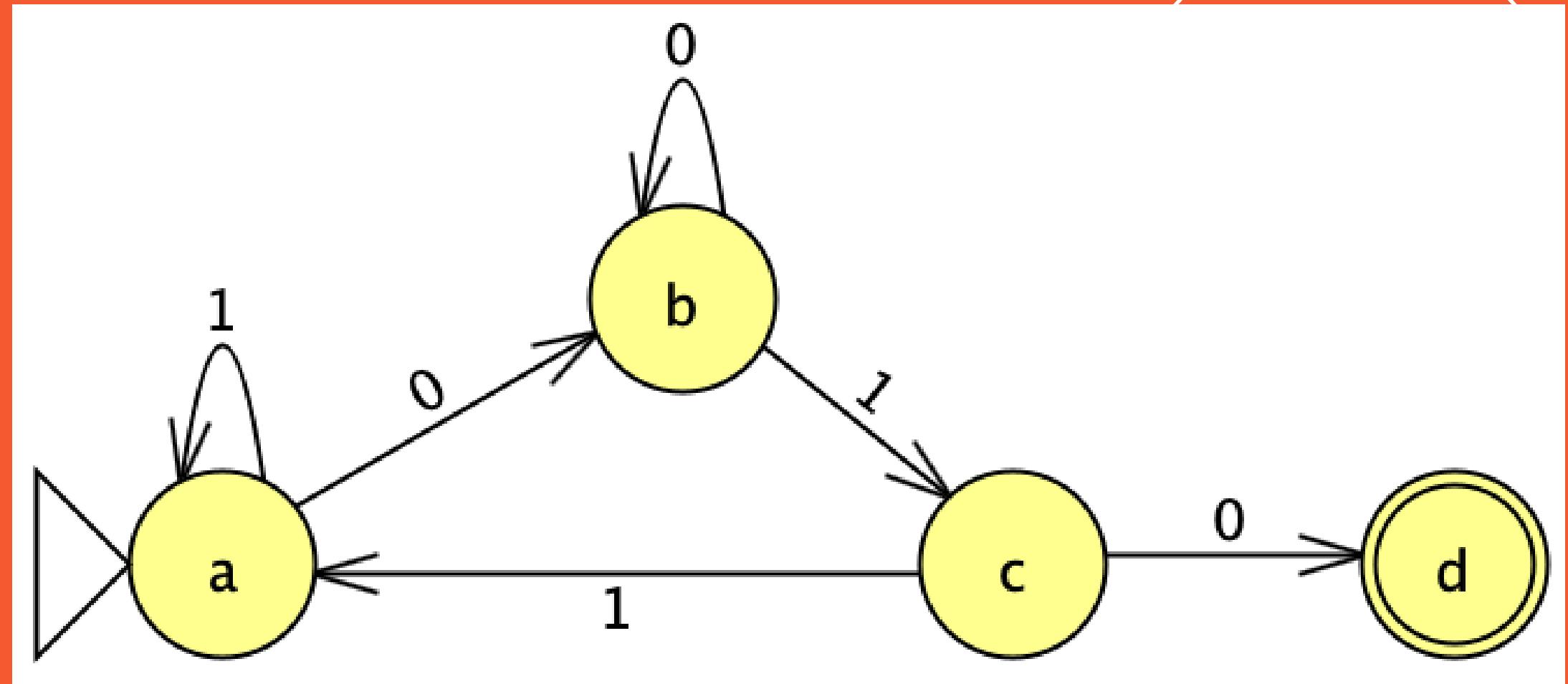
$$\begin{array}{c|c|c} b & b & c \end{array}$$

$$\begin{array}{c|c|c} c & d & a \end{array}$$

$$\begin{array}{c|c|c} d & d & d \end{array}$$

$$q_0 = a$$

$$F = \{d\}$$



Implementación de DFAs

¿Cómo lo programamos?

- El algoritmo siempre es casi el mismo
- Lo que cambia son las transiciones (tabla)
- Tabla:

Filas: estados

Columnas: símbolos

Contenido: siguiente estado

- Vector de aceptación/rechazo

Implementación de DFAs. cont

- Se puede tener un driver general para DFAs
- Argumentos:
 - + Tabla de transiciones
 - + Vector de aceptación/rechazo
 - + Puntero a función de codificación de simbolo
 - + Hilera a ser revisada
 - + Estado inicial
- Regresa "aceptación" o "rechazo" (1 o 0)

Implementación de DFAs - Ejemplo de código

```
int DFA_driver (int **Table,  
                int *Accept,  
                int (* code)(char c), // puntero a función  
                char * string,  
                int state)  
{  
    char *s;  
    int k;  
  
    s = string;  
    k = state;  
    while (*s)  
        k = Table[k] [code(*s++)]  
    return(Accept[k]);  
}
```

Proyecto 1

Usar Flex para generar el scanner



Ejemplo de uso

```
int tabla [4] [4] = {{3,1,2,0}, {0,1,2,0}, {3,1,3,0}, {3,3,3,3}};  
int accept[4] = {1,1,1,0};
```

```
int code(char c){  
    if ( c== 'A') return 0;  
    if ( c== 'T') return 1;  
    if ( c== 'C') return 2;  
    if ( c== 'G') return 3;  
}
```

```
...  
DFA_driver (tabla, accept, code, hilera, 0);  
...
```

	A	T	C	G
0	3	1	2	0
1	0	1	2	0
2	3	1	3	0
3	3	3	3	3

0	1
1	1
2	1
3	0

DFA y Lenguajes

- + $M = (Q, \Sigma, \delta, q_0, F)$ un DFA
- + Sea $w = w_1, w_2, \dots, w_n$ una hilera sobre Σ
- + M **acepta** w si existe una secuencia de estados $r_0, r_1, r_2, \dots, r_n$ tal que:

1. Todos los r_i pertenecen a Q
2. $r_0 = q_0$
3. $\delta(r_i, w_{i+1}) = r_{i+1}$, para $i = 0, \dots, n-1$
4. r_n pertenece a F

La máquina M reconoce al lenguaje L si y solo si:

$$L = \{w \mid M \text{ acepta } w\}$$

Nuestra misión

+ Diseñar DFAs que reconozcan lenguajes.

- Parte del scanner de un compilador típico
- Se puede automatizar...



Diseño de DFAs

Misión: nos describen un lenguaje L y hacemos un DFA que lo reconozca.

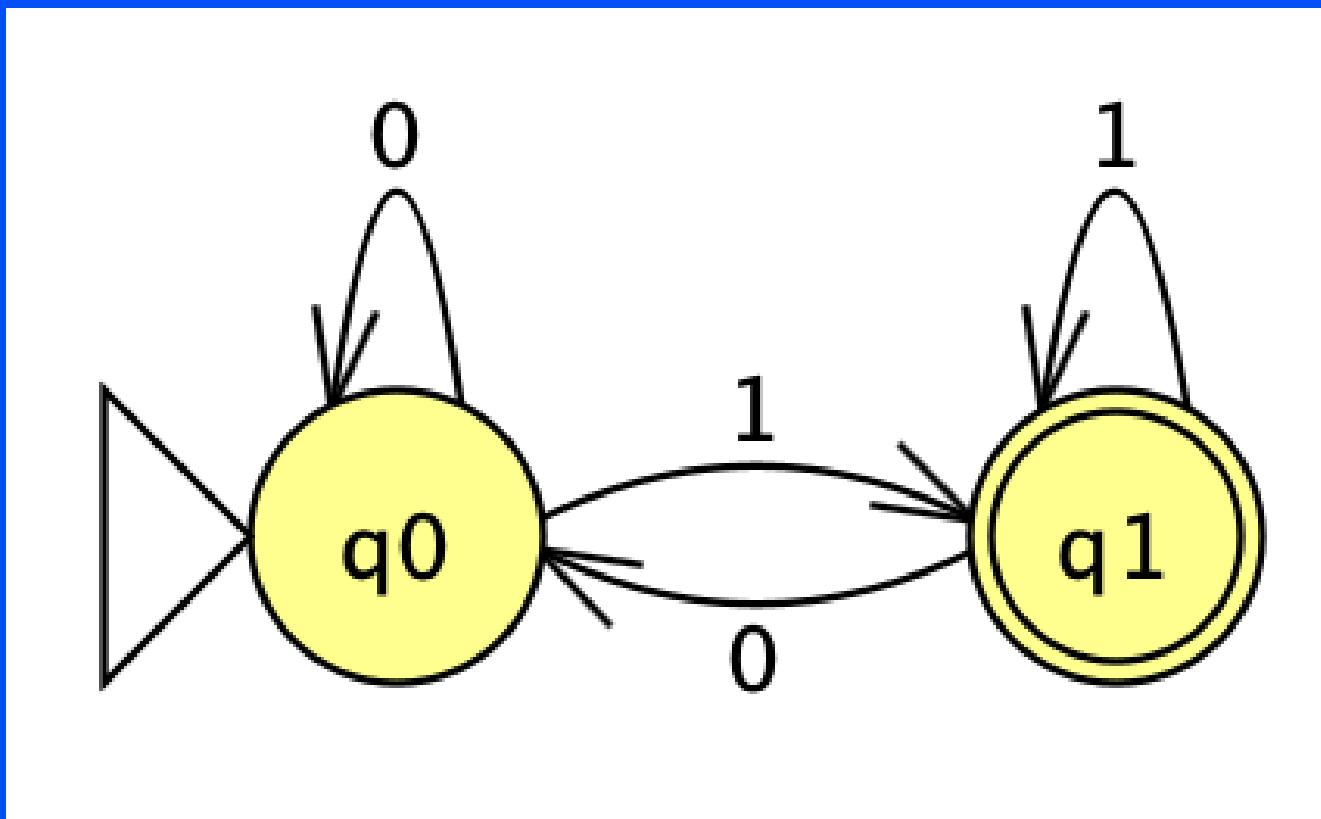
- + El DFA tiene que aceptar **todas** las hileras de L .
- + El DFA **no puede** aceptar hileras que **no sean miembros** de L .

Conceptos básicos:

- Número finito de estados.
- Las etiquetas de los estados son "nuestras amigas".
- En principio, de cada estado salen transiciones con todos los elementos de Σ .
- ¿Cuáles estados (situaciones) se deben aceptar?
- ¿Es epsilon parte de L ? (estado inicial de aceptación o rechazo)

Ejemplo 1

- Sea \mathbf{L} el lenguaje sobre $\Sigma = \{0,1\}$ de hileras que terminan en 1.
- Diseñe un DFA que reconozca a \mathbf{L} .

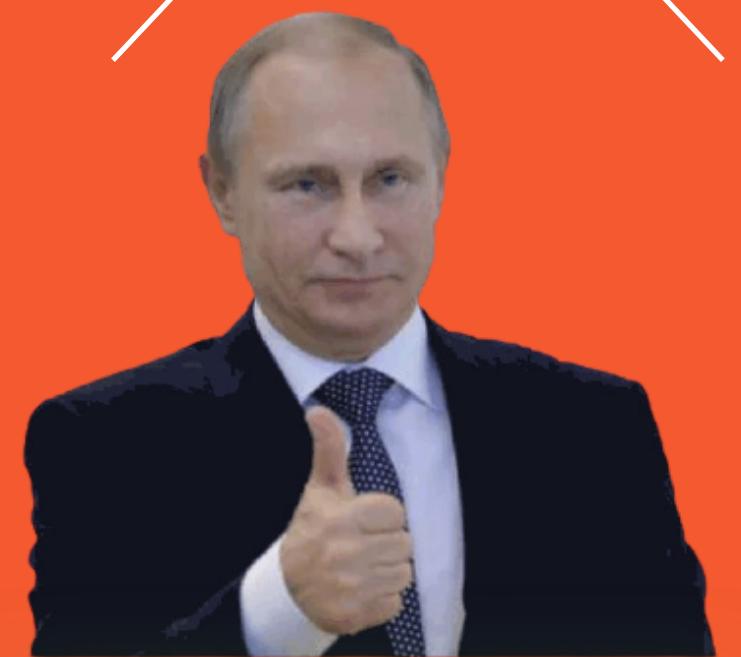
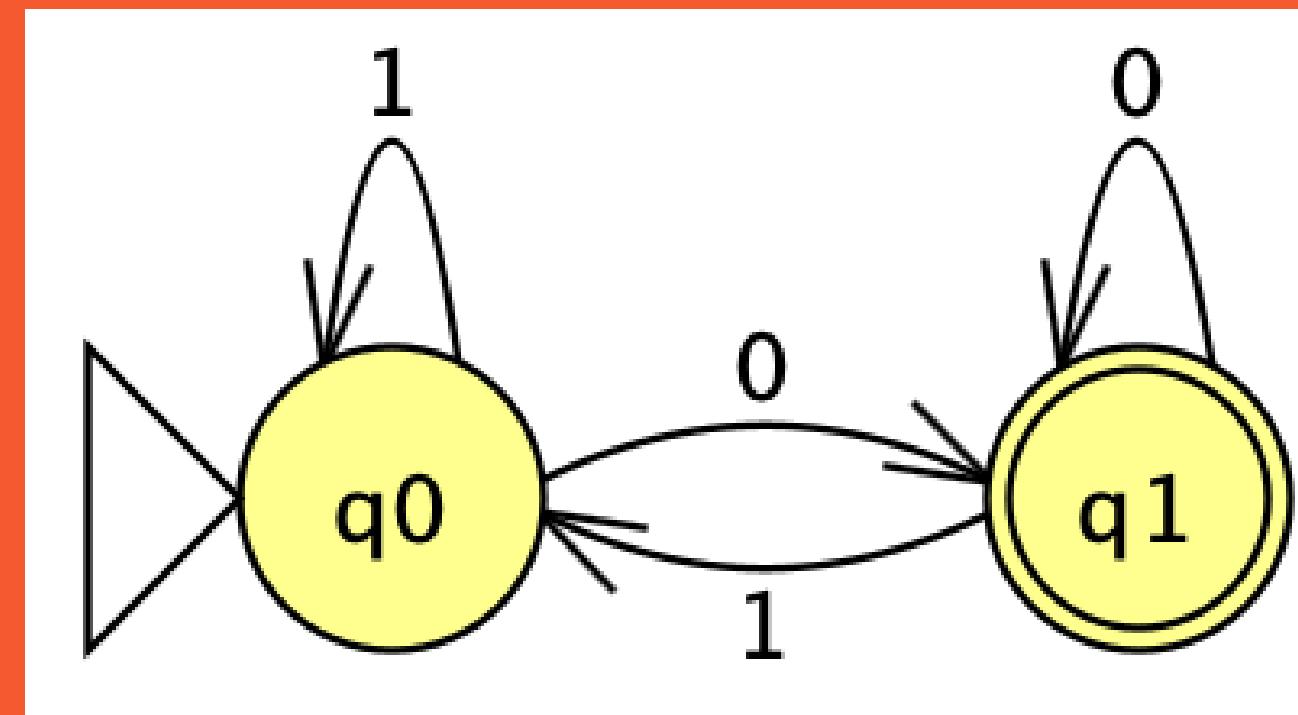


APROBADO

El software utilizado para hacer los DFAs, se llama JFLAP.
Lo pueden encontrar en jflap.org

Ejemplo 2

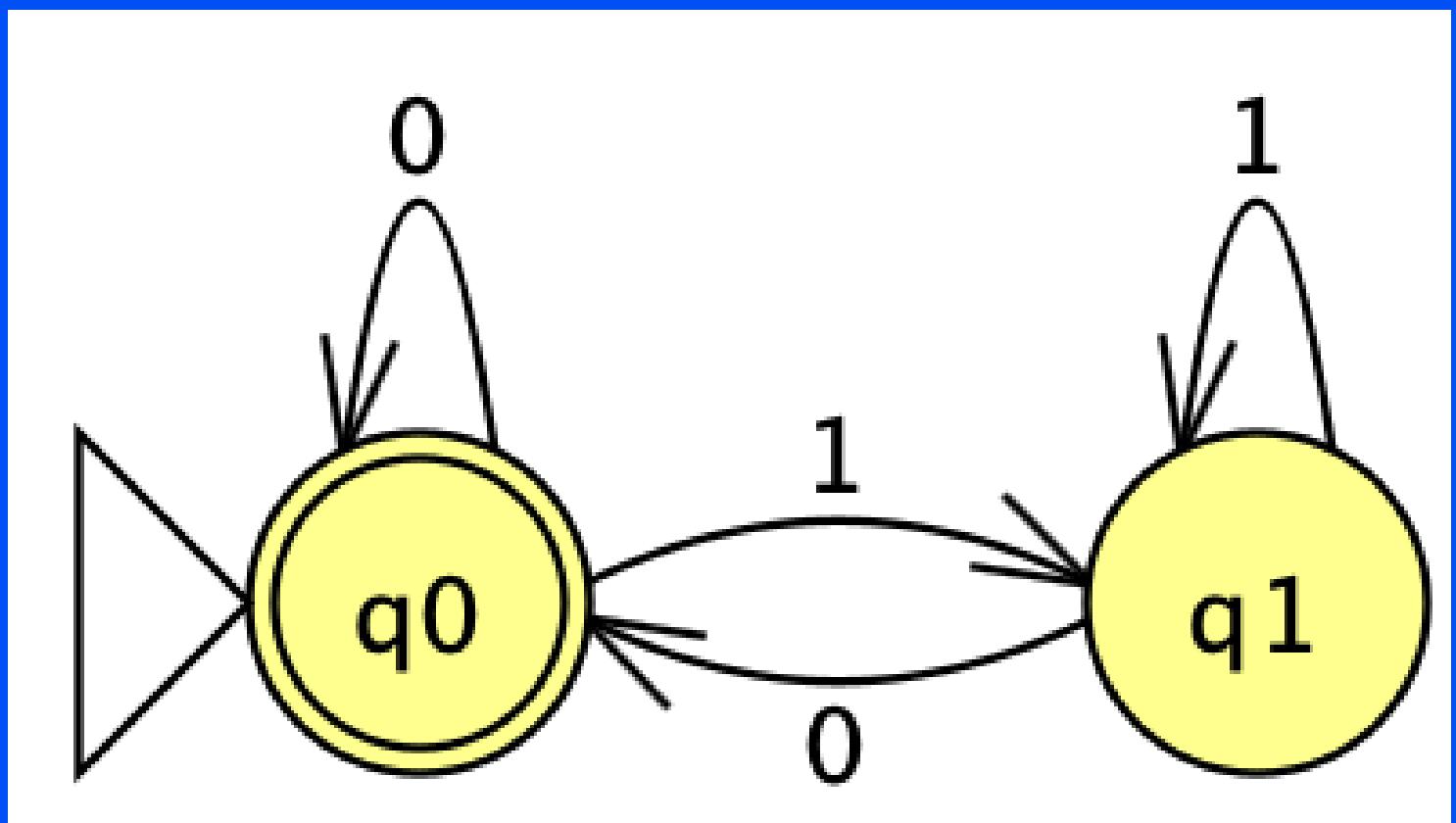
- Sea \mathbf{L} el lenguaje sobre $\Sigma = \{0,1\}$ de hileras que terminan en 0.
- Diseñe un DFA que reconozca a \mathbf{L} .



APROBADO

Ejemplo 3

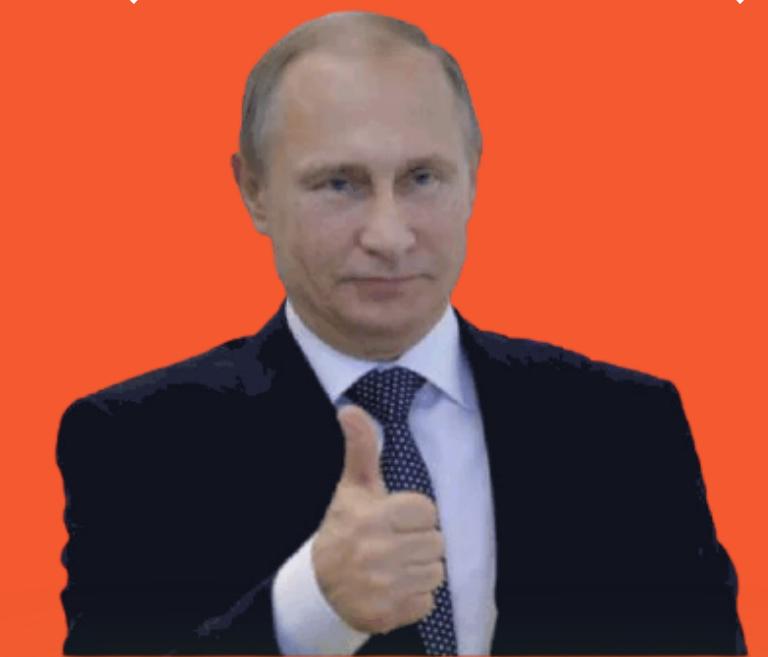
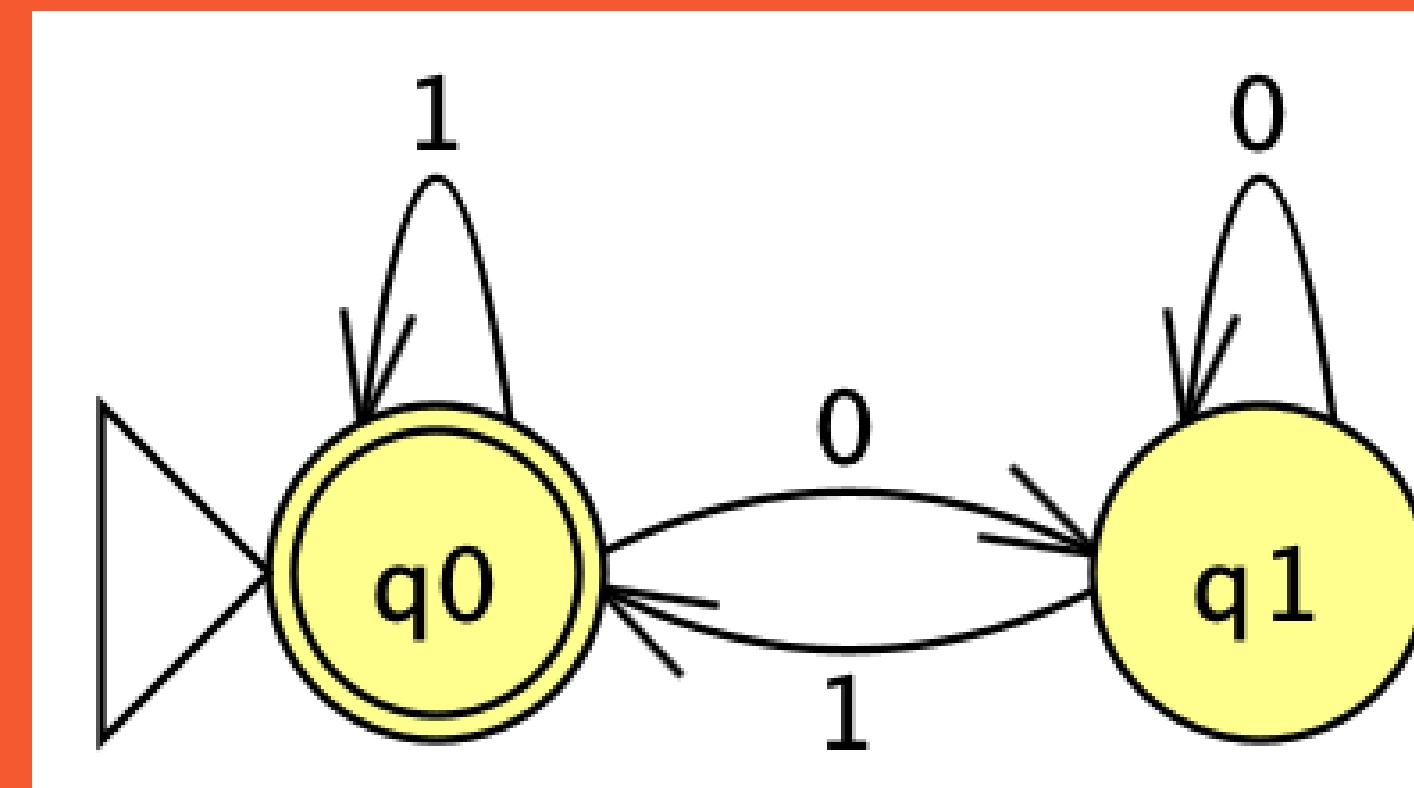
- Sea \mathbf{L} el lenguaje sobre $\Sigma = \{0,1\}$ de hileras que no terminan en 1.
- Diseñe un DFA que reconozca a \mathbf{L} .



APROBADO

Ejemplo 4

- Sea \mathbf{L} el lenguaje sobre $\Sigma = \{0,1\}$ de hileras que no terminan en 0.
- Diseñe un DFA que reconozca a \mathbf{L} .

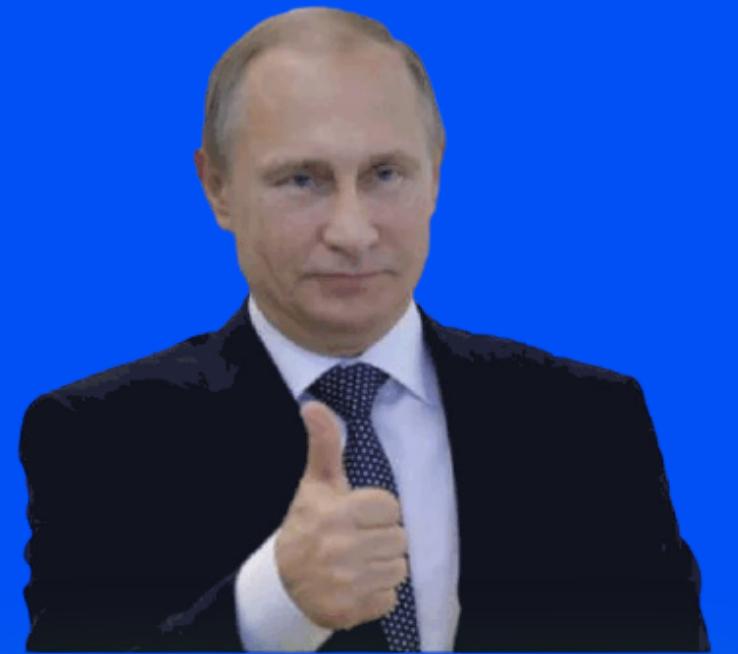
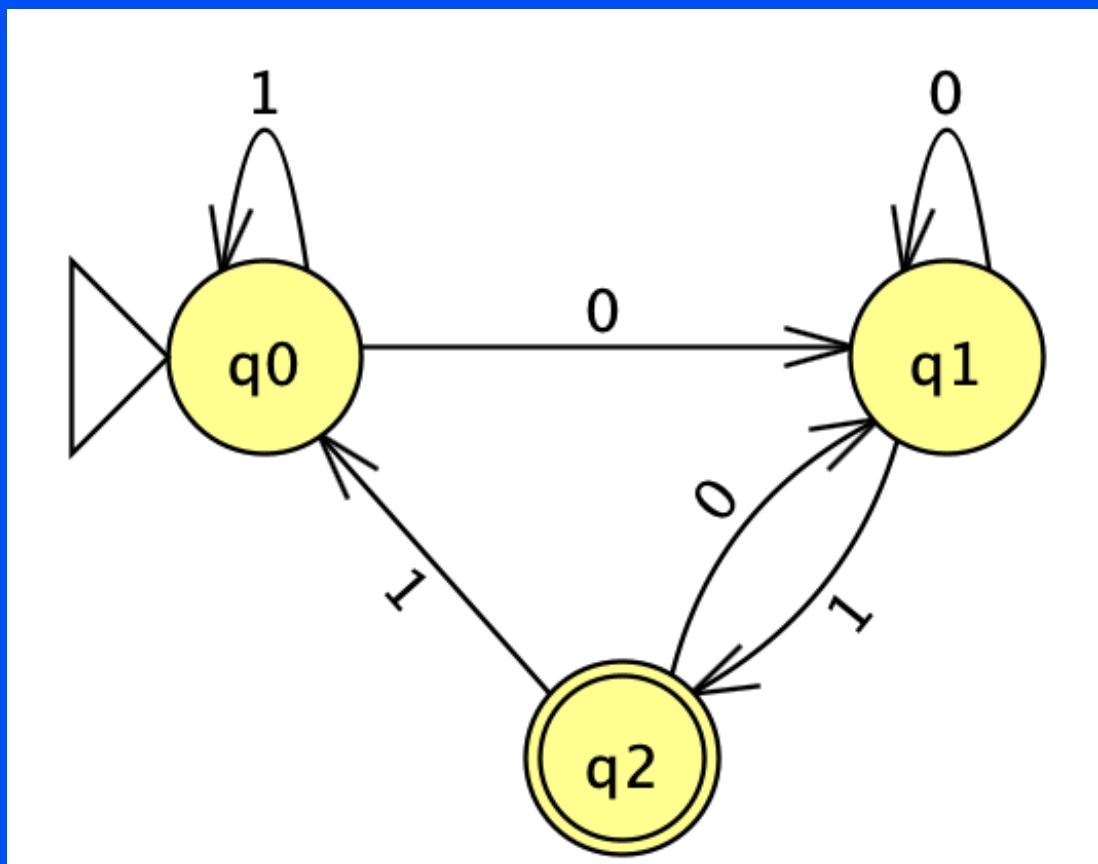


APROBADO

*complemento del ejemplo 2

Ejemplo 5

- Sea \mathbf{L} el lenguaje sobre $\Sigma = \{0,1\}$ de hileras que terminan en 01.
- Diseñe un DFA que reconozca a \mathbf{L} .



APROBADO

Apuntes Miércoles 15/10/20

Apuntador: Gerald Sánchez Fonseca



Ejemplo 5

Sea \mathcal{L} el lenguaje sobre $\Sigma = \{0,1\}$ de hileras que terminan en 01

- Diseñe un DFA que reconozca a \mathcal{L}



Ejemplo 6

Sea \mathcal{L} el lenguaje sobre $\Sigma = \{0,1\}$ de hileras que **no** terminan en 00

- Diseñe un DFA que reconozca a \mathcal{L}

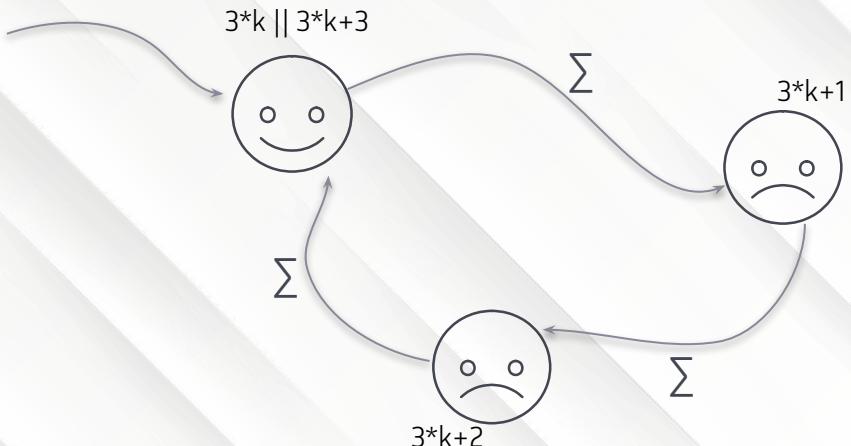
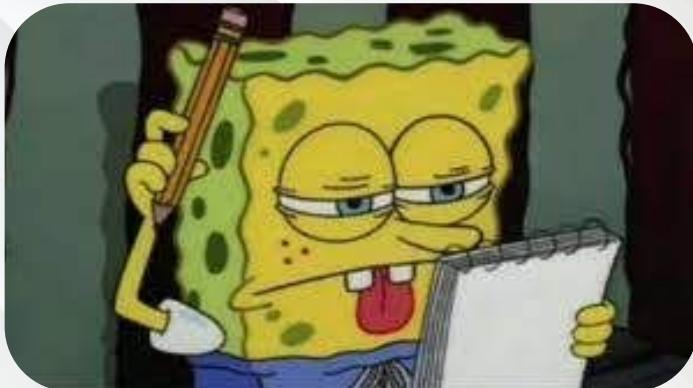


Ejemplo 7

 A partir de aqui, utilizaré Σ para referirme a todos los símbolos del alfabeto

Sea \mathcal{L} el lenguaje sobre $\Sigma = \{A,G,T,C\}$ de hileras cuya longitud sea múltiplo de 3. (Tener en cuenta que ϵ es aceptable, $0^*3 = 0$)

- ▷ Diseñe un DFA que reconozca a \mathcal{L}



Boletín informativo

En LaTeX hay una herramienta para poder diseñar distintos autómatas, el profesor nos informará sobre cual es :D

En caso de no informar es nuestro deber investigarlo

>:C

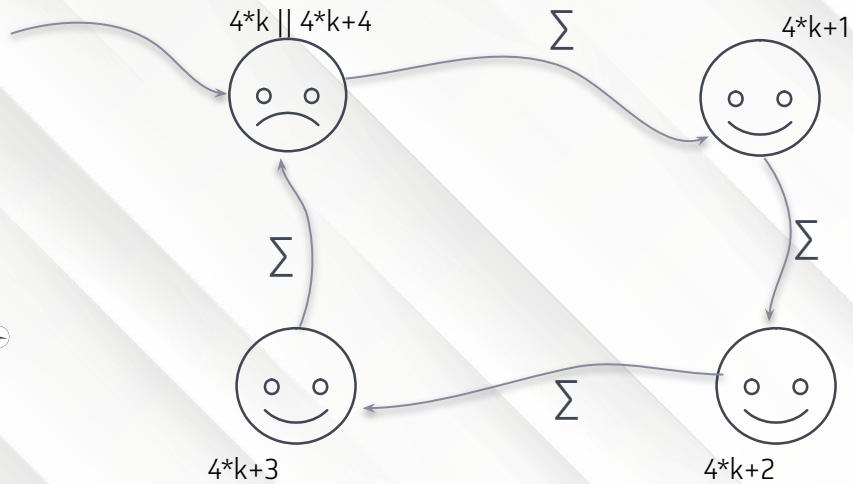


Ejemplo 8

Sea \mathcal{L} el lenguaje sobre $\Sigma = \{0,1\}$ de hileras cuya longitud **no** sea múltiplo de 4. (Tener en cuenta que ϵ **no** es aceptable, $0^*4 = 0$)

- ▷ Diseñe un DFA que reconozca a \mathcal{L}

Una inocente criatura
que cree ya saber
todo de autómatas



Ejemplo 9

Sea \mathcal{L} el lenguaje sobre $\Sigma = \{A, T, G, C\}$ de hileras que no contengan la sub hilera “GAGA”. *Ojo que hablamos de contener la sub hilera, da igual su ubicación dentro de la hilera*

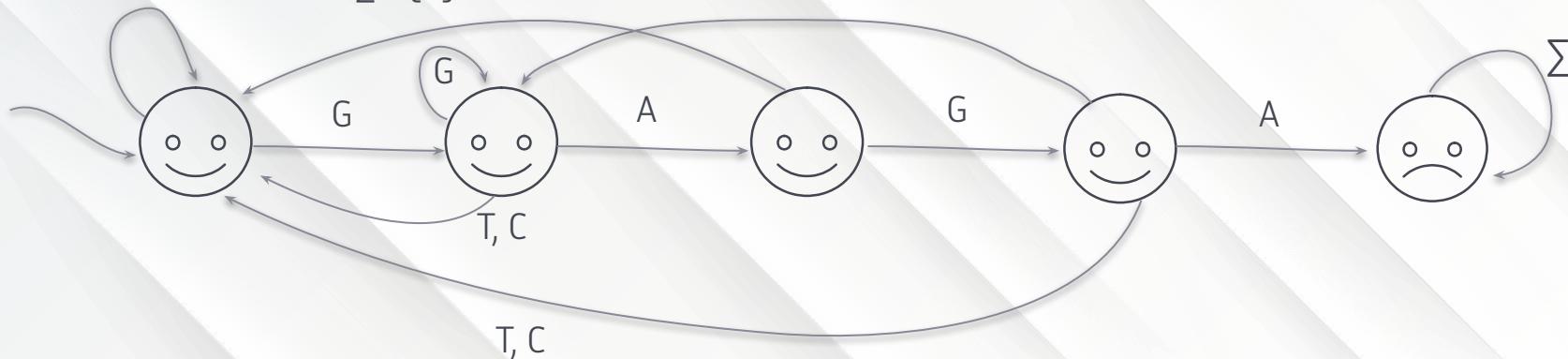


▷ Diseñe un DFA que reconozca a \mathcal{L}

$\Sigma - \{G\}$

$\Sigma - \{G\}$

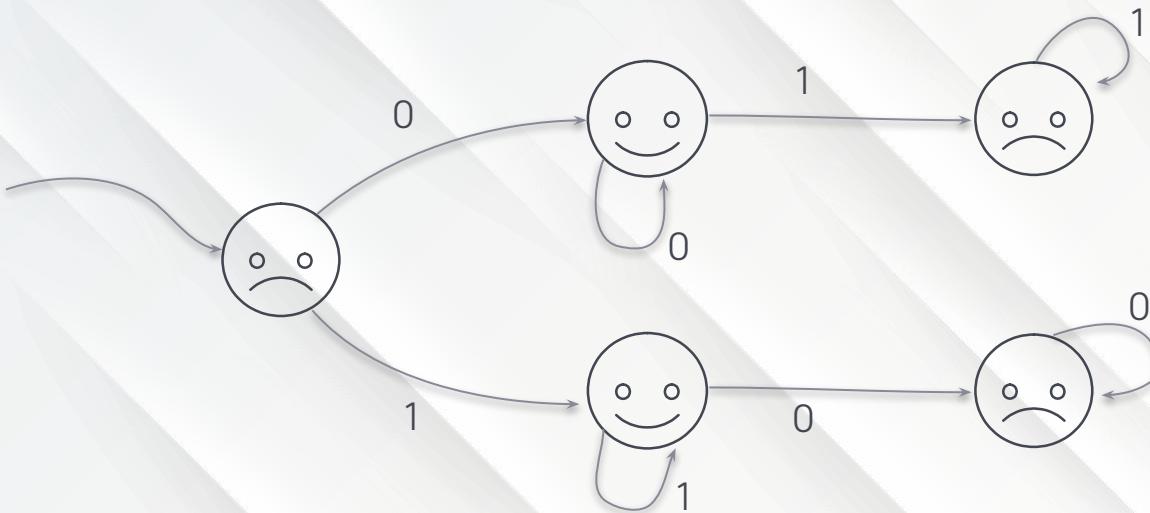
G



Ejemplo 10

Sea \mathcal{L} el lenguaje sobre $\Sigma = \{0,1\}$ de hileras terminen en el mismo símbolo con el cual iniciaron.

- Diseñe un DFA que reconozca a \mathcal{L}



10101000	x
10100001	✓
ϵ	x



Ejemplo 11

Sea \mathcal{L} el lenguaje sobre $\Sigma = \{0,1\}$ de hileras terminen **no** en el mismo símbolo con el cual iniciaron.

- Diseñe un DFA que reconozca a \mathcal{L}



101111000	✓
111000101	✗
ϵ	✓

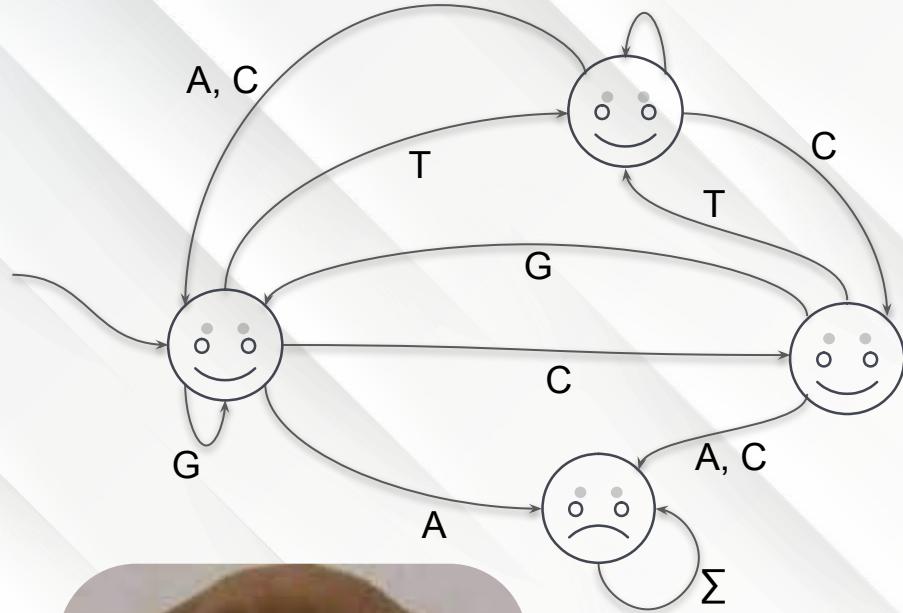


Pd: si son diferentes
cuidado

Ejemplo 12

Sea \mathcal{L} el lenguaje sobre $\Sigma = \{A, T, G, C\}$ de hileras que contengan la sub hilera “CC” y donde “A” sea inmediatamente precedida de una “T”

- Diseñe un DFA que reconozca a \mathcal{L}



Ejemplo 13 - Análisis

Sea \mathcal{L} el lenguaje sobre $\Sigma = \{0,1,2,3,4,5,6,7,8,9\}$ de hileras que representen el número en base 10 divisible entre 3

Si divido un número en 3, sus posibles residuos son 0,1,2

¿Qué significa que sea divisible entre 3?

- ▀ Ser $3k$
- ▀ Residuo 0
- ▀ Residuo 1 y 2 es no ser divisible

¿Qué es ser base 10?

- ▀ Cada posición corresponde a una potencia de 10

Si el número actual es "n" y le concatenamos un nuevo símbolo "q" a la derecha; Nuevo valor = $(n*10)+q$

Si a un número divisible por 3 le concateno un 0 a la derecha. ¿Cuál es su residuo?

Antes: $3*k$

Después: $(3*k)*10+0 = 3(k*10) = 3*k'$

Sigue siendo divisible por 3



Mucho texto

$3k+0$



Nuevo Dígito	Nuevo Valor	Residuo
$n+0$	$(3*k)*10+0$	0
$n+1$	$(3*k)*10+1$	1
$n+2$	$(3*k)*10+2$	2
$n+3$	$(3*k)*10+3$	0
$n+4$	$(3*k)*10+4$	1
$n+5$	$(3*k)*10+5$	2
$n+6$	$(3*k)*10+6$	0
$n+7$	$(3*k)*10+7$	1
$n+8$	$(3*k)*10+8$	2
$n+9$	$(3*k)*10+9$	0

$3k+1$



Nuevo Dígito	Nuevo Valor	Residuo
$n+0$	$((3*k)+1)*10+0$	1
$n+1$	$((3*k)+1)*10+1$	2
$n+2$	$((3*k)+1)*10+2$	0
$n+3$	$((3*k)+1)*10+3$	1
$n+4$	$((3*k)+1)*10+4$	2
$n+5$	$((3*k)+1)*10+5$	0
$n+6$	$((3*k)+1)*10+6$	1
$n+7$	$((3*k)+1)*10+7$	2
$n+8$	$((3*k)+1)*10+8$	0
$n+9$	$((3*k)+1)*10+9$	1

$3k+2$



Nuevo Dígito	Nuevo Valor	Residuo
$n+0$	$((3*k)+2)*10+0$	2
$n+1$	$((3*k)+2)*10+1$	0
$n+2$	$((3*k)+2)*10+2$	1
$n+3$	$((3*k)+2)*10+3$	2
$n+4$	$((3*k)+2)*10+4$	0
$n+5$	$((3*k)+2)*10+5$	1
$n+6$	$((3*k)+2)*10+6$	2
$n+7$	$((3*k)+2)*10+7$	0
$n+8$	$((3*k)+2)*10+8$	1
$n+9$	$((3*k)+2)*10+9$	2

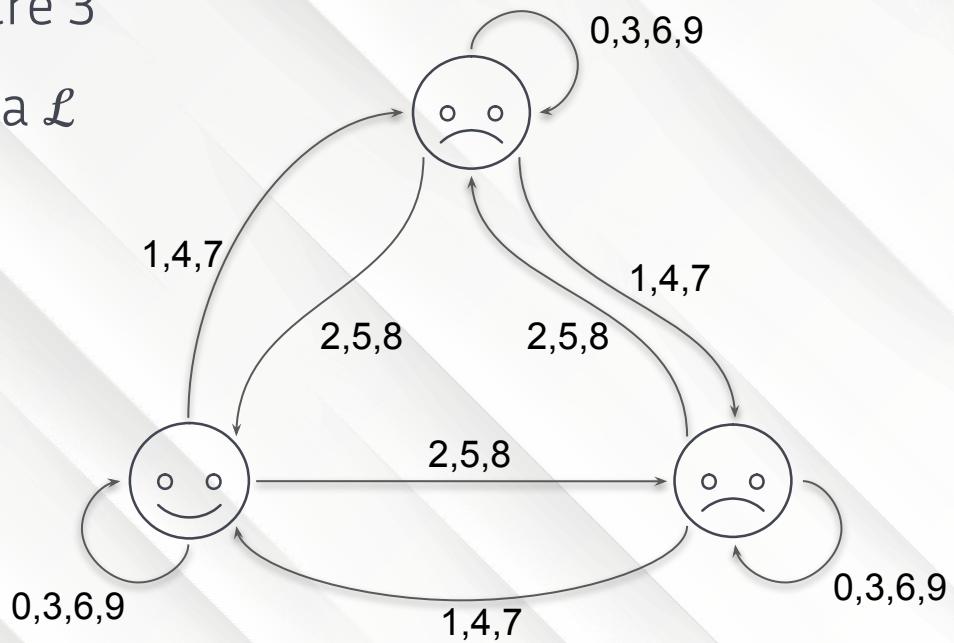
Felicidades compañero
Acaba de llegar a la
mitad de los apuntes,
descansa aquí



Ejemplo 13

Sea L el lenguaje sobre $\Sigma = \{0,1,2,3,4,5,6,7,8,9\}$ de hileras que representen el número en base 10 divisible entre 3

- Diseñe un DFA que reconozca a \mathcal{L}



Ejemplo 14 - Análisis

Sea L el lenguaje sobre $\Sigma = \{0,1,2\}$ de hileras en base 3 que no sean divisibles entre 5. (base 10)

Si divido un número en 5, sus posibles residuos son 0,1,2,3,4

¿Qué significa que sea divisible entre 5?

- ▀ Ser $5k$
- ▀ Residuo 0
- ▀ Residuo 1, 2, 3 ó 4 es no ser divisible

¿Qué es ser base 3?

- ▀ Cada posición corresponde a una potencia de 3

Si el número actual es “ n ” y le concatenamos un nuevo símbolo “ q ” a la derecha; Nuevo valor = $(n*3)+q$

Si a un número en base 3 y divisible por 5 le concateno un 0 a la derecha. ¿Cuál es su residuo?

Antes: $5*k$

Después: $(5*k)*10+0 = 5(k*10) = 5*k'$

Sigue siendo divisible por 5



Ejemplo 14 - Análisis 2

Si el número actual es de la forma $(5*k) + 0$ y le concateno un dígito,
¿Cuál sería su residuo?

Nuevo Dígito	Nuevo Valor	Residuo
0	$(5*k)*3+0$	0
1	$(5*k)*3+1$	1
2	$(5*k)*3+2$	2

Ejemplo 14 - Análisis 3

Si el número actual es de la forma $(5*k) + 1$ y le concateno un dígito,
¿Cuál sería su residuo?

Nuevo Dígito	Nuevo Valor	Residuo
0	$((5*k)+1)*3+0$	3
1	$((5*k)+1)*3+1$	4
2	$((5*k)+1)*3+2$	0

Ejemplo 14 - Análisis 4

Si el número actual es de la forma $(5*k) + 2$ y le concateno un dígito, ¿Cuál sería su residuo?

Nuevo Dígito	Nuevo Valor	Residuo
0	$((5*k)+2)*3+0$	1
1	$((5*k)+2)*3+1$	2
2	$((5*k)+2)*3+2$	3

Ejemplo 14 - Análisis 5

Si el número actual es de la forma $(5*k) + 3$ y le concateno un dígito, ¿Cuál sería su residuo?

Nuevo Dígito	Nuevo Valor	Residuo
0	$((5*k)+3)*3+0$	4
1	$((5*k)+3)*3+1$	0
2	$((5*k)+3)*3+2$	1

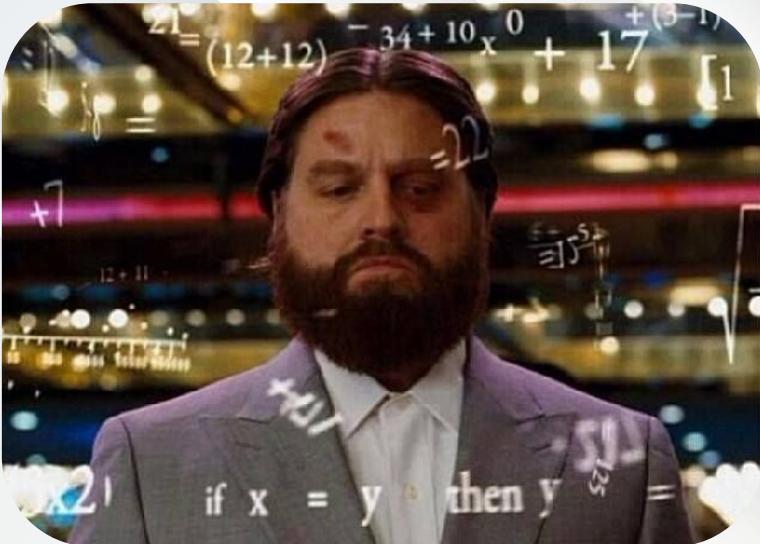
Ejemplo 14 - Análisis 6

Si el número actual es de la forma $(5*k) + 4$ y le concateno un dígito,
¿Cuál sería su residuo?

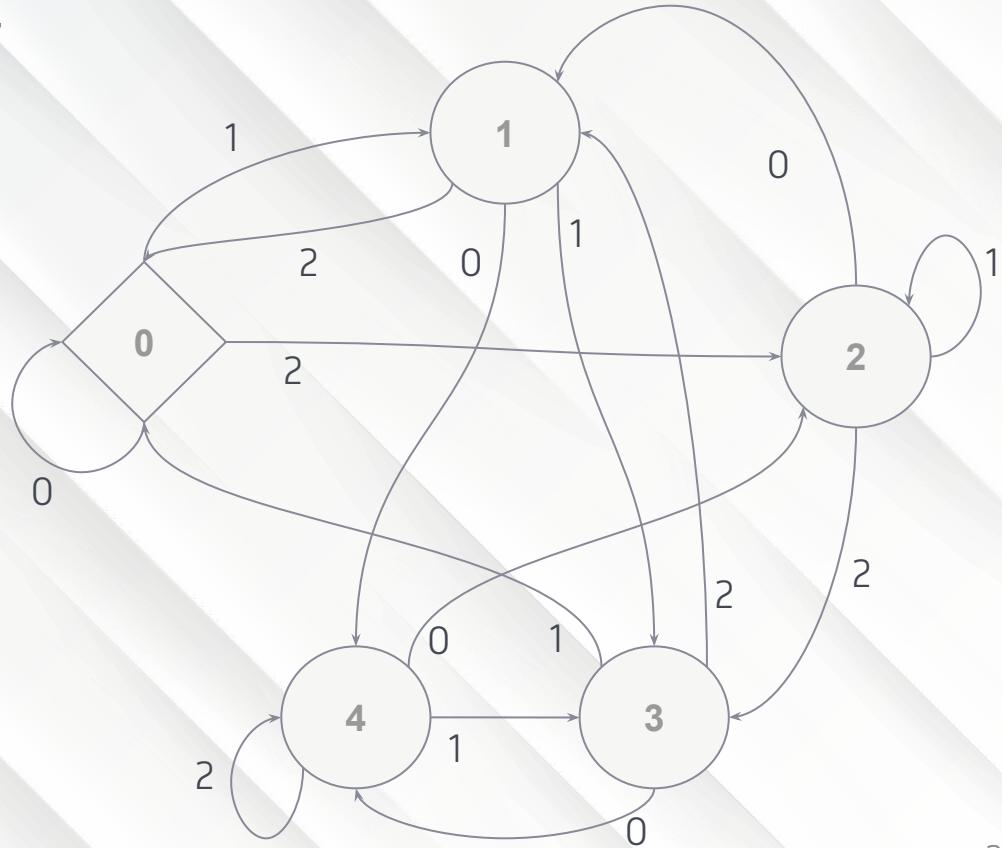
Nuevo Dígito	Nuevo Valor	Residuo
0	$((5*k)+4)*3+0$	2
1	$((5*k)+4)*3+1$	3
2	$((5*k)+4)*3+2$	4

Ejemplo 14

Construcción

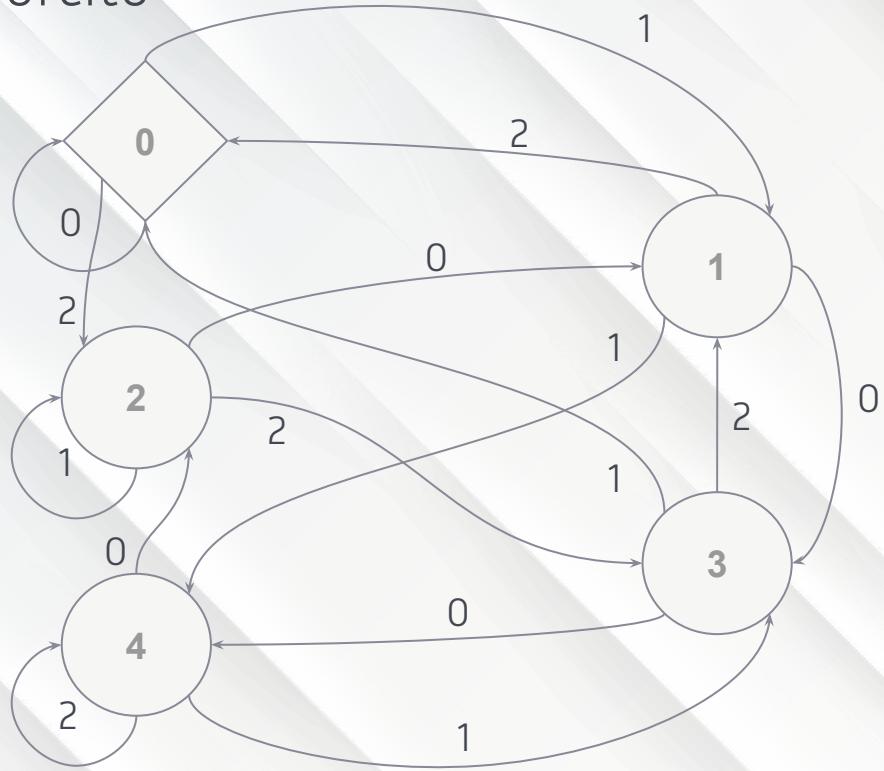


Círculo aceptado
Rombo rechazado



Ejemplo 14

Mejorcito



¿Dónde está la entrada
Drake?

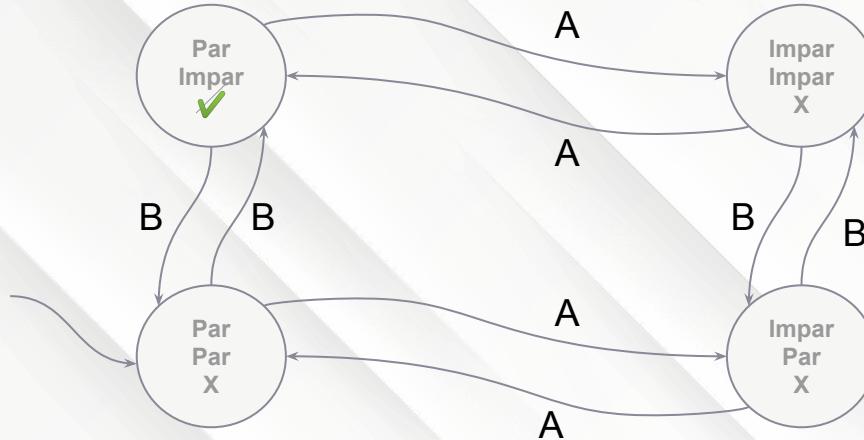
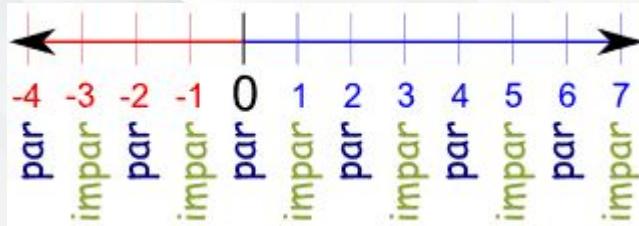


Ejemplo 15

Sea \mathcal{L} el lenguaje sobre $\Sigma = \{A, B\}$ de hileras que contengan un número par de A's y un número impar de B's

- Diseñe un DFA que reconozca a \mathcal{L}

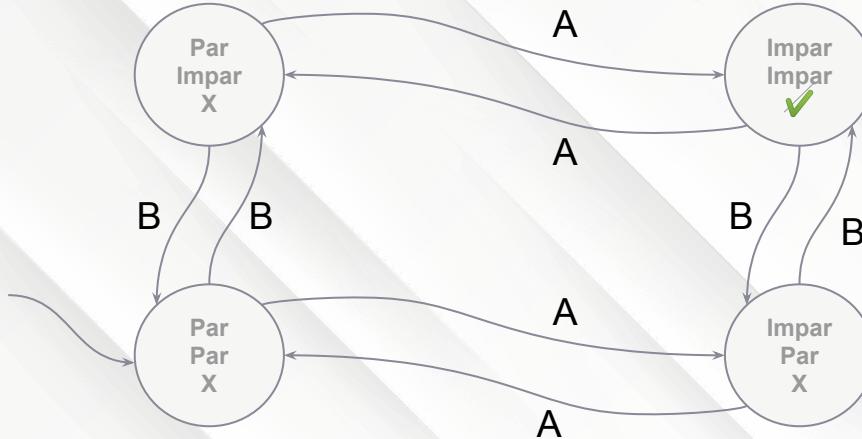
A tener en cuenta:



Ejemplo 16

Sea \mathcal{L} el lenguaje sobre $\Sigma = \{A, B\}$ de hileras que contengan un número impar de As y un número impar de B's

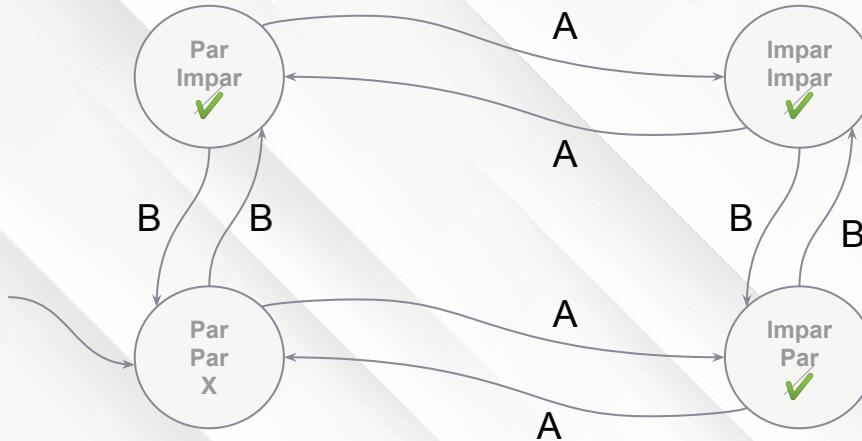
- Diseñe un DFA que reconozca a \mathcal{L}



Ejemplo 17

Sea \mathcal{L} el lenguaje sobre $\Sigma = \{A, B\}$ de hileras que contengan un número impar de As o un número impar de B's

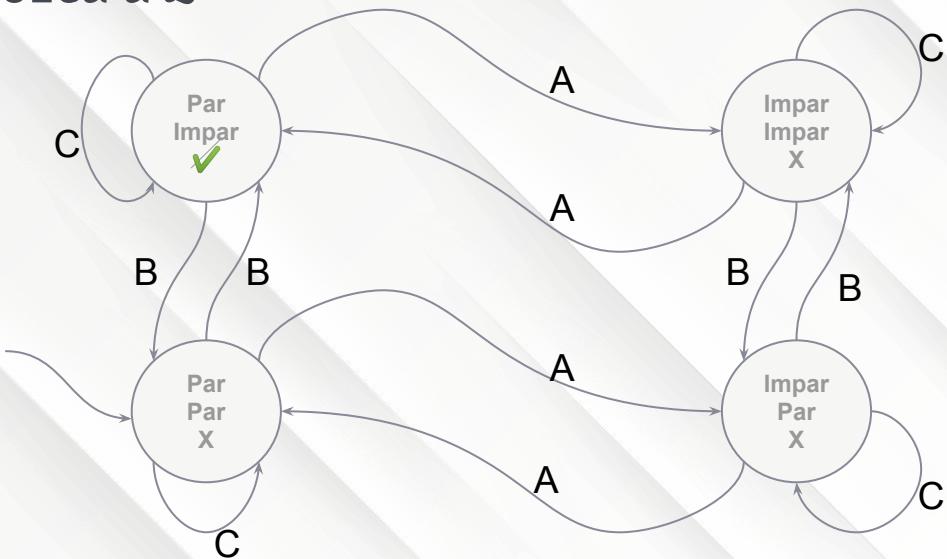
- ▷ Diseñe un DFA que reconozca a \mathcal{L}



Ejemplo 18

Sea \mathcal{L} el lenguaje sobre $\Sigma = \{A, B, C\}$ de hileras que contengan un número par de A's o un número impar de B's

- Diseñe un DFA que reconozca a \mathcal{L}



Fin

Pueden ir en paz ;3



Apuntes

16/10/2020

Edxon Rodríguez Aguilar

2015007575

Notas

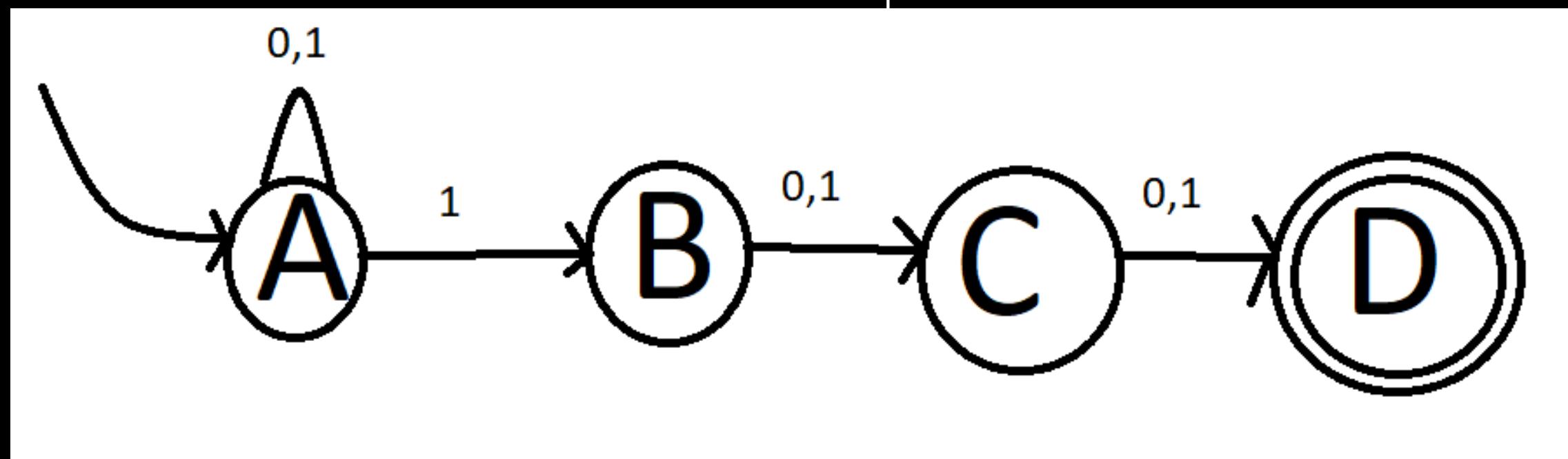
- Este es el ultimo material para el viernes
- Se acabaron los Pinkers (salen en el examen)

Automatas No Determinísticos de Estados Finitos

Ejemplo 1

Sea L el lenguaje sobre $\Sigma = \{0,1\}$ de hileras tales que el tercer simbolo de derecha a izquierda sea un 1

Diseñe un automata que reconozca L



El nodo A apunta a 2 resultados con 1

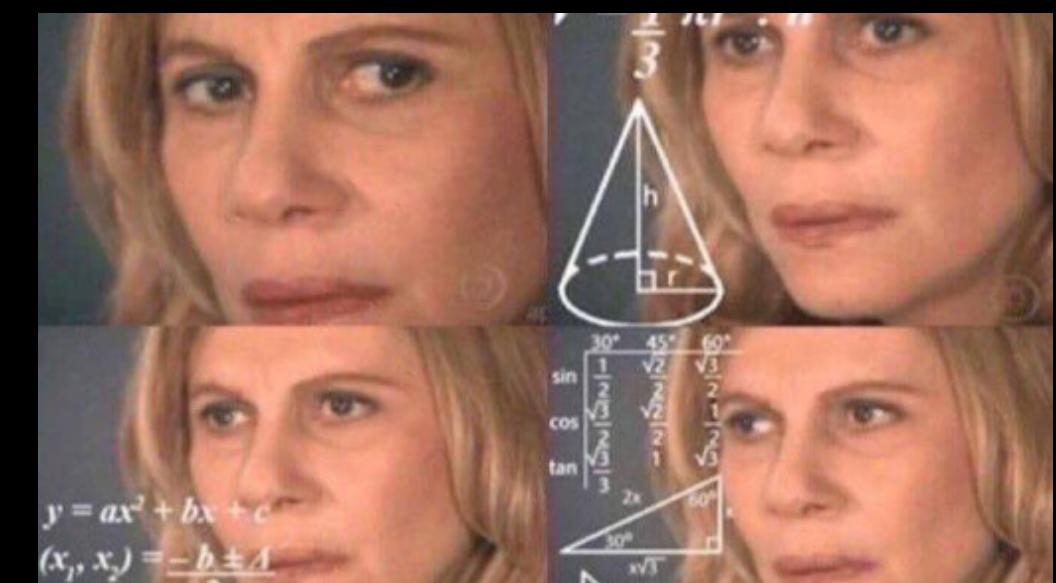
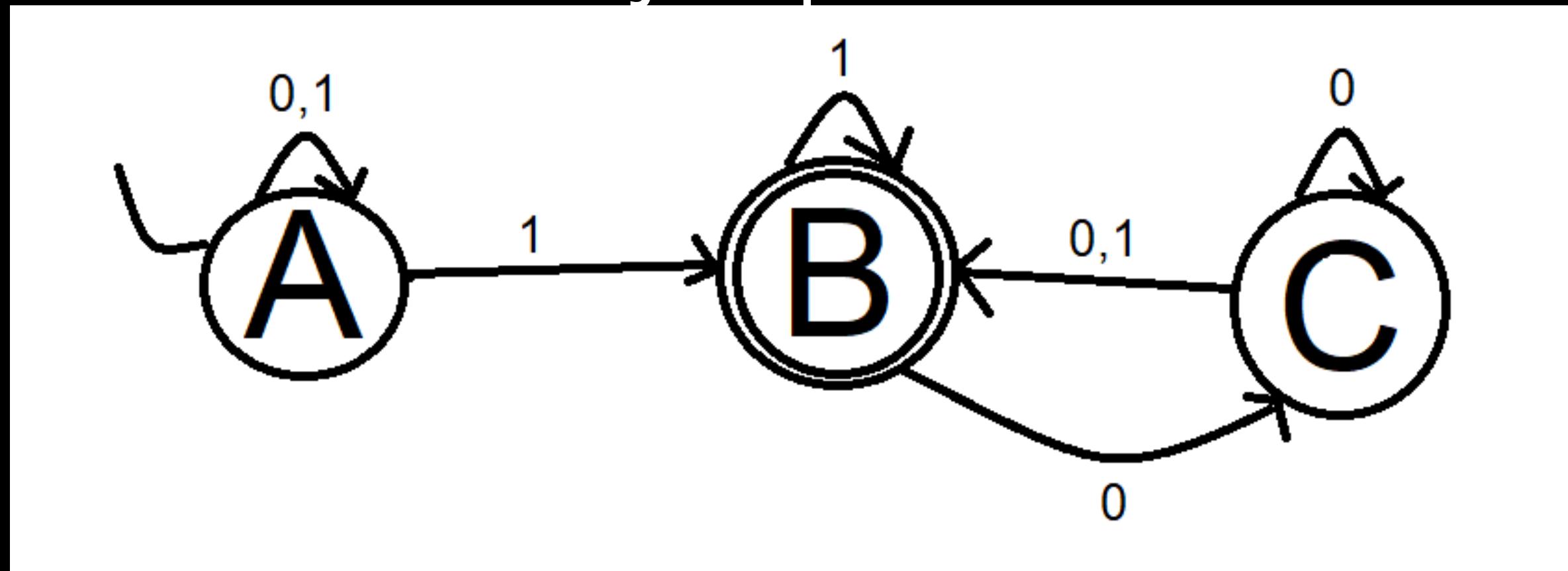
¿Cómo sabe que ese 1 es el necesario para irse a B?



Debemos domesticarlo!

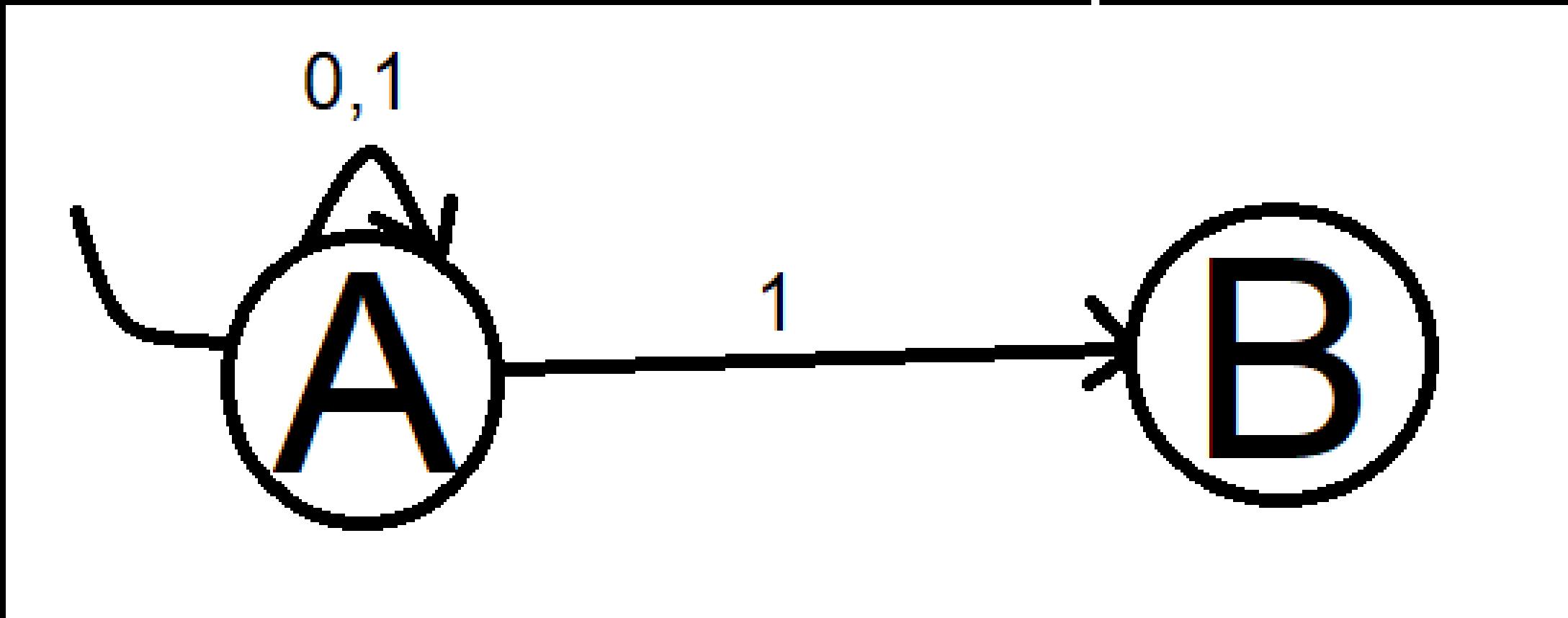


Ejemplo 2



No Determinismo

Existen estados donde un mismo símbolo tiene varias transiciones posibles



Non-Deterministic Finite States Automaton (NFA)

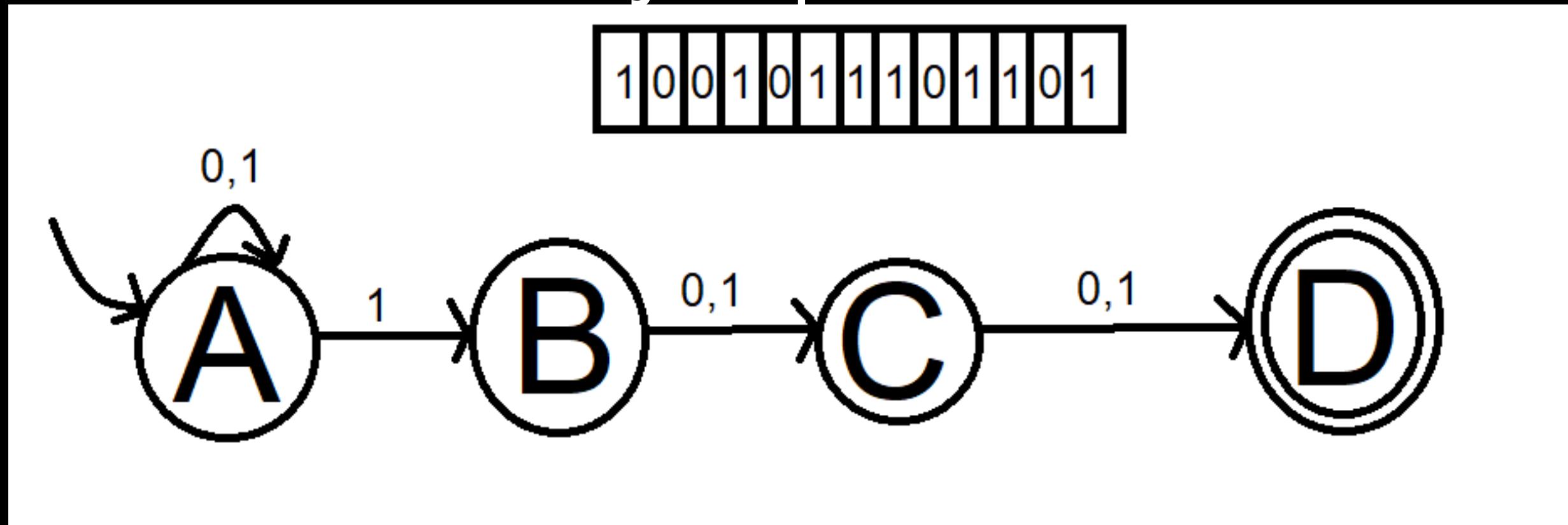
Consisten en: Tomar la decisión correcta



Aceptación en NFAs

- Al procesar una hilera podría haber varias rutas posibles
(No determinismo)
 - La hilera se podría acabar en diferentes lugares
- Un NFA rechaza a una hilera si no existe al menos una ruta que partiendo del estado inicial y avanzando con las transiciones disponibles llegue a un estado de aceptación
- Un NFA acepta a una hilera si existe al menos una ruta que partiendo del estado inicial y avanzando con las transiciones disponibles llega a un estado de aceptación

Ejemplo 3



Mientras recorremos la hilera, ponemos un dedo en cada posición

Los NFAs son unos
descarados, tenemos que
dar vuelta y vuelta hasta que
se acabe la hilera



Definición formal

Un NFA es un quinteto $M = (Q, \Sigma, \delta, q_0, F)$ donde

- Q es un conjunto finito de estados

- Σ es un alfabeto

- $\delta: Q \times \Sigma \rightarrow P(Q)$ es la función de transición

- $q_0 \in Q$ es el estado inicial

- $F \subseteq Q$ conjunto de estados de aceptación

Teorema

Sea $M = (Q, \Sigma, \delta, q_0, F)$ un NFA. Siempre existen un DFA $M' = (P(Q), \Sigma, \delta', \{q_0\}, F')$ que reconoce exactamente el mismo lenguaje que M

Todo NFA tiene un DFA equivalente que reconoce el mismo lenguaje

Observaciones

- El alfabeto Σ es el mismo en las dos máquinas
- Los estados de M' son subconjuntos del conjunto Q de M
- Hay que calcular la nueva función de transición δ
- F' es un conjunto de subconjuntos del conjunto Q de M



EL PARA QUE COSA DE QUIEN?

Cálculo de nueva función de transición

1. El estado inicial de M' es $\{q_0\}$, o sea un conjunto que contiene al estado inicial de M , esta es la primera fila de la tabla asociada a δ'
2. En la tabla asociada a la función de transición δ' coloque en la columna de todo simbolo de Σ el conjunto de todos los lugares a los que se puede llegar desde q_0 con dicho simbolo (el conjunto vacío es una opción válida)
3. Tome el primero de estos subconjuntos y póngalo como una nueva fila de la tabla de δ'
4. Llene de columnas con el conjunto de todos los destinos a los que se llega con el símbolo de la columna desde todos los miembros de este subconjunto
5. Repita mientras queden subconjuntos pendientes



- Vuelvanla a ver despues de ver el ejemplo
- Por ahora no lo entiendan
- Fila es estado, columna es simbolo de alfabeto

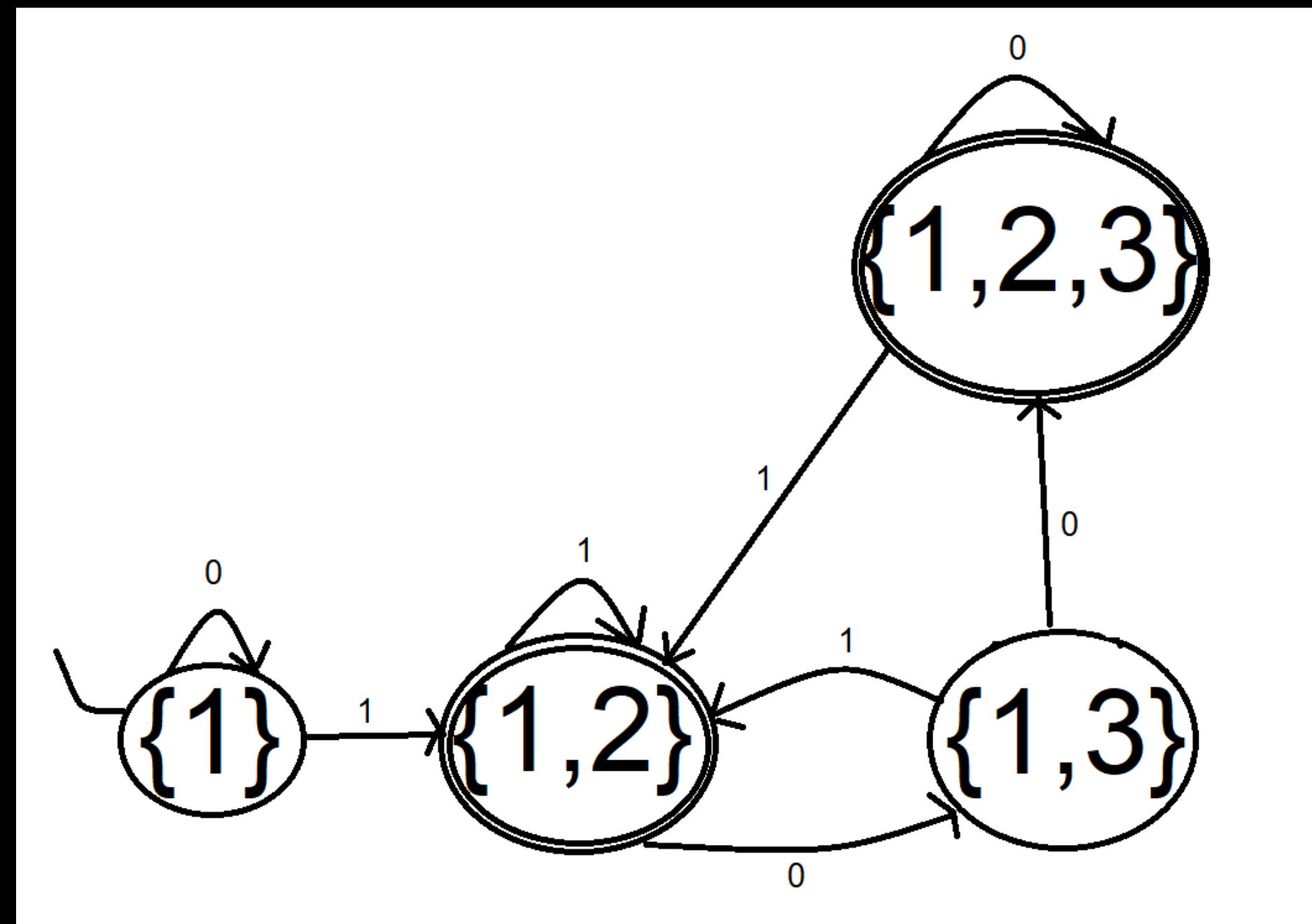


No lo entenderías

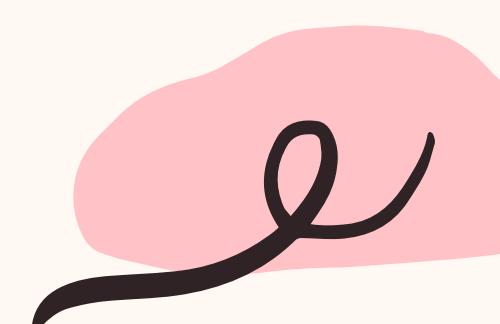
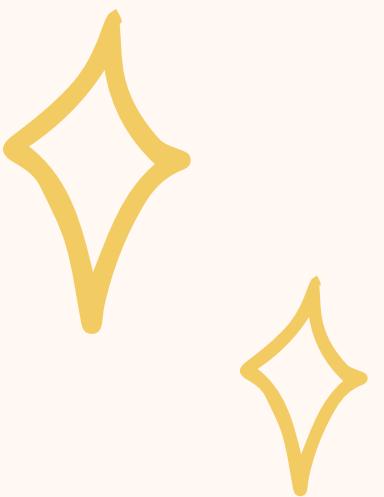
Ejemplo

	0	1
{1}	{1}	{1,2}
{1,2}	{1,3}	{1,2}
{1,3}	{1,2,3}	{1,2}
{1,2,3}	{1,3,2}	{1,2}

Recuerden que el máximo de estados puede ser 2^N
Y preparese para lo peor

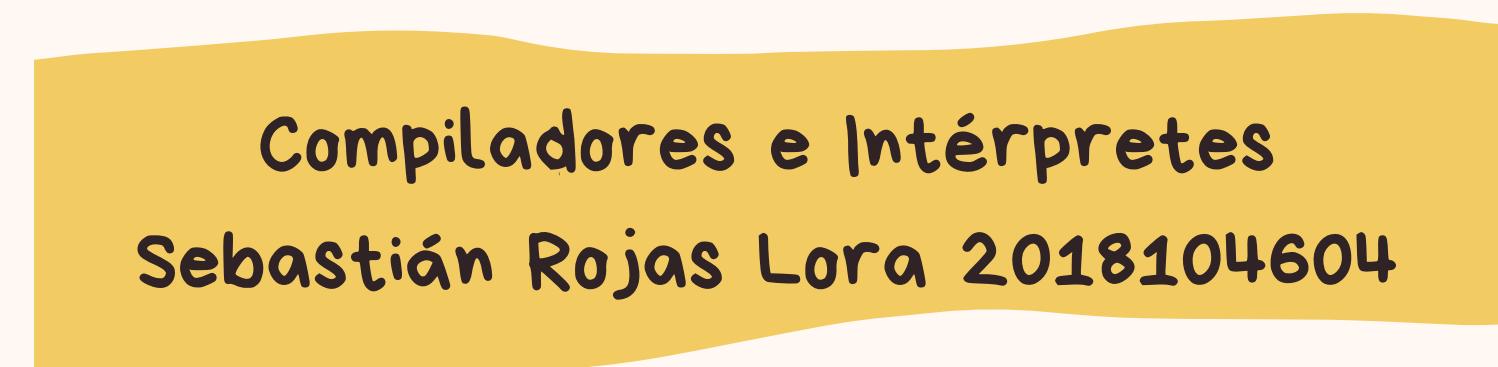


Grafo de aceptación



APUNTES

16/10/2020



Compiladores e Intérpretes

Sebastián Rojas Lora 2018104604





ANÁLISIS LÉXICO

AUTÓMATAS NO DETERMINÍSTICOS DE ESTADOS FINITOS



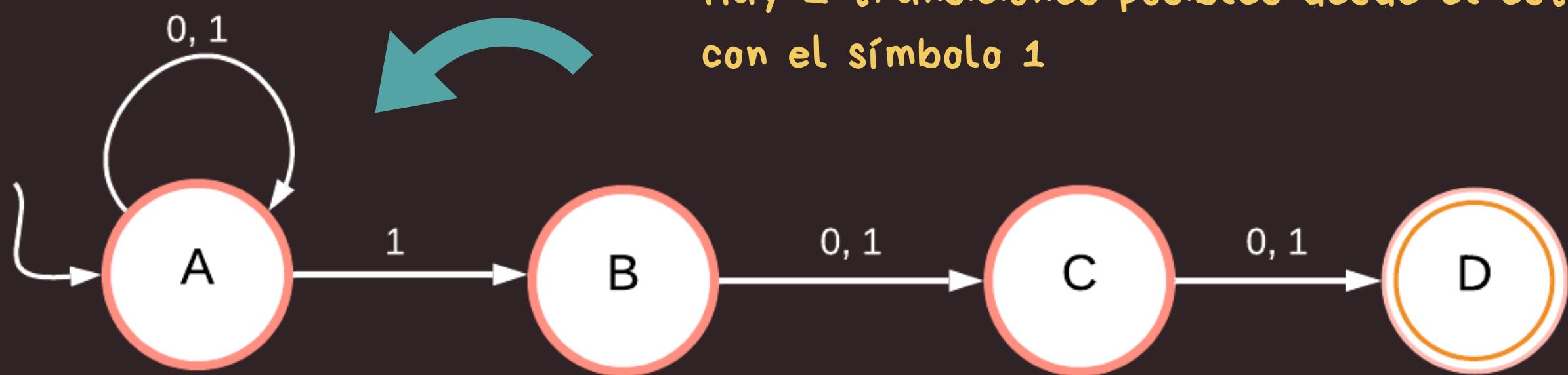
También conocido como
Non-Deterministic finite
automata. (NFA)



Ejemplo 1

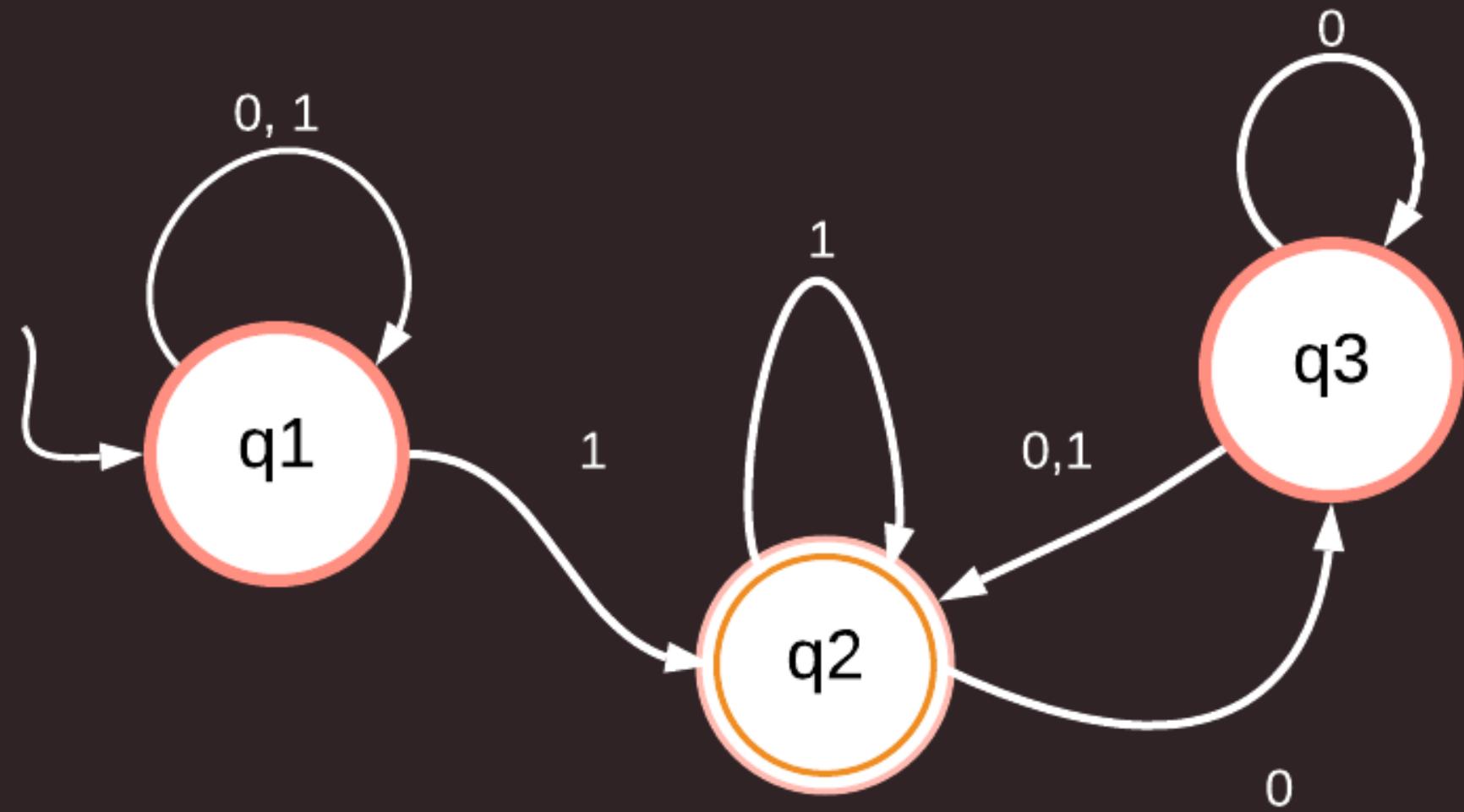
- Sea L el lenguaje sobre $\Sigma = \{0, 1\}$ de hileras que el tercer símbolo de derecha a izquierda sea un 1
- Diseñe un autómata que reconozca a L
- Antes habíamos visto los DFA's pero viendo este autómata vemos algún problema?

La respuesta siempre es por "Arte de magia"



- Hay 2 transiciones posibles desde el estado A con el símbolo 1

Ejemplo 2

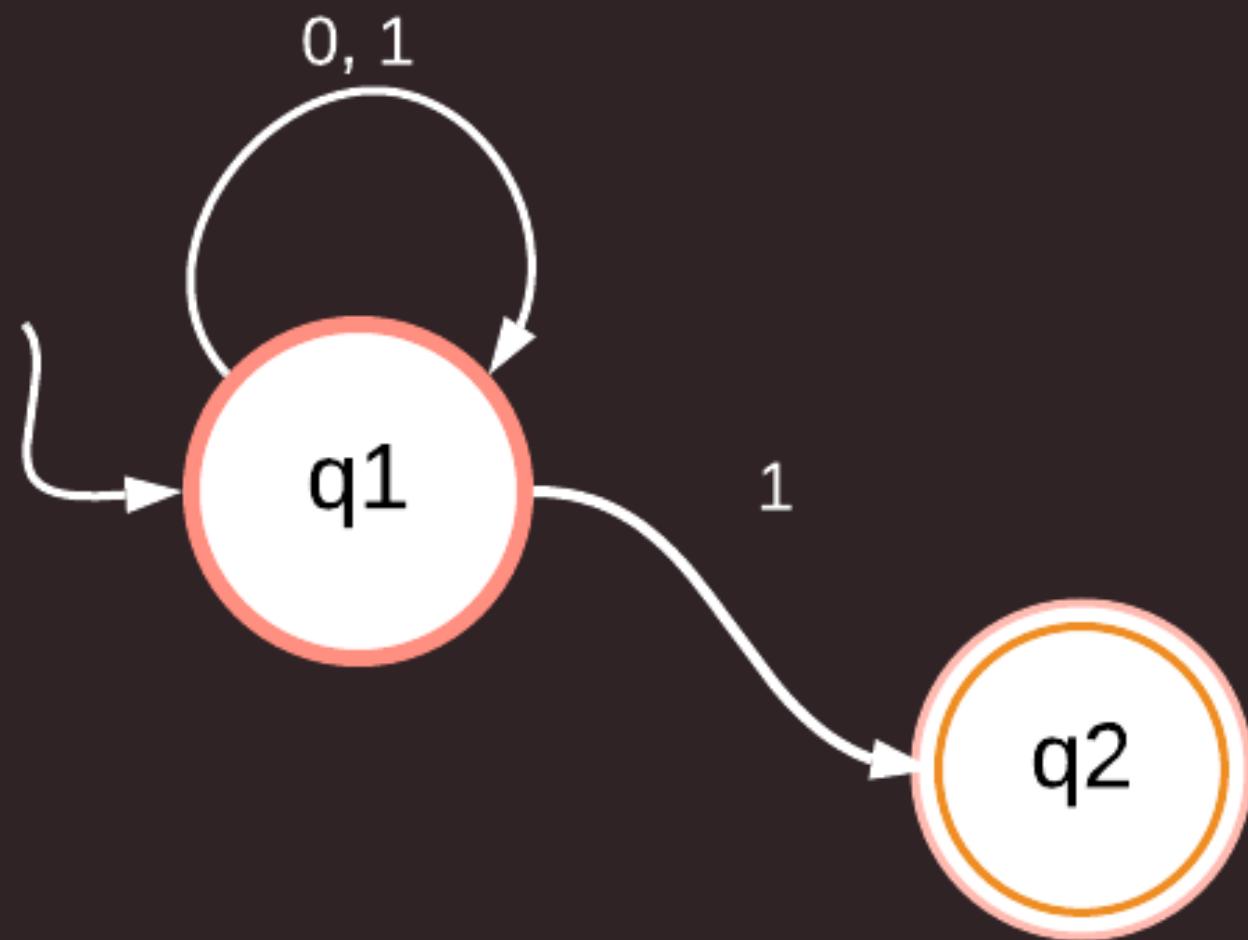


•
•



NO DETERMINISMO

- Existen estados donde un mismo símbolo tiene varias transiciones posibles

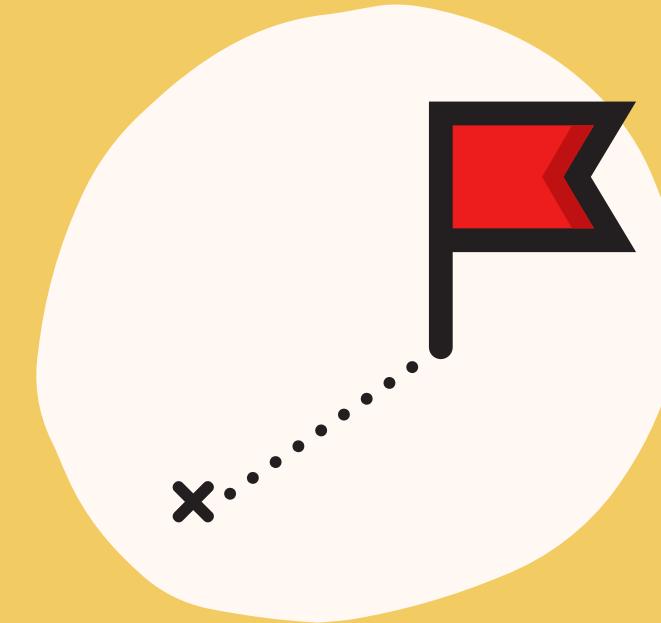


También conocido como
Non-Deterministic finite
automata. (NFA)

ACEPTACIÓN EN NFA



La hilera podría acabar en diferentes lugares



Un NFA **rechaza** a una hilera si no existe **ninguna ruta** partiendo del estado inicial y avanzando con las transiciones disponibles

Al procesar una hilera podría haber varias rutas posibles



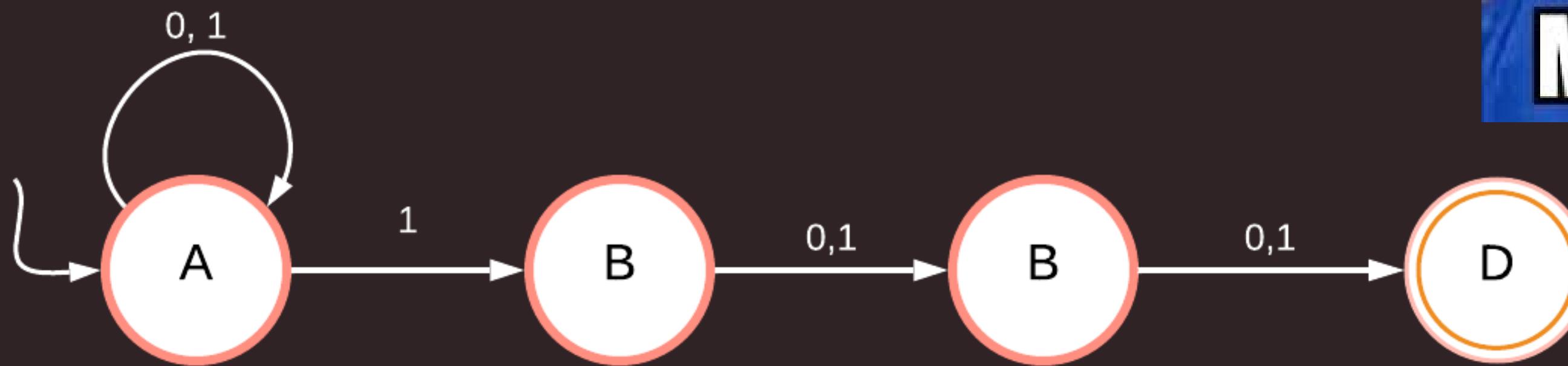
Un NFA **acepta** una hilera si existe **al menos una ruta** partiendo del estado inicial y avanzando con las transiciones disponibles



Ejemplo 3



1001011101101



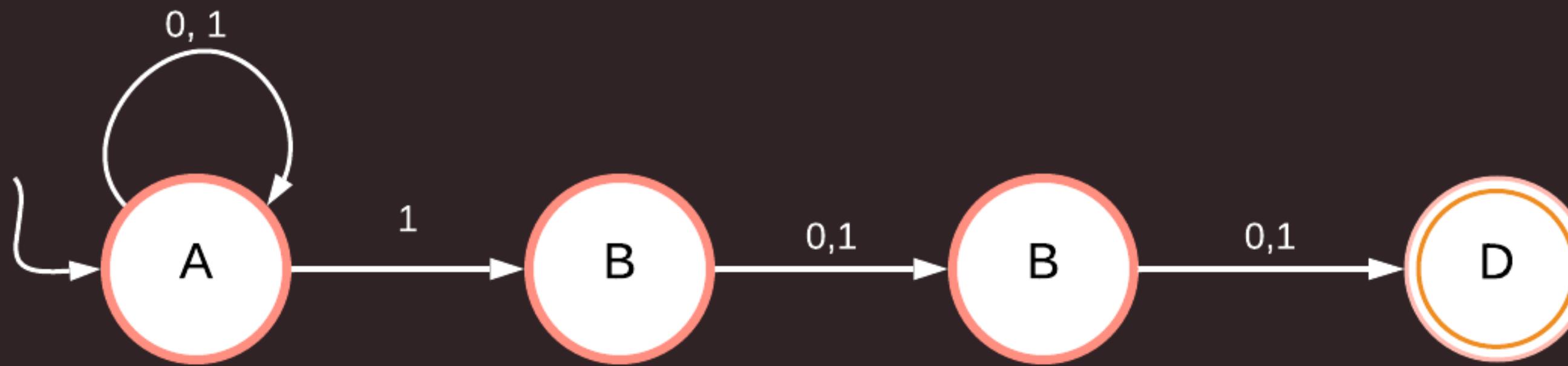
Se acepta!



Ejemplo 4



10010

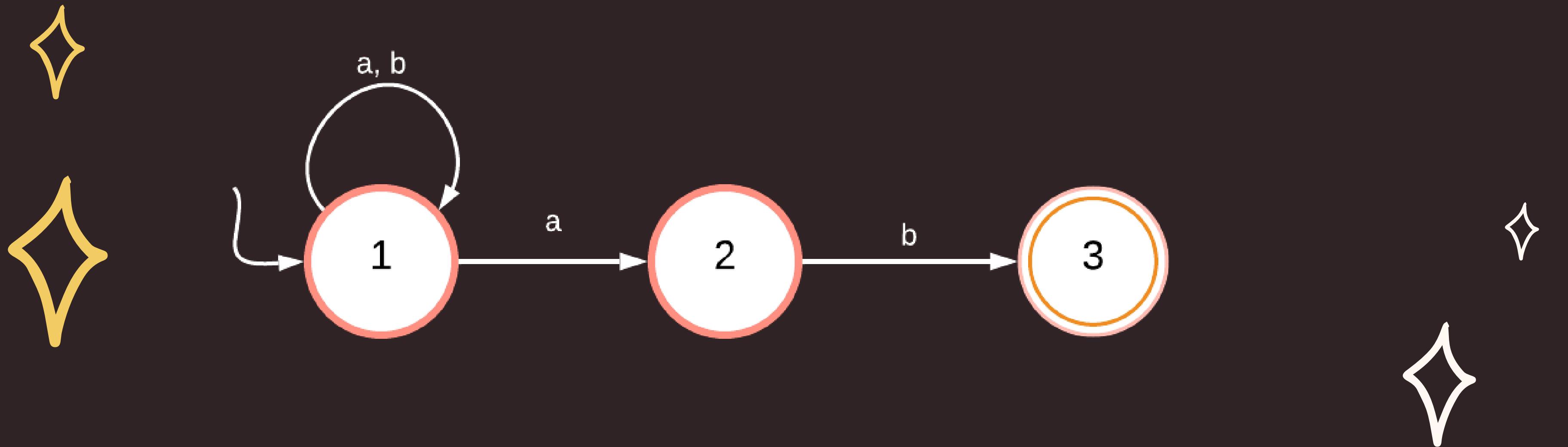


Se rechaza!



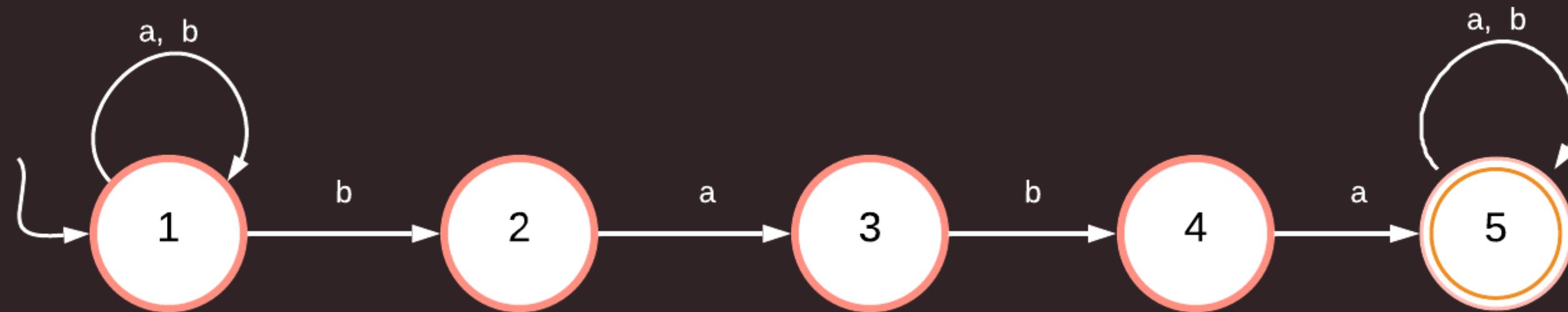
Ejemplo 5

- Sea L el lenguaje sobre $\Sigma = \{a, b\}$ de hileras tales que terminan en "bb"
- Diseñe un NFA que reconozca a L .



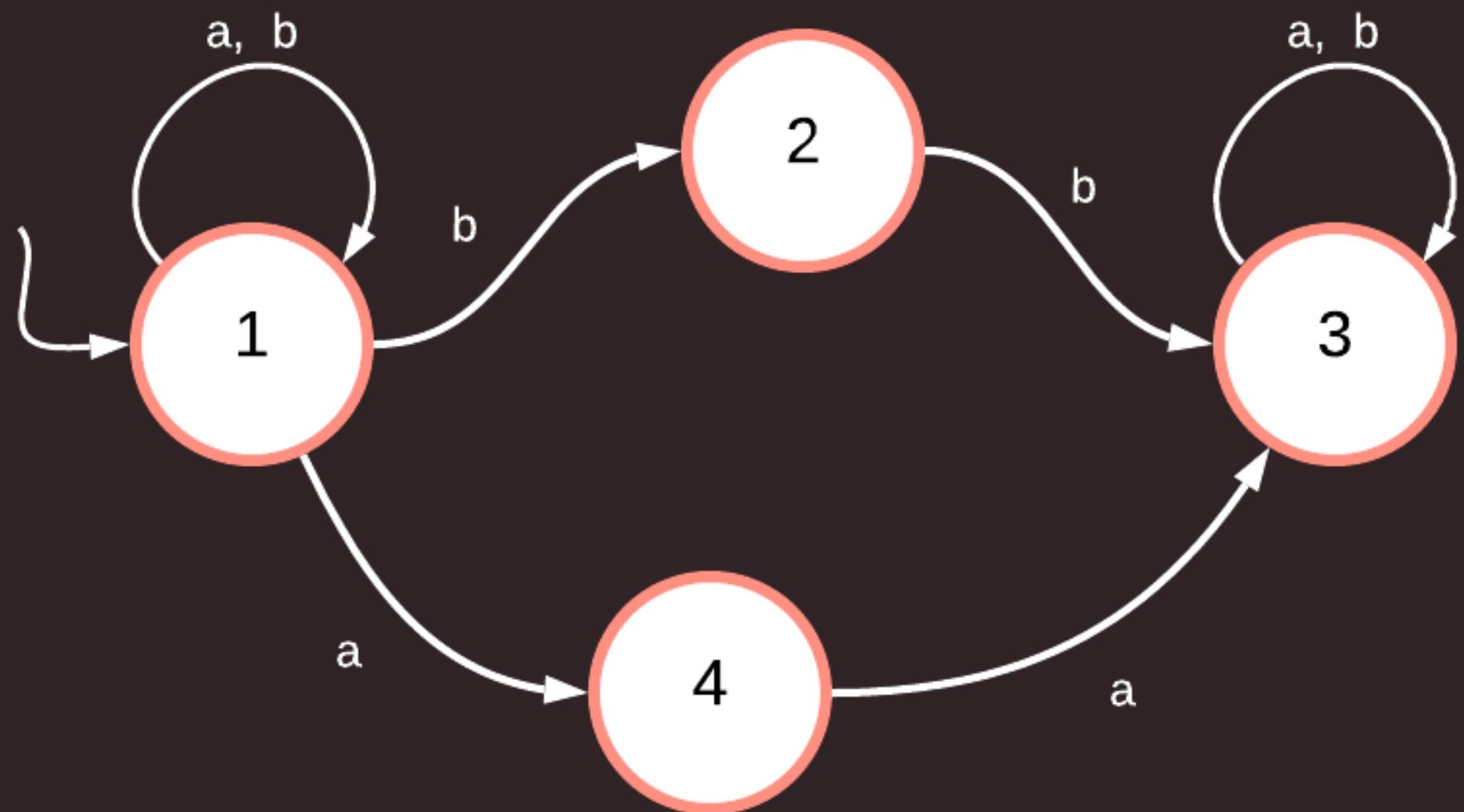
Ejemplo 6

- Sea L el lenguaje sobre $\Sigma = \{a, b\}$ de hileras tales que terminan en "bab"
- Diseñe un NFA que reconozca a L .



Ejemplo 7

- Sea L el lenguaje sobre $\Sigma = \{a, b\}$ de hileras que contengan las subhileras "aa" o "bb"
- Diseñe un NFA que reconozca a L .



Definición Formal

- Un Autómata No Determinístico de Estados Finitos (NFA) es un quinteto $M = (Q, \Sigma, \delta, q_0, F)$, donde:

- Q es un conjunto finito de estados
- Σ es un alfabeto
- $\delta: Q \times \Sigma \rightarrow P(Q)$ es la función de transición
- $q_0 \in Q$ es el estado inicial
- $F \subseteq$ conjunto de estados de aceptación

• En este caso a diferencia con los DFA tenemos $P(Q)$ (Partes de Q)



Teorema

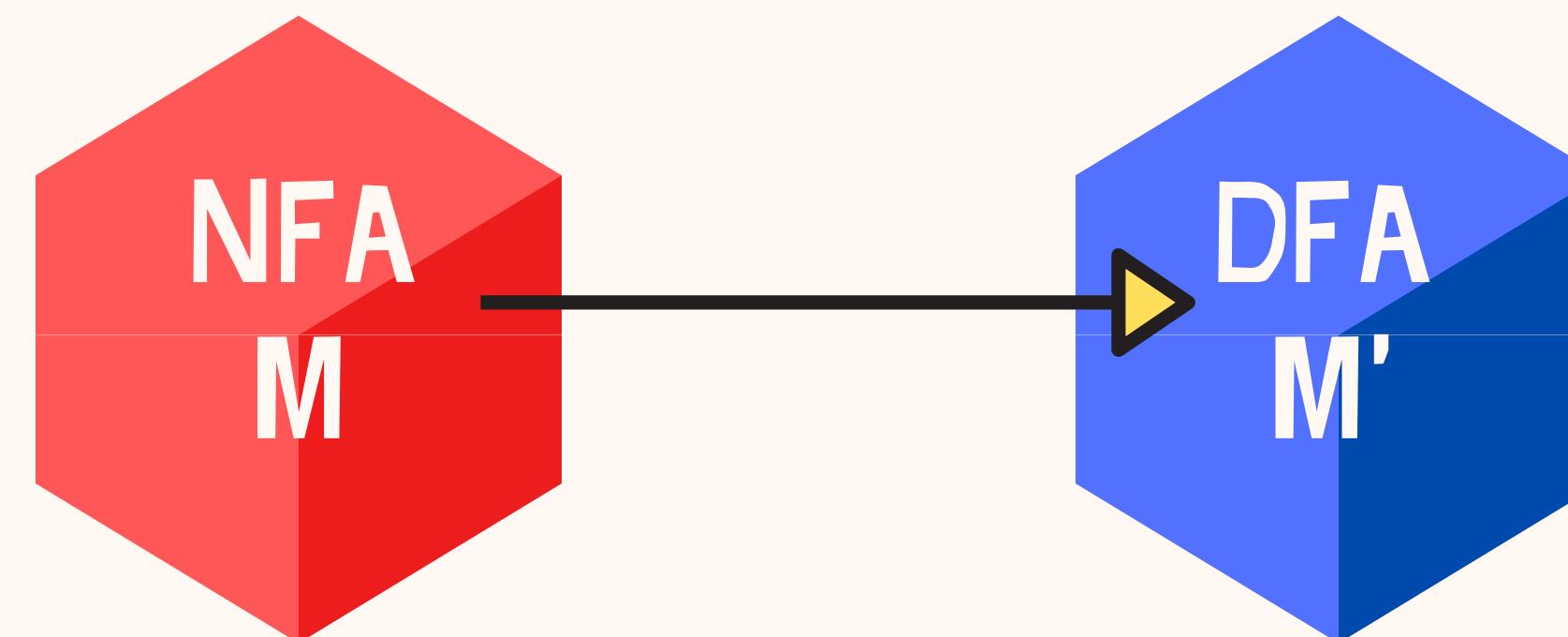
IMPORTANT!



Sea $M = (Q, \Sigma, \delta, q_0, F)$ un NFA.



Siempre existe un DFA
 $M' = (P(Q), \Sigma, \delta, \{q_0\}, F')$ que
reconoce exactamente el mismo
lenguaje que M

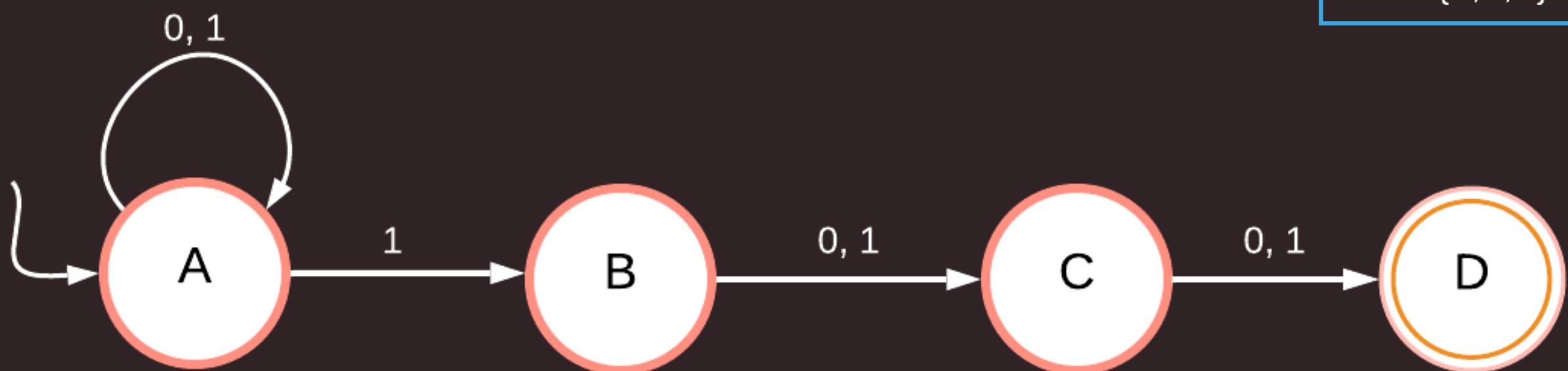


Cálculo de Nueva Función de Transición

1. El estado inicial de M' es $\{q_0\}$, o sea un conjunto que contiene al estado inicial de M , esta es la primera fila de la tabla asociada a δ'
2. En la tabla asociada a la función de transición δ' coloque en la columna de todo símbolo de Σ el conjunto de todos los lugares a los que se puede llegar desde q_0 con dicho símbolo (el conjunto vacío es una opción válida)
3. Tome el primero de estos subconjuntos y póngalo como una nueva fila de la tabla de δ'
4. Llene las columnas con el conunto de todos los destinos a los que se llega con el símbolo de la columna desde todos los miembros de este subconjunto.
5. Repita mientras quedan subconjuntos pendientes.

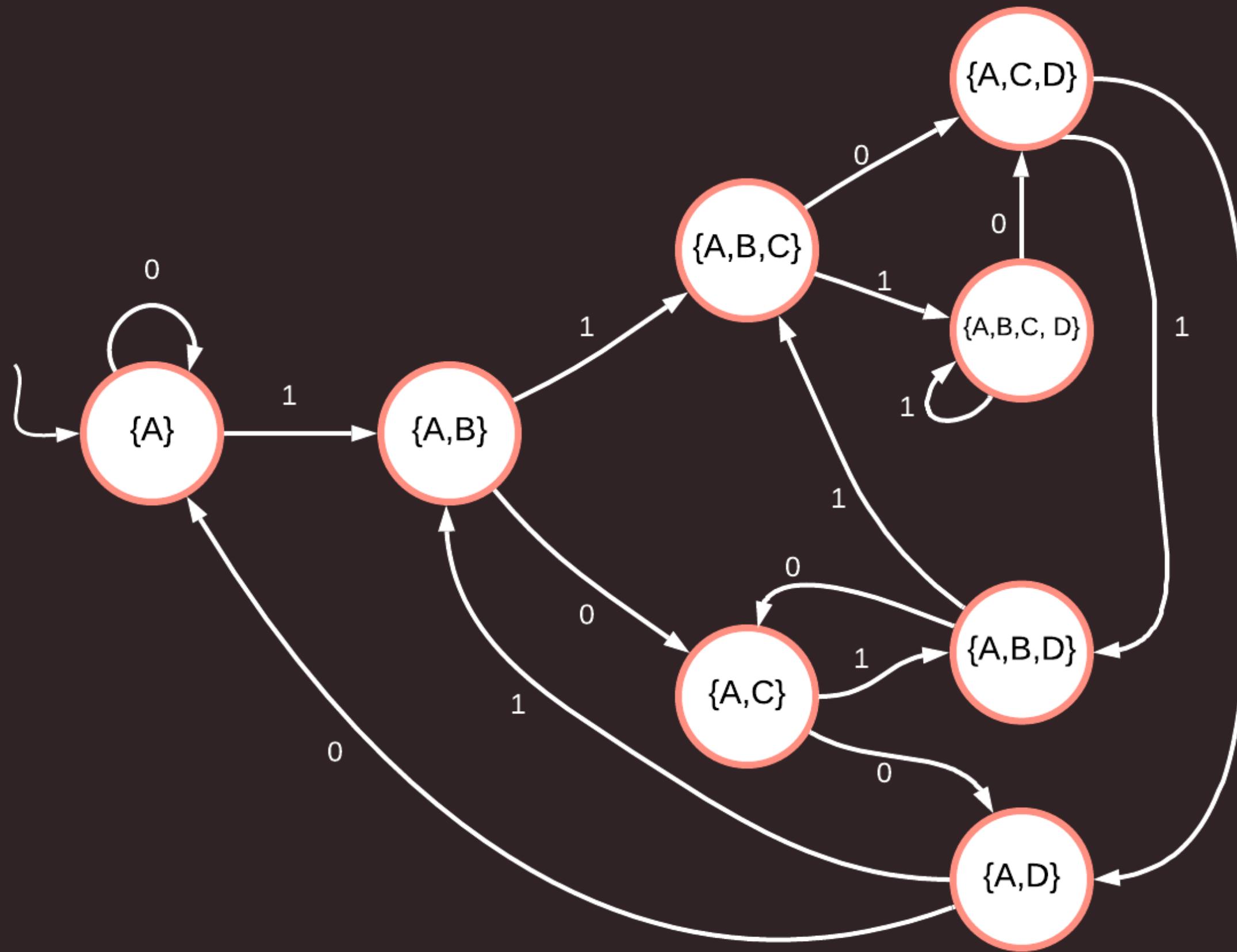


Ejemplo



	0	1
{A}	{A}	{A,B}
{A,B}	{A,C}	{A,B,C}
{A,B,C}	{A,C,D}	{A,B,C,D}
{A,C}	{A,D}	{A,B,D}
{A,C,D}	{A,D}	{A,B,D}
{A,B,C,D}	{A,C,D}	{A,B,C,D}
{A,D}	{A}	{A,B}
{A,B,D}	{A,C}	{A,B,C}

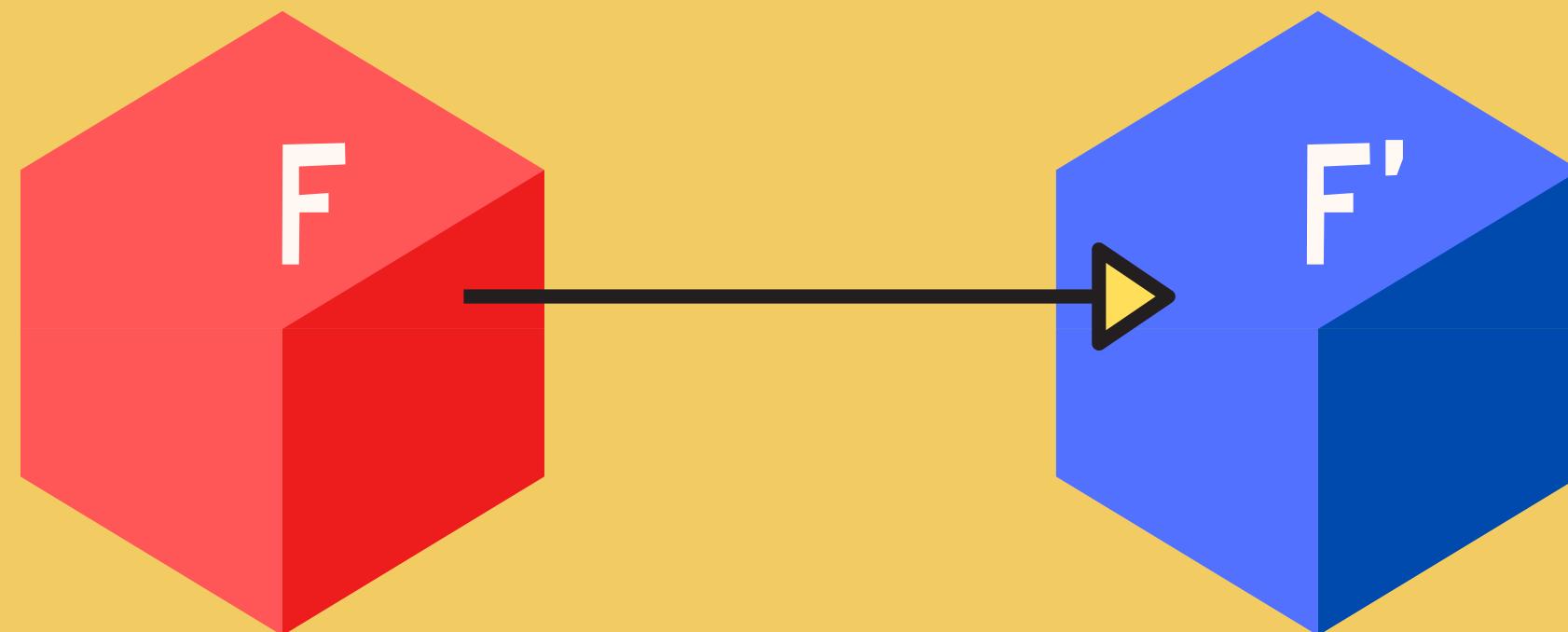
Genera el siguiente DFA





ESTADOS DE ACEPTACIÓN

- Cualquier estado en M' que contenga al menos un elemento que sea miembro de F es miembro de F'



Con Estados de Aceptación

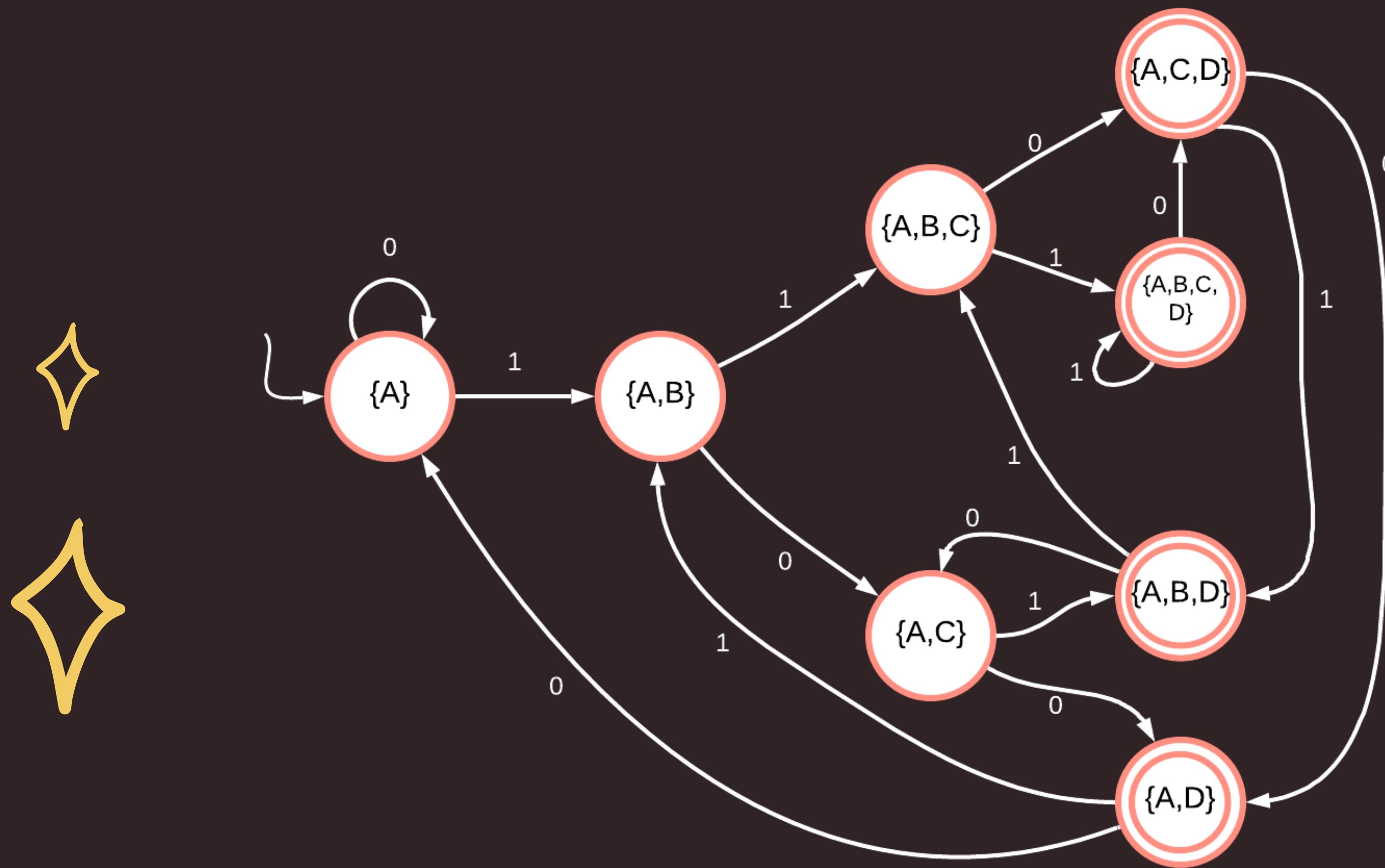
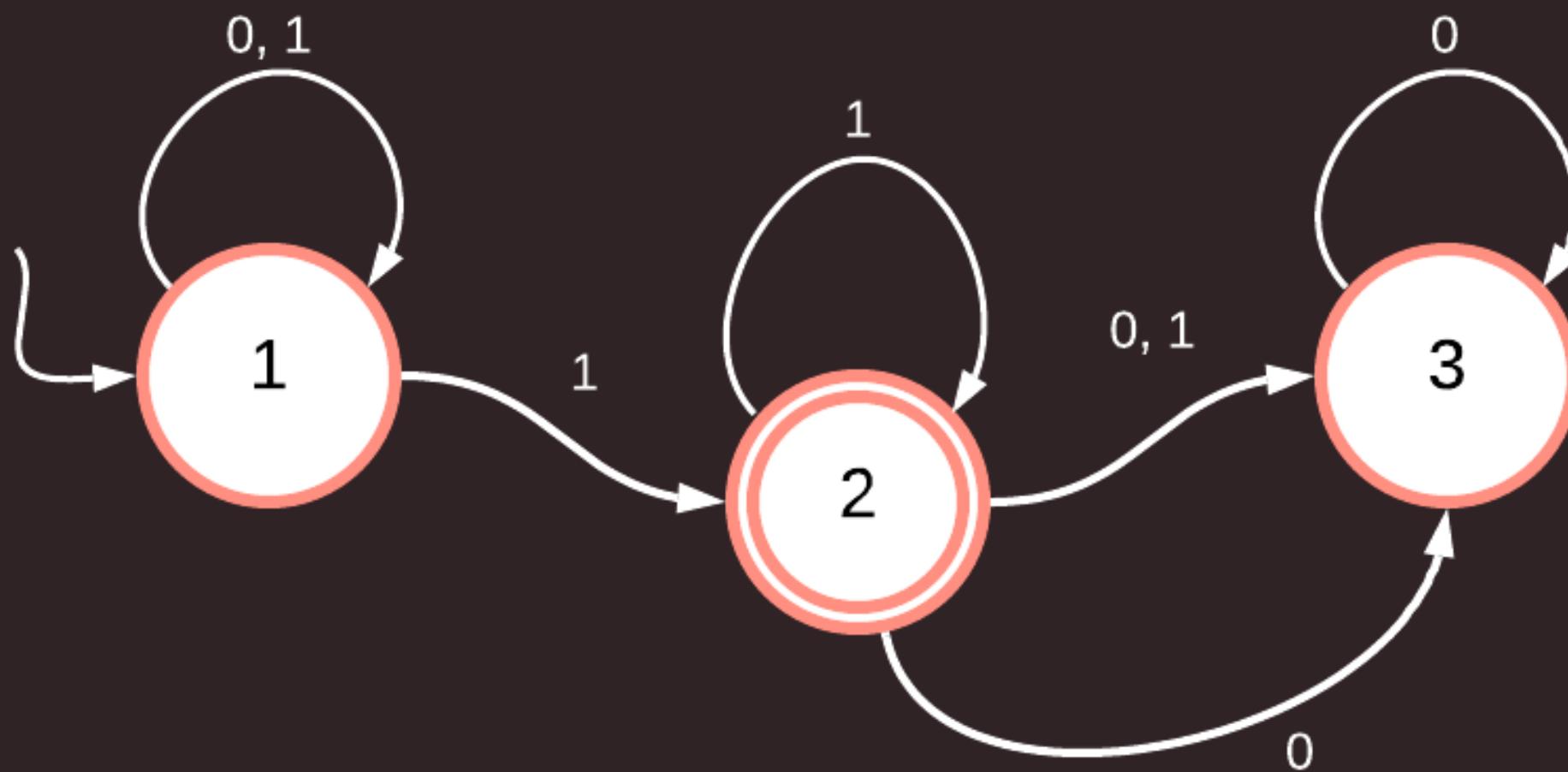


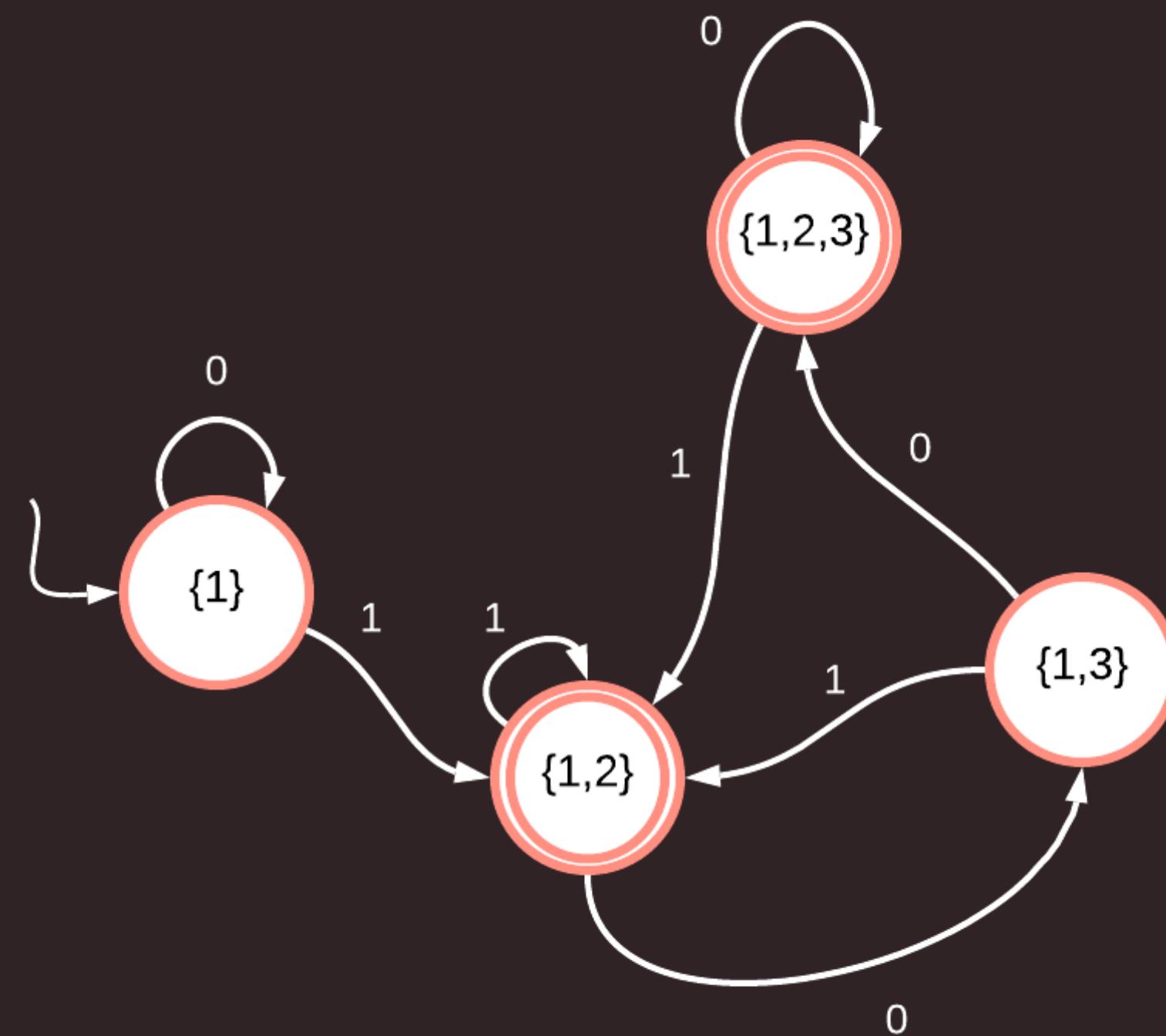
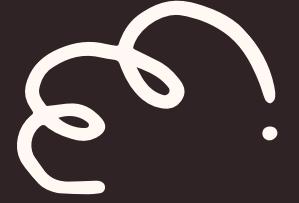
Tabla resultante

Ejemplo 2

	0	1
{1}	{1}	{1,2}
{1,2}	{1,3}	{1,2}
{1,3}	{1,2,3}	{1,2}
{1,2,3}	{1,3,2}	{1,2}



Con la tabla creamos el siguiente DFA



Ejemplo 3

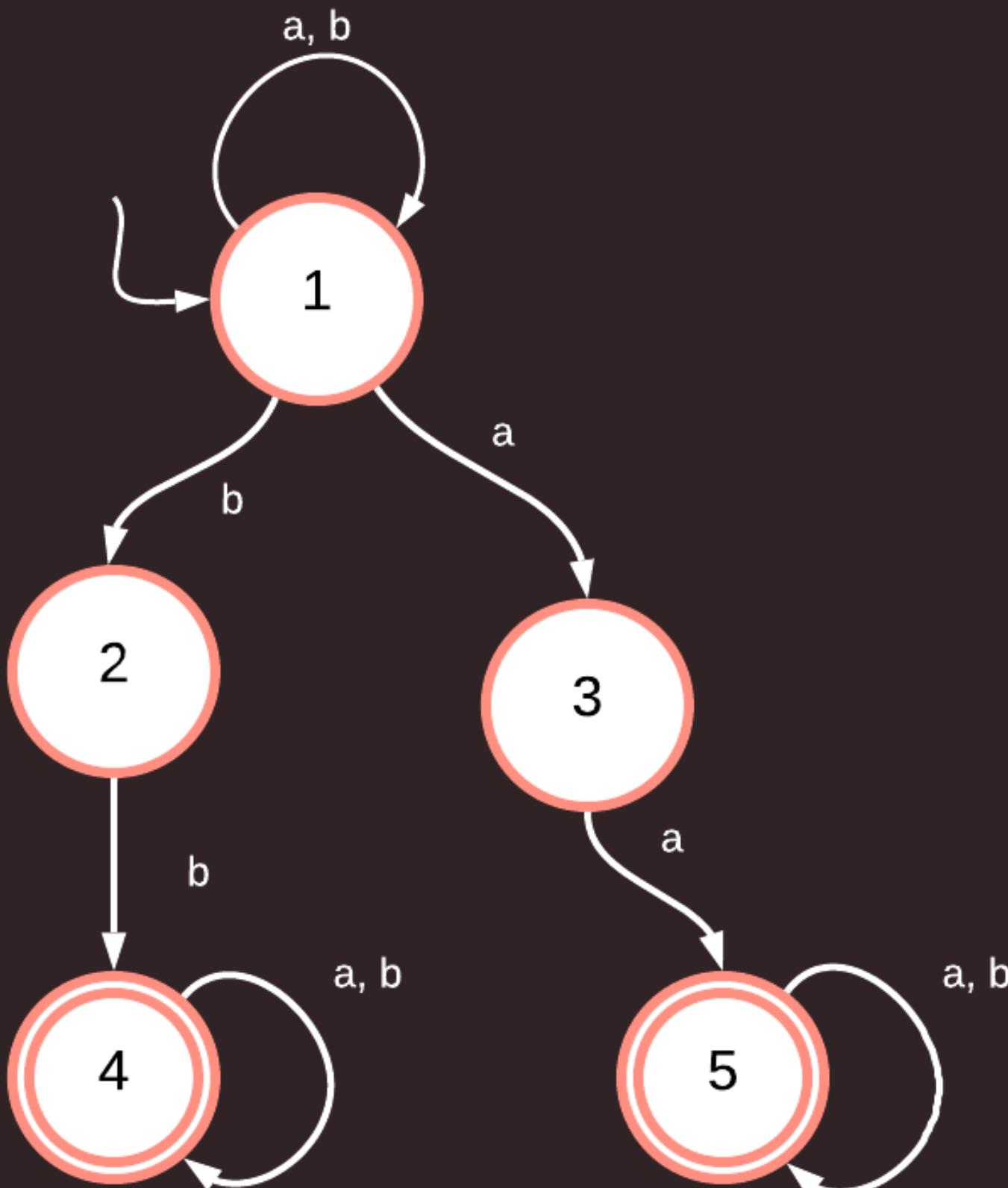


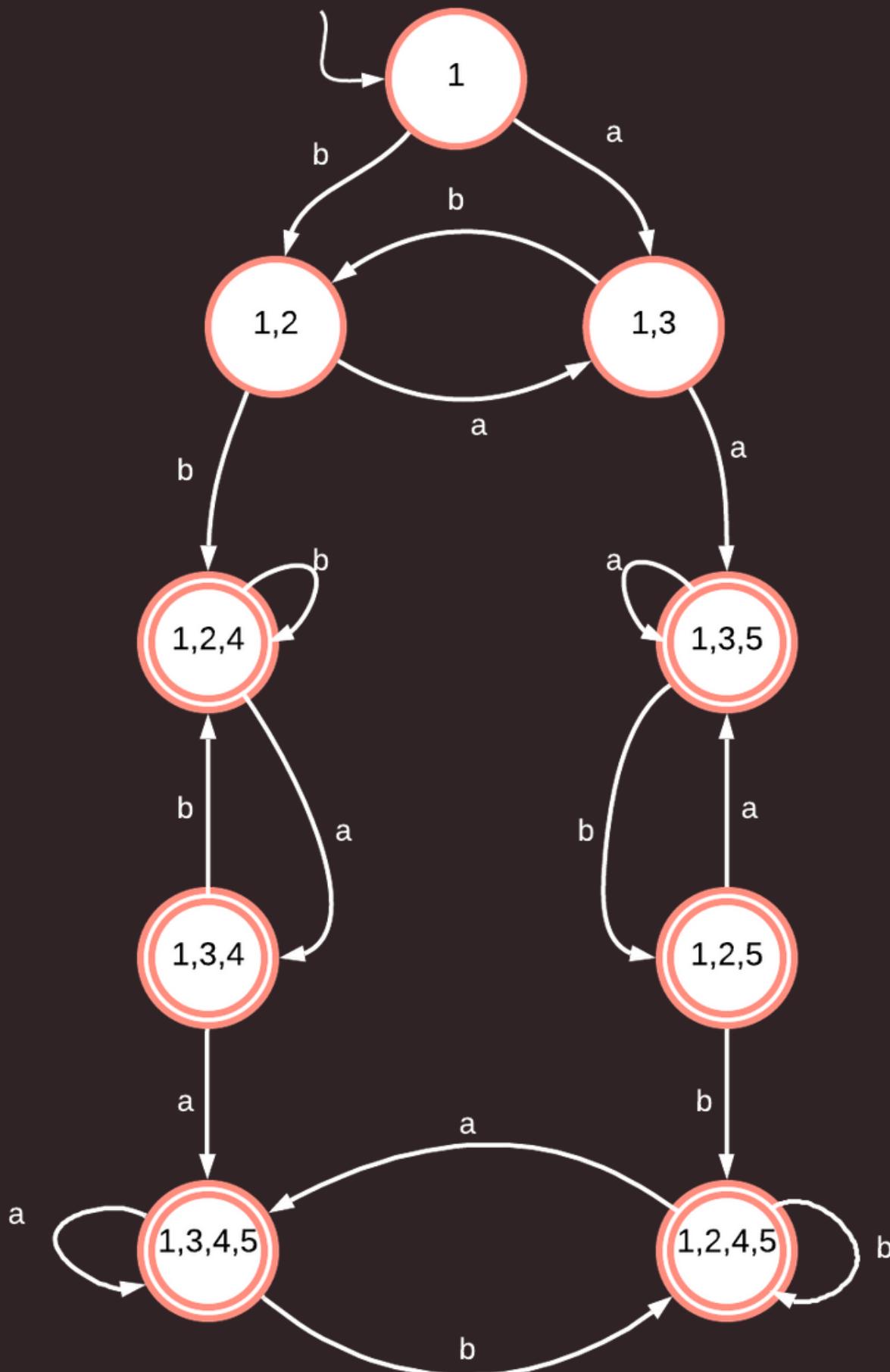
Tabla resultante

	0	1
{1}	{1,3}	{1,2}
{1,3}	{1,3,5}	{1,2}
{1,2}	{1,3}	{1,2,4}
{1,3,5}	{1,3,5}	{1,2,5}
{1,2,4}	{1,3,4}	{1,2,4}
{1,2,5}	{1,3,5}	{1,2,4,5}
{1,3,4}	{1,3,4,5}	{1,2,4}
{1,2,4,5}	{1,3,4,5}	{1,2,4,5}
{1,3,4,5}	{1,3,4,5}	{1,2,4,5}

- Tener **MUCHO** cuidado de no repetir algún conjunto , recordar que los conjuntos **no tienen orden**



Con la tabla creamos el siguiente DFA



	0	1
{1}	{1,3}	{1,2}
{1,3}	{1,3,5}	{1,2}
{1,2}	{1,3}	{1,2,4}
{1,3,5}	{1,3,5}	{1,2,5}
{1,2,4}	{1,3,4}	{1,2,4}
{1,2,5}	{1,3,5}	{1,2,4,5}
{1,3,4}	{1,3,4,5}	{1,2,4}
{1,2,4,5}	{1,3,4,5}	{1,2,4,5}
{1,3,4,5}	{1,3,4,5}	{1,2,4,5}

La clase estuvo muy callada por la
falta de nuestro compañero Dennis :(

