LAS FRONTERAS DE LA CIENCIA
Máquinas y Lenguajes

Real Sociedad Económica de Amigos del País de Tenerife

nautis et incolis

28 de Enero 2021

Casiano Rodríguez León

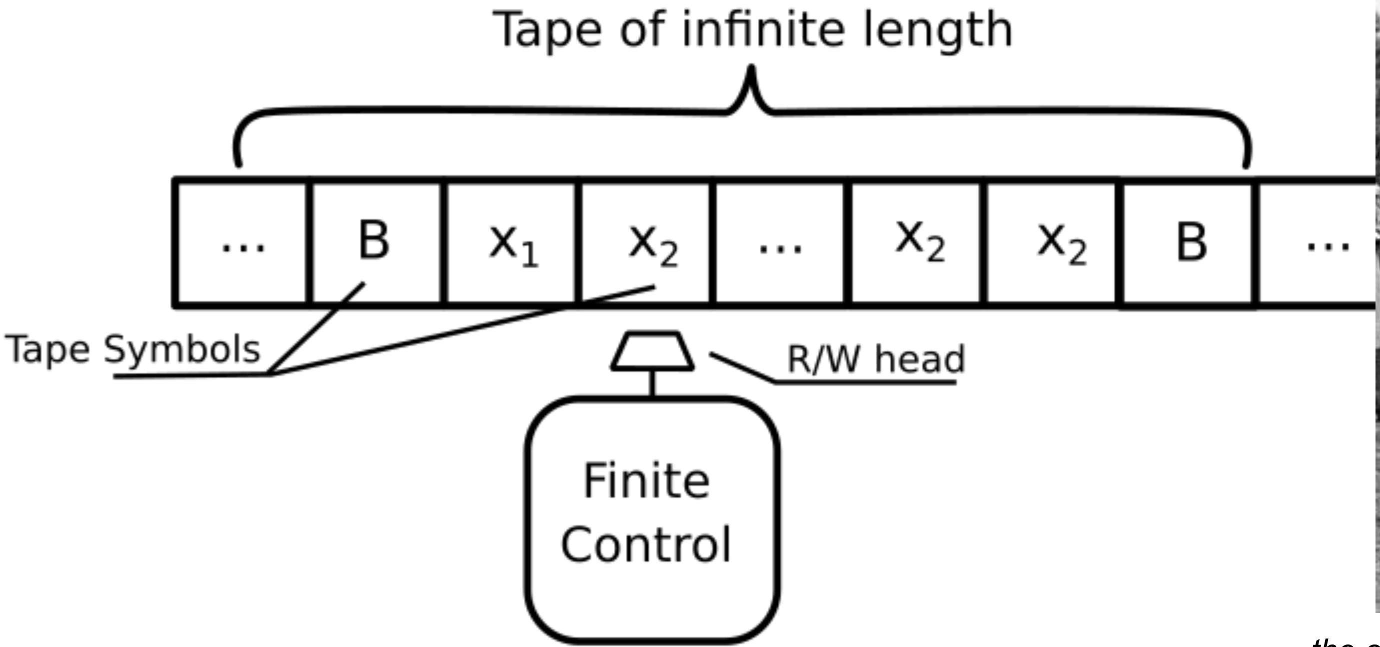
Departamento de Ingeniería Informática y de Sistemas
Universidad de La Laguna

Aula Cultural de Pensamiento Computacional
Universidad de La Laguna

Grupo de Investigación Algoritmos y Lenguajes Paralelos
Universidad de La Laguna

Machines

Máquinas de Turing




Alan Turing (1947?)

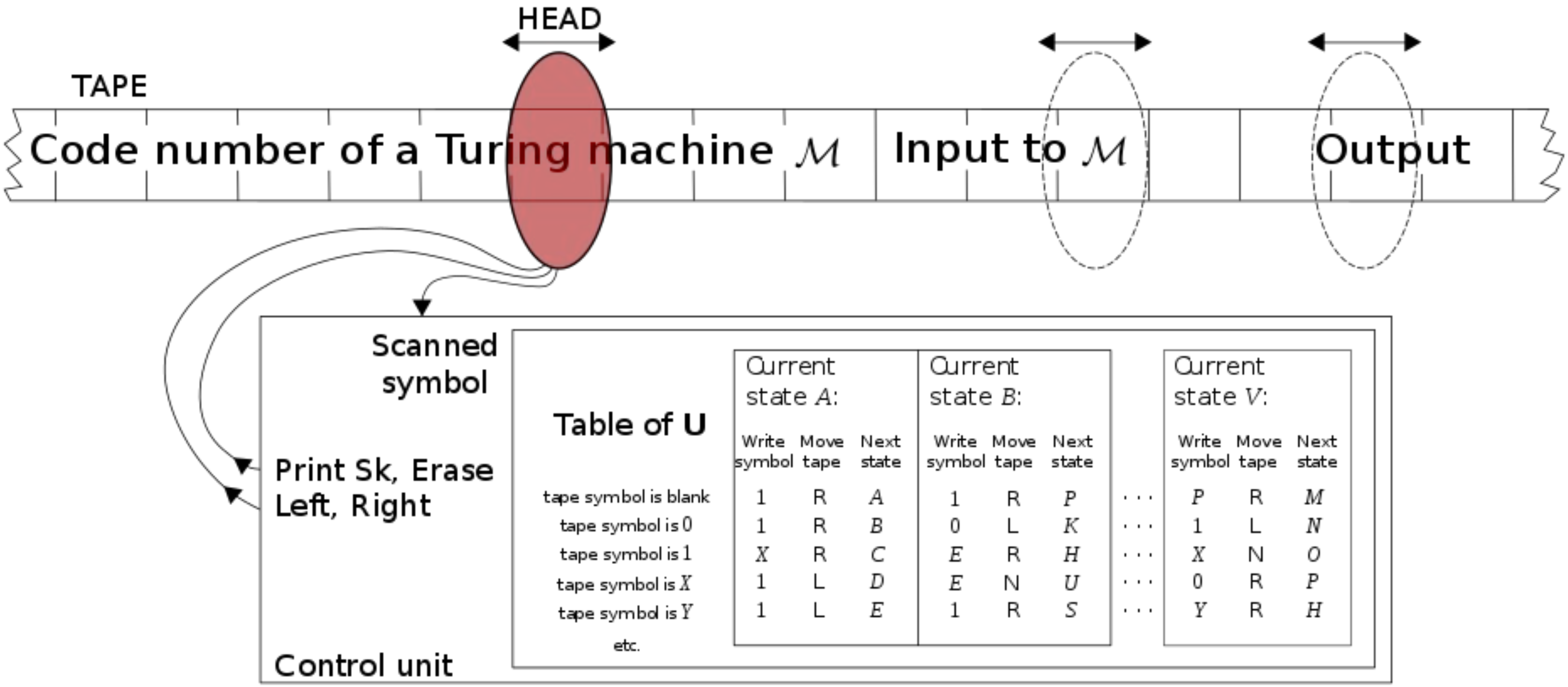
… the only way I can get it out of my mind is by running hard; it's the only way I can get some release

https://turingmachine.io/

https://turingmachine.io/

```
# Add two unary numbers
input: '11111 1111' # 5 + 4
blank: ' '
start state: searchForWhite
table:
  #  search for the first white
  searchForWhite:
    1       :  R
    ' '      :  {write: 1, R: searchForTheEnd}
  # then carry the 1
  searchForTheEnd:
    1       : R
    ' '      : {L: removeLastOne}
  removeLastOne:
    1 : {write: ' ', L: done}
  done:
```

UNIVERSAL TURING MACHINE: http://morphett.info/turing/



Alan Turing ***On Computable Numbers, with an Application to the Entscheidungsproblem***. 1937

Turing reduces this problem to the **Halting Problem**.

Halting Problem: *Write a program that from a description of an arbitrary computer program will always stop telling us whether the program will finish running, or continue to run forever.*

```
let halts = loadSolution('halting-problem-solution');
function forEver() { while (true) print(‘Hello’); }
if (halts(forEver)) print('It halts!')
else print(‘Loops for ever!’);
```

```
function paradox() {
  if (halts(paradox)) { forEver(); } else stop;
}
```

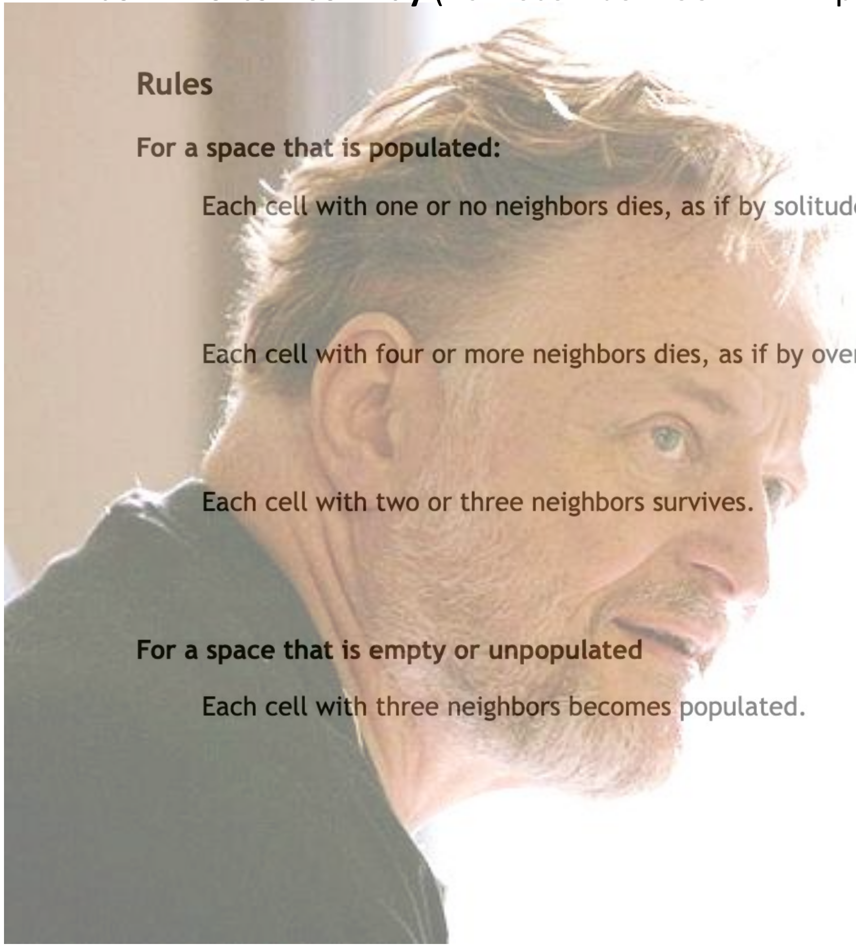
```
halts(paradox);
```

Corollary: There are problems for which you cannot find a program to solve them

Cellular Automata

Cellular automata were invented in the 1940s by **John von Neumann** and **Stanislaw Ulam** at Los Alamos National Laboratory. They consist of a two-dimensional array of cells that “evolve” step-by-step according to the state of neighbouring cells and certain rules that depend on the simulation.

John Horton Conway (26 December 1937 – 11 April 2020) **The Game of Life.**



Rules

For a **space that** is **populated:**

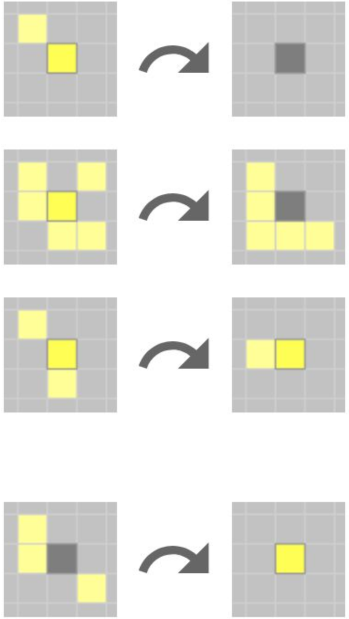
Each cell with one or no neighbors dies, as if by solitude.

Each cell with four or more neighbors dies, as if by overpopulation.

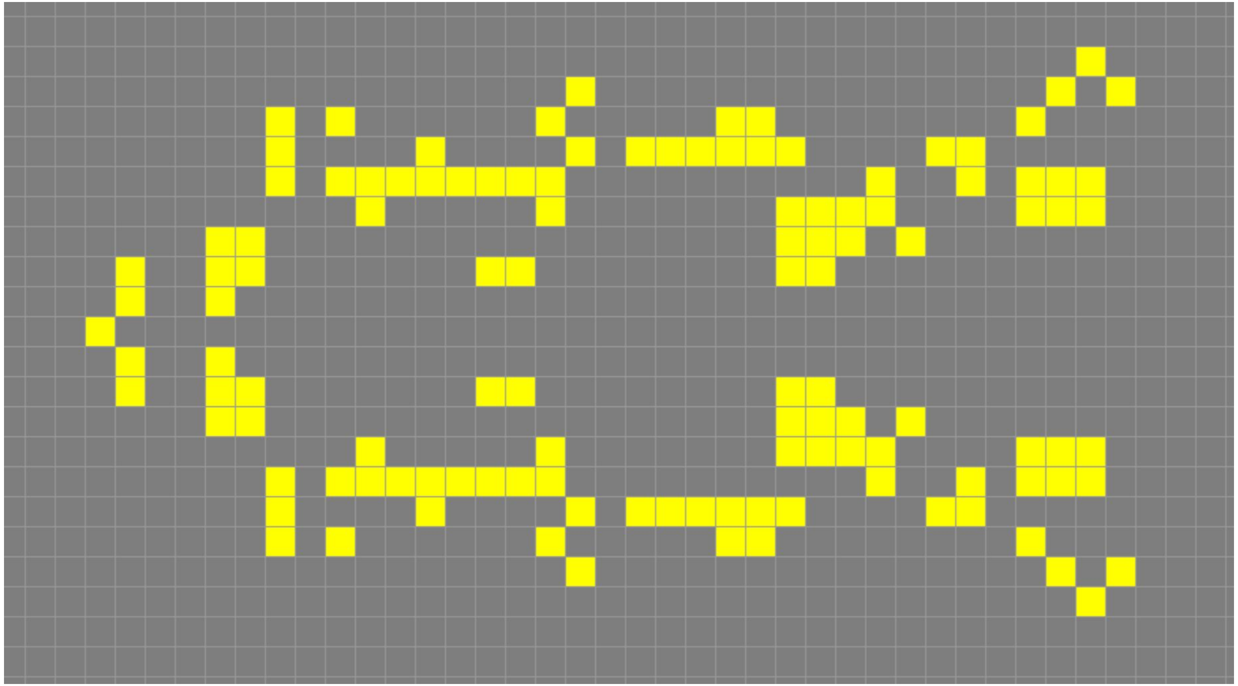
Each cell with two or three neighbors survives.

For a space that is empty or unpopulated

Each cell with three neighbors becomes populated.



119P4H1V0: A spaceship discovered by Dean Hickerson in December 1989



On February 10, 2010 Paul Rendell found a Codification of a Universal Turing Machine inside Conway's game.

A Turing Machine In Conway's Game Life.

Paul Rendell

I have constructed a Turing Machine in Conway's Game Life (figure 1). In this paper I describes the machine's parts, how it works and the principle choices made during the construction.

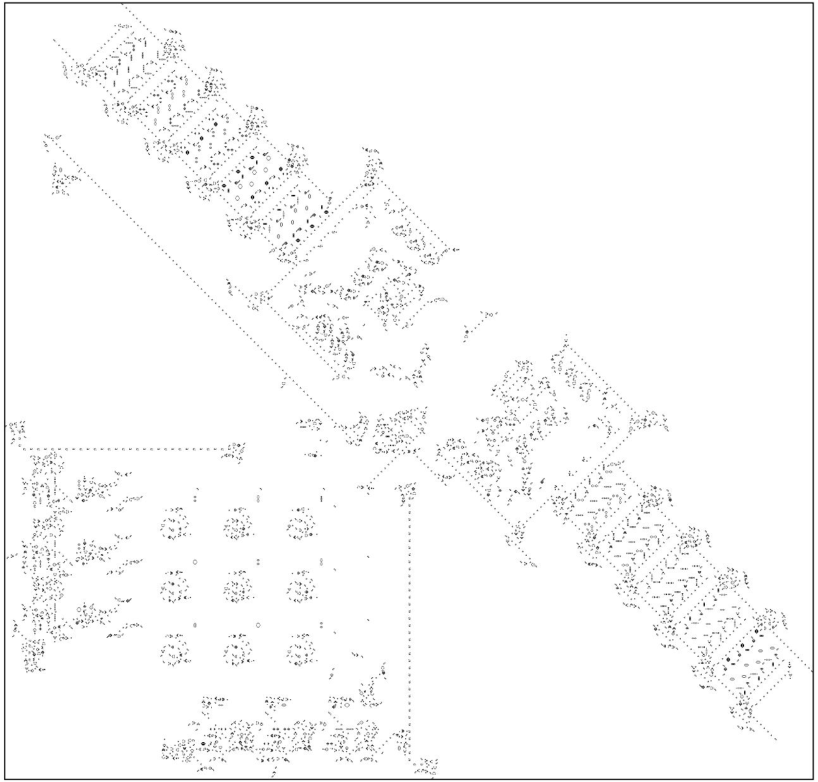
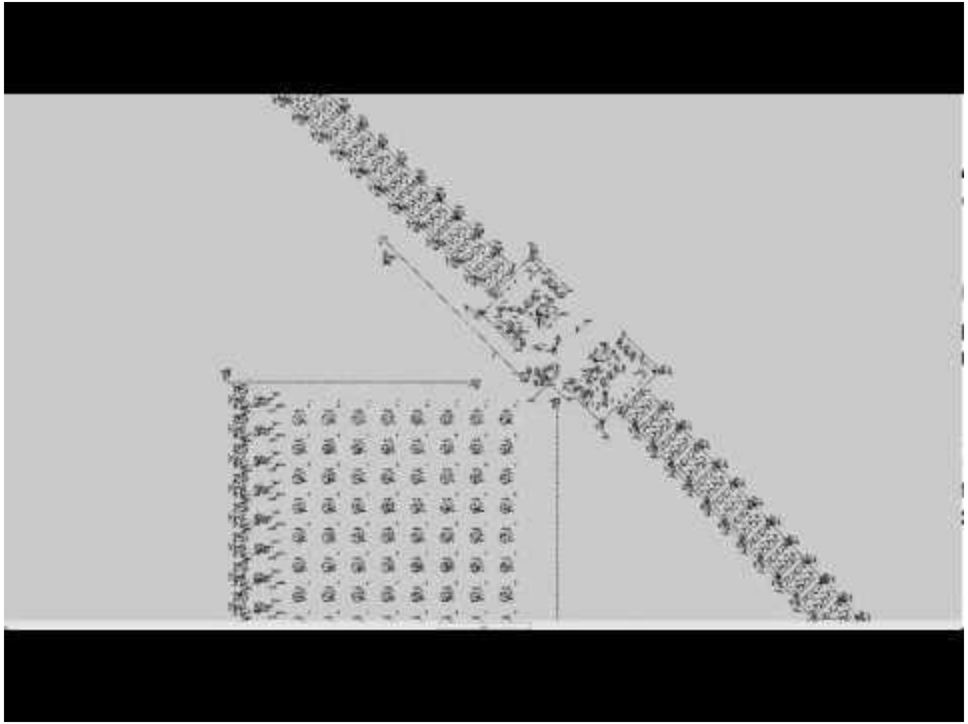


Figure 1The Complete Turing Machine



Church–Turing thesis

Two computer models *P* and *Q* are called **equivalent** if *P* can simulate *Q* and *Q* can simulate *P*

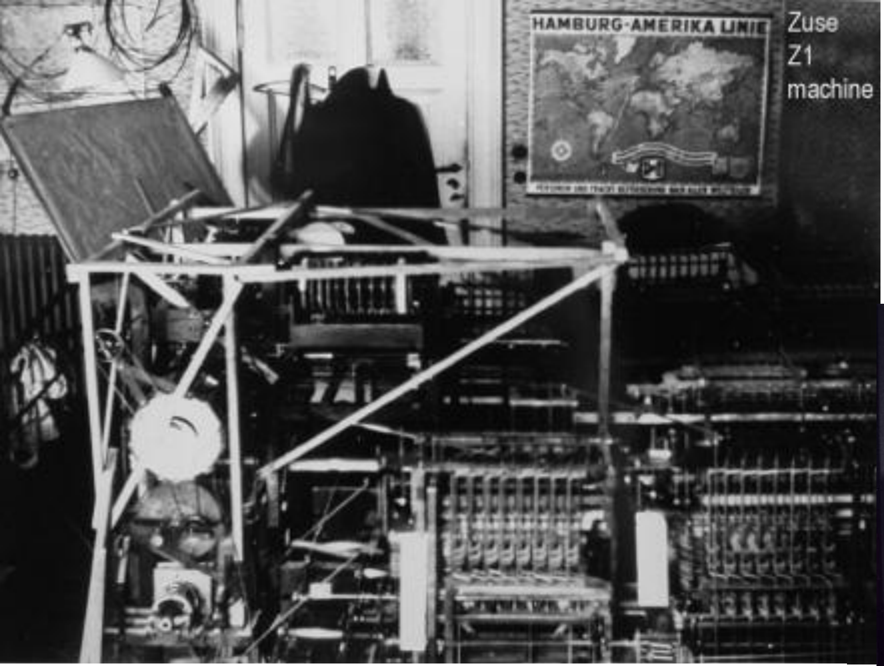
- In 1933, Kurt Gödel created a formal definition of a class called **general recursive functions**
- In 1936, Alonzo Church created a method for defining functions called the **λ-calculus**
- In 1936, Turing created the **Turing machines**

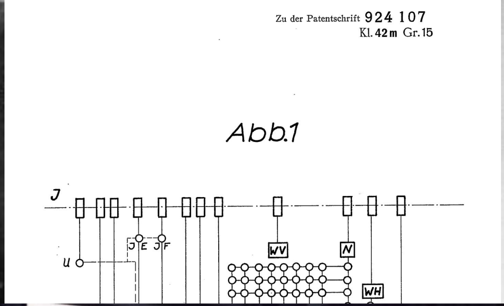
These three formally defined classes of computable functions coincide:
Theorem: A function is λ-computable if and only if it is Turing computable, and if and only if it is general recursive.

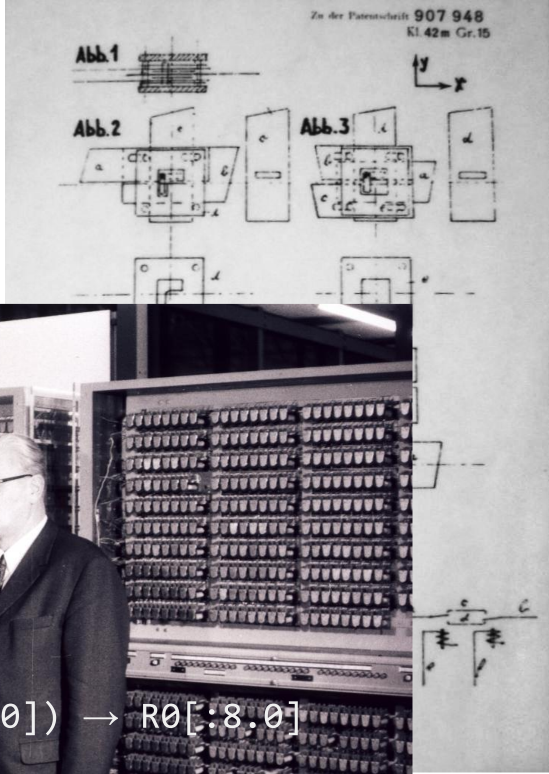
Any **effectively (mechanically) calculable** function can be computed by any of the above three formally-defined models (**informal notion**)

*Any **real-world** calculation can be done using the lambda calculus or Turing Machines* (Wolfrang)

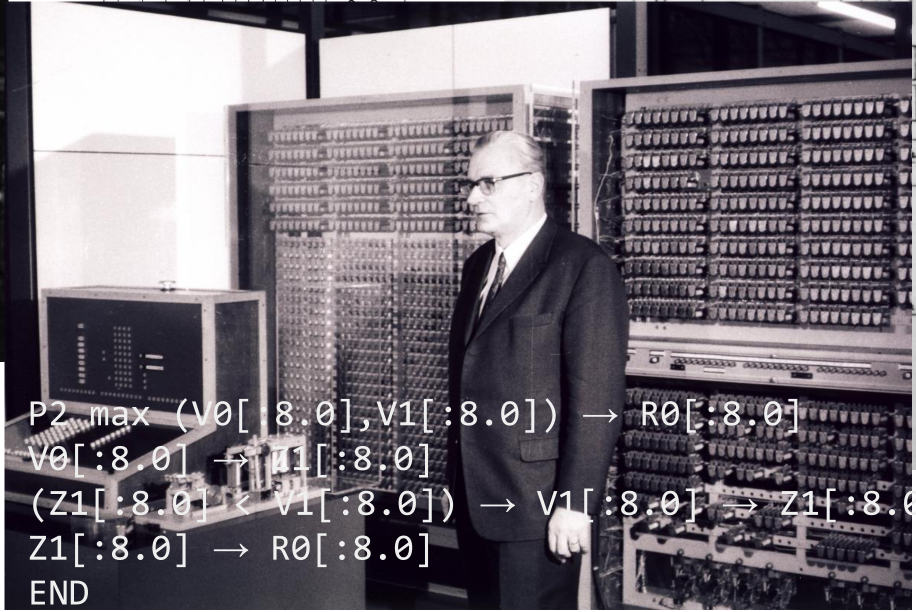
Essentially, then, the Church-Turing thesis says that no human computer, or machine that mimics a human computer, can out-compute the universal Turing machine. Copeland. The Stanford Encyclopedia of Philosophy







En 1935 Zuse construye la Z1. Leía las instrucciones desde una cinta perforada de 35 mm. No era una máquina Turing completa

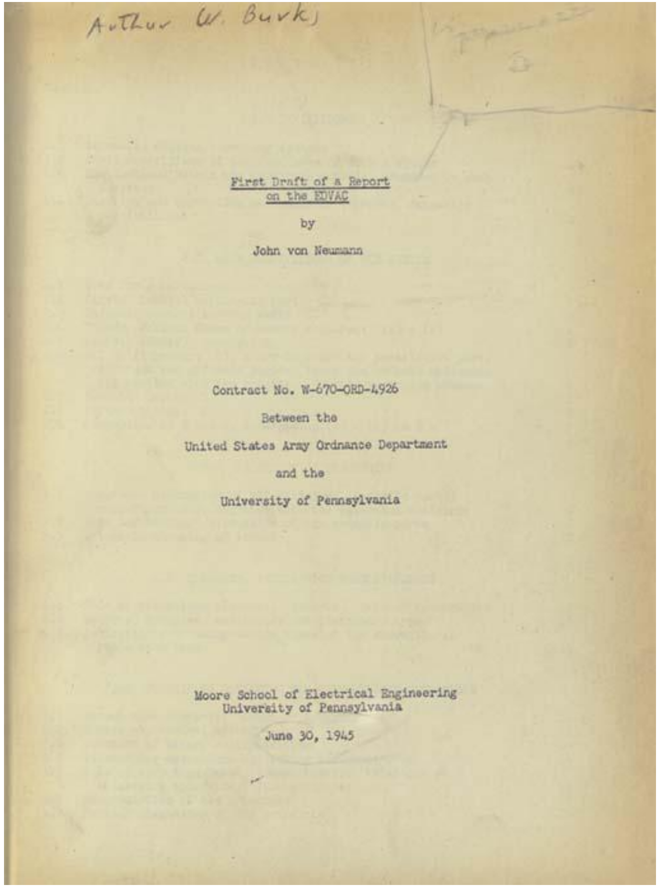


```
P2 max (V0[:8.0],V1[:8.0]) → R0[:8.0]
V0[:8.0] → Z1[:8.0]
(Z1[:8.0] < V1[:8.0]) → V1[:8.0] → Z1[:8.0]
Z1[:8.0] → R0[:8.0]
END
```

The Henschel Hs 129B ground attack aircraft

La Z3 (1941) era un computador binario de punto flotante de 22 bits con memoria y unidad de cálculo basada en relés telefónicos. No almacenaba el programa en memoria. A pesar de la ausencia de saltos condicionales, el Z3 era un ordenador Turing-completo

John von Neumann "First Draft on a Report on the EDVAC" June 1945



Arthur W. Burks

First Draft of a Report on the EDVAC

by

John von Neumann

Contract No. W-670-ORD-4926

Between the

United States Army Ordnance Department

and the

University of Pennsylvania

Moore School of Electrical Engineering
University of Pennsylvania

June 30, 1945

Alan Turing "Proposed Electronic Calculator" 1945. Page 3/48 (ACE)

It is intended that the setting up of the machine for new problems shall be virtually only a matter of paper work. Besides the paper work nothing will have to be done except to prepare a pack of Hollerith cards in accordance with this paper work, and to pass them through a card reader connected with the machine. There will positively be no internal alterations to be made even if we wish suddenly to switch from calculating the energy levels of the neon atom to the enumeration of groups of order 720. It may appear somewhat puzzling that this can be done. How can one expect a machine to do all this multitudinous variety of things? The answer is that we should consider the machine as doing something quite simple, namely carrying out orders given to it in a standard form which it is able to understand.

The actual calculation done by the machine will be carried out in the binary scale. Material will however be put in and taken out in decimal form.

In order to obtain high speeds of calculation the calculator will be entirely electronic. A unit operation (typified by adding one and one) will take 1 microsecond. It is not thought wise to design for higher speeds than this as yet.

The present report gives a fairly complete account of the proposed calculator. It is recommended however that it be read in conjunction with J. von Neumann's 'Report on the EDVAC'.

2. Composition of the Calculator.

We list here the main components of the calculator as at present conceived:-

(1) Erasible memory units of fairly large capacity, to be known as dynamic storage (DS). Probably consisting of between 50 and 500 mercury tanks with a capacity of about 1000 digits each.

(2) Quick reference temporary storage units (TS) probably numbering about 50 and each with a capacity of say 32 binary digits.

(3) Input organ (IO) to transfer instructions and other material into the calculator from the outside world. It will have a mechanical part consisting of a Hollerith card reading unit, and an electronic part which will be internal to the calculator.

(4) Output organ (OO), to transfer results out of the calculator. It will have an external part consisting of a Hollerith card reproducer and an internal electronic part.

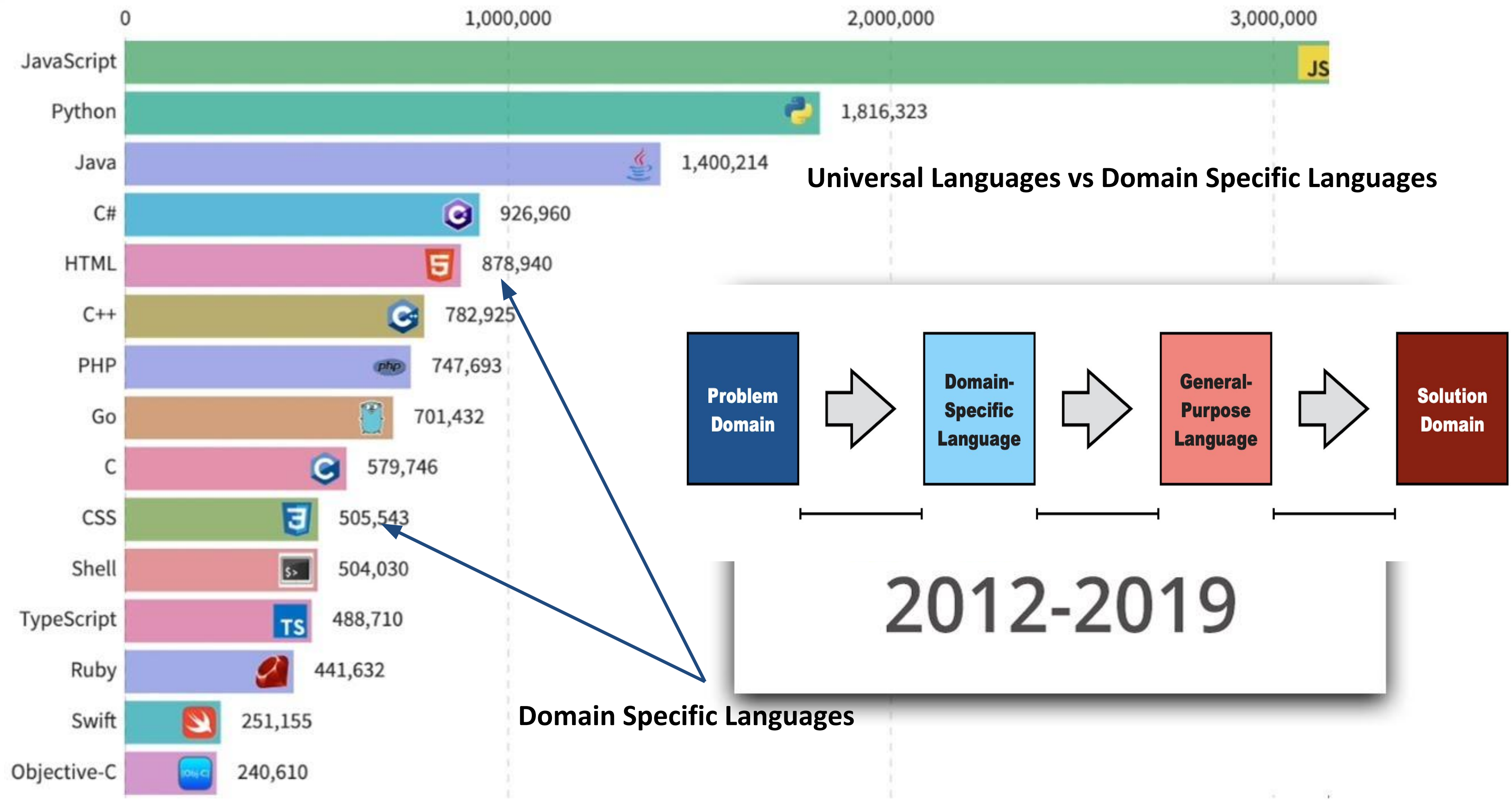
(5) The logical control (LC). This is the very heart of the machine. Its purpose is to interpret the instructions and give them effect. To a large extent it merely passes the instructions on to CA. There is no very distinct line between LC and CA.

(6) The central arithmetic part (CA). If we like to consider LC as the analogue of a computer then CA must be considered a desk calculating machine. It carries out the four fundamental arithmetical processes (with possible exception of division, see p. 27), and various others of the nature of copying, substituting, and the like. To a large extent these processes can be reduced to one another by various roundabout means; judgment is therefore required in choosing an appropriate set of fundamental processes.

(7)/

Languages

Universal Languages vs Domain Specific Languages

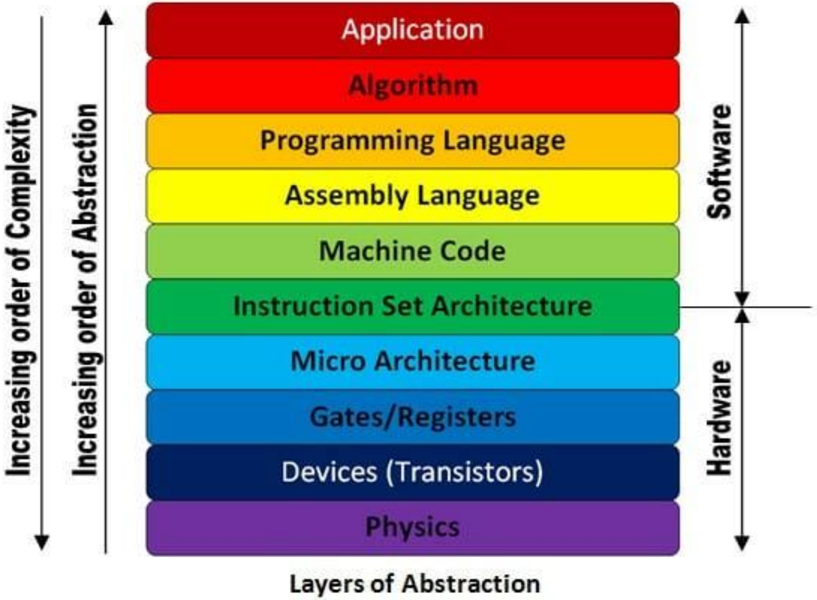


Domain Specific Languages

Universal Programming Language = Creation of a new **universal** (In Turing's sense) machine on top of an existing one (software)

Operating System = Creation of a new **universal** (In Turing's sense) machine on top of an existing one (hardware)

It's (Software Machines) all the way down up to the Hardware



Computer programming is the process of telling a computer to do certain things by giving it instructions. These instructions are called programs. A person who writes instructions is a computer programmer. The instructions come in different languages; they are called programming languages.

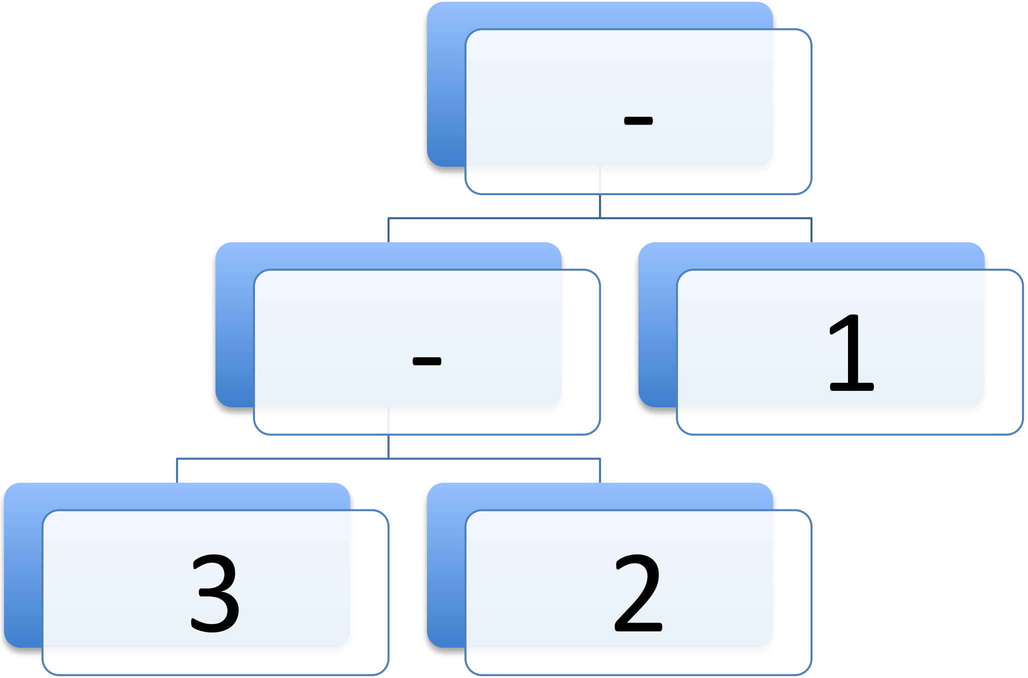
https://simple.wikipedia.org/

3 - 2 - 1

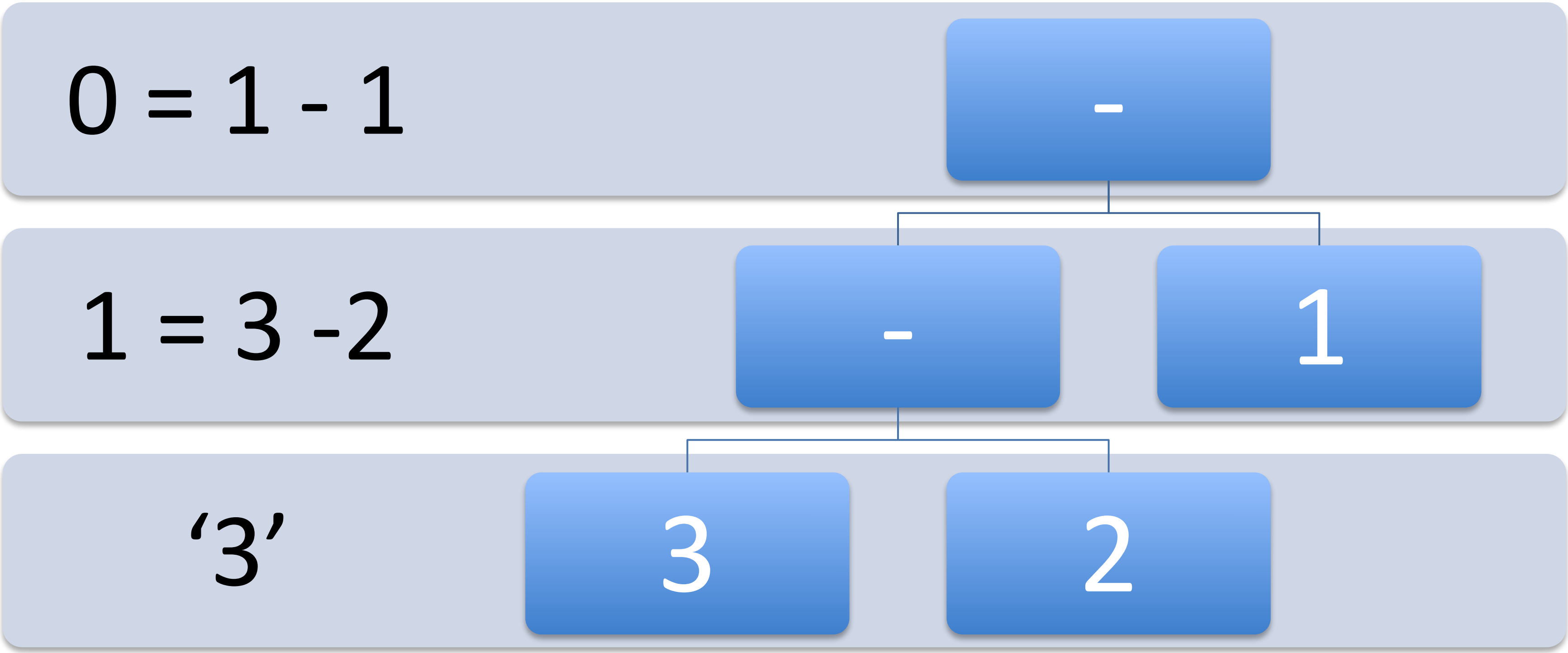
Árbol Sintáctico Abstracto

3-2-1

(3-2)-1



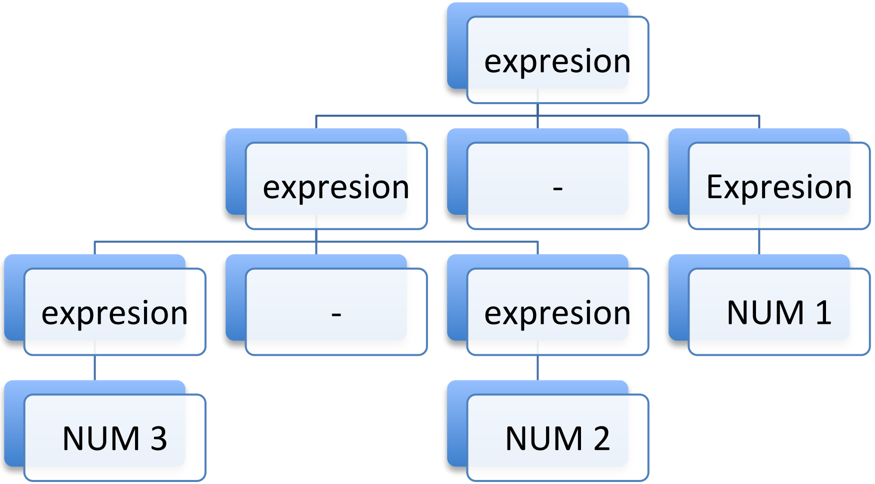
Semántica 3 - 2 - 1



Gramática Independiente del Contexto

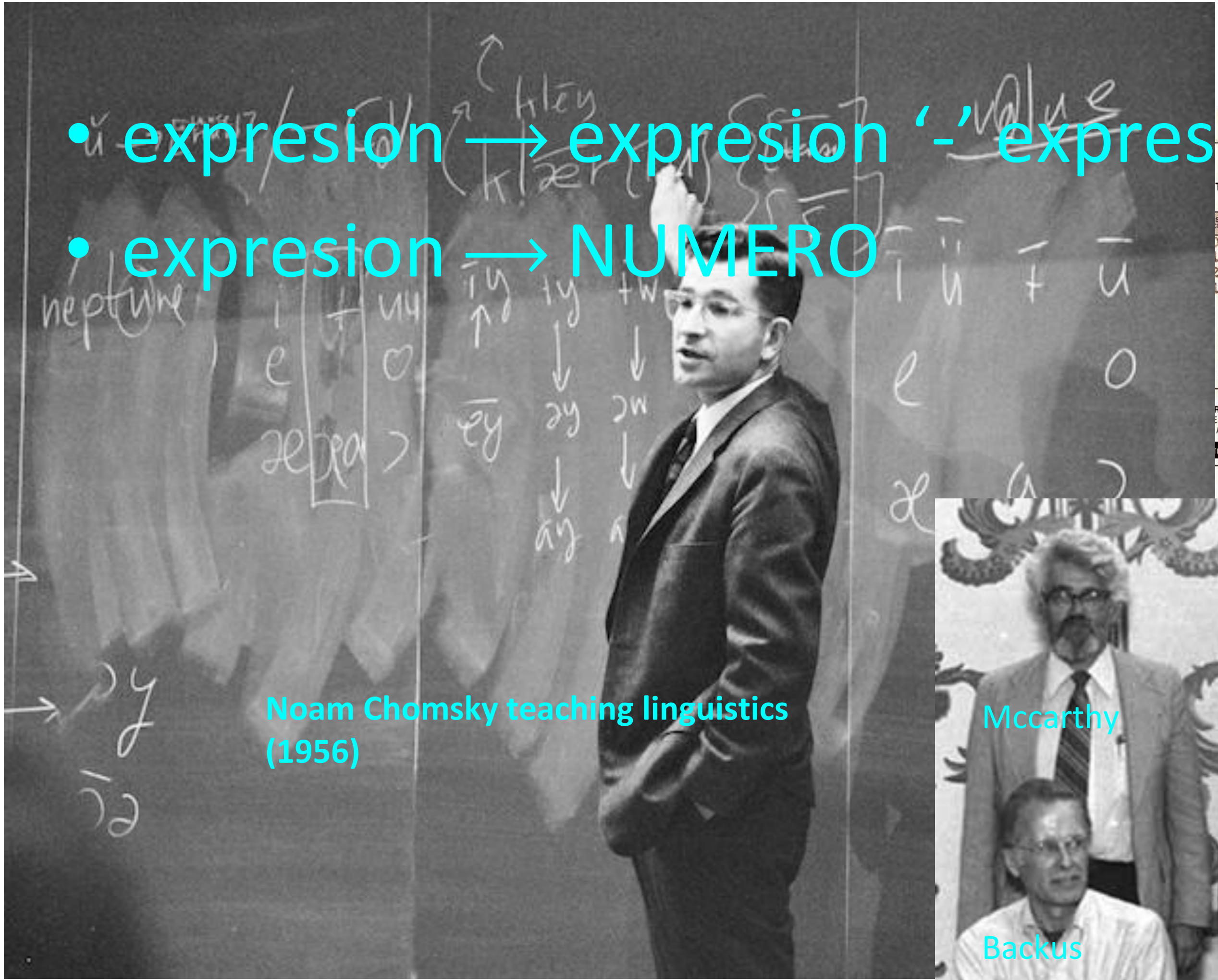
- expresion ⟶ expresion '-' expresion
- expresion ⟶ NUMERO

3-2-1

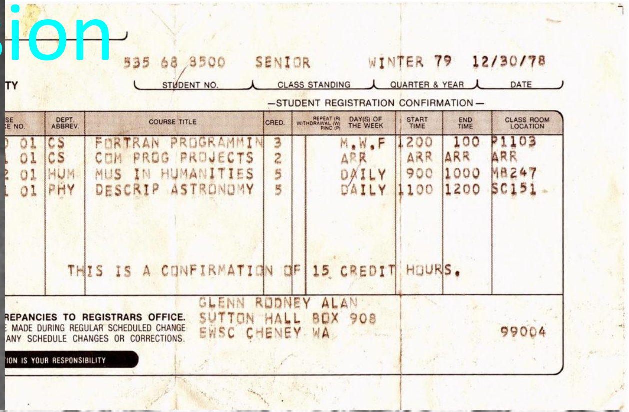


Context Free Grammars

- expresion ⟶ expresion '-' expresion
- expresion ⟶ NUMERO



Noam Chomsky teaching linguistics (1956)



535 68 8500 SENIOR WINTER 79 12/30/78

STUDENT NO. CLASS STANDING QUARTER & YEAR DATE

—STUDENT REGISTRATION CONFIRMATION—

SE NO.	DEPT ABBREV.	COURSE TITLE	CRED.	REPEAT (R) WITHDRAWAL (W) PINC (P)	DAY(S) OF THE WEEK	START TIME	END TIME	CLASS ROOM LOCATION
01	CS	FORTRAN PROGRAMMIN	3		M,W,F	1200	100	P1103
01	CS	COM PROG PROJECTS	2		ARR	ARR	ARR	ARR
01	HUM	MUS IN HUMANITIES	5		DAILY	900	1000	MB247
01	PHY	DESCRIP ASTRONOMY	5		DAILY	1100	1200	SC151

THIS IS A CONFIRMATION OF 15 CREDIT HOURS.

REPANCIES TO REGISTRARS OFFICE.
E MADE DURING REGULAR SCHEDULED CHANGE
ANY SCHEDULE CHANGES OR CORRECTIONS.

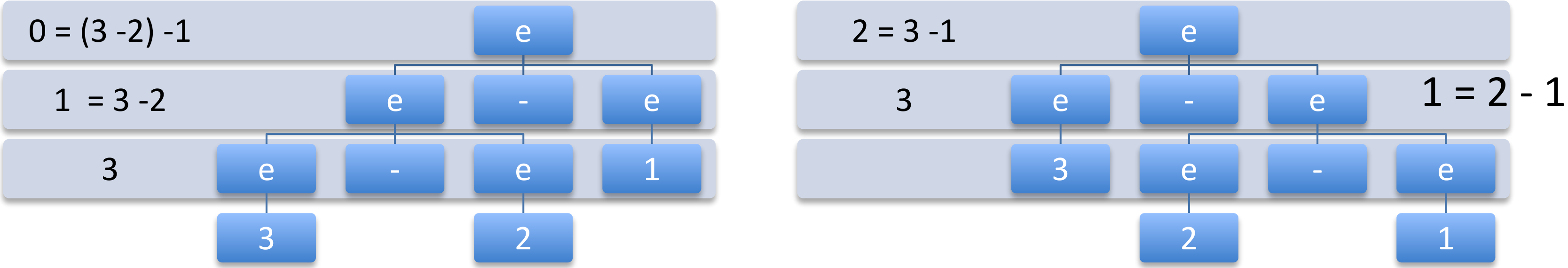
GLENN RODNEY ALAN
SUTTON HALL BOX 908
EWSC CHENEY WA 99004

ION IS YOUR RESPONSIBILITY

The Algol 60 people

Gramática Ambigua

- expresion ⟶ expresion '-' expresion
- expresion ⟶ NUMERO

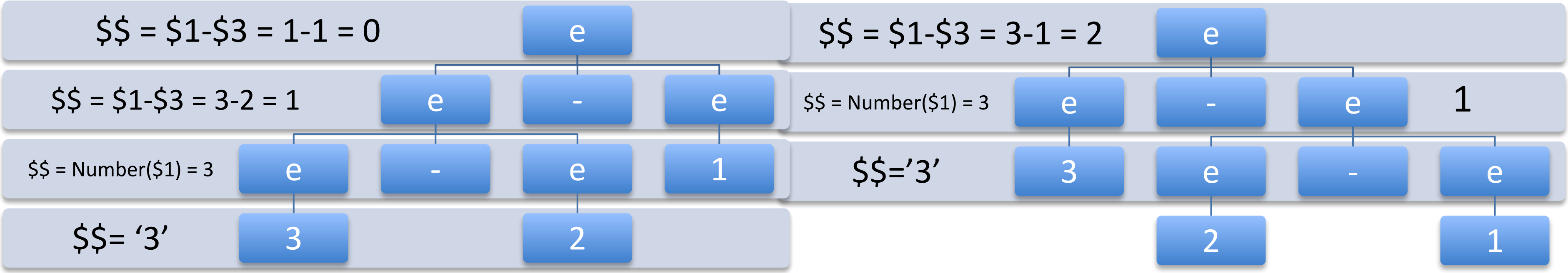


Esquema de Traducción (yacc)

e ⟶ e '-' e { $$ = $1 - $3; }

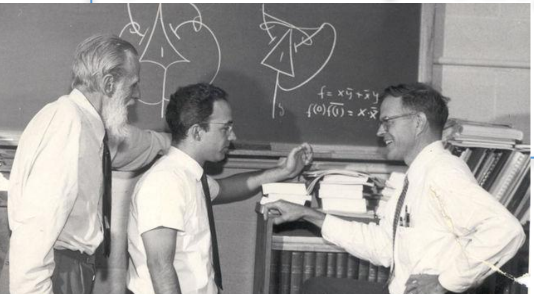
e ⟶ NUM { $$ = Number($1); }

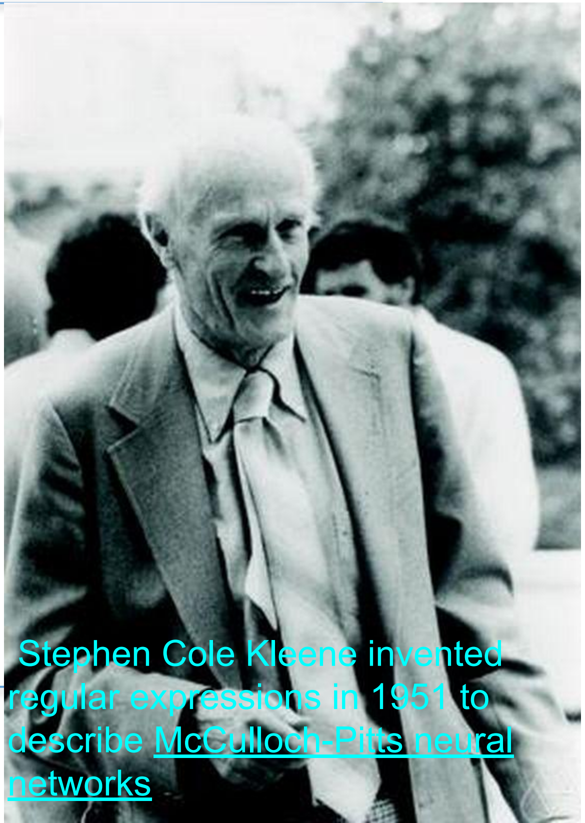
3–2-1



Análisis Léxico y Expresiones Regulares

```
[0-9]+ /* is a Natural Number */

"-"    /* is a '-' */

.      /*Any character but \n*/
```





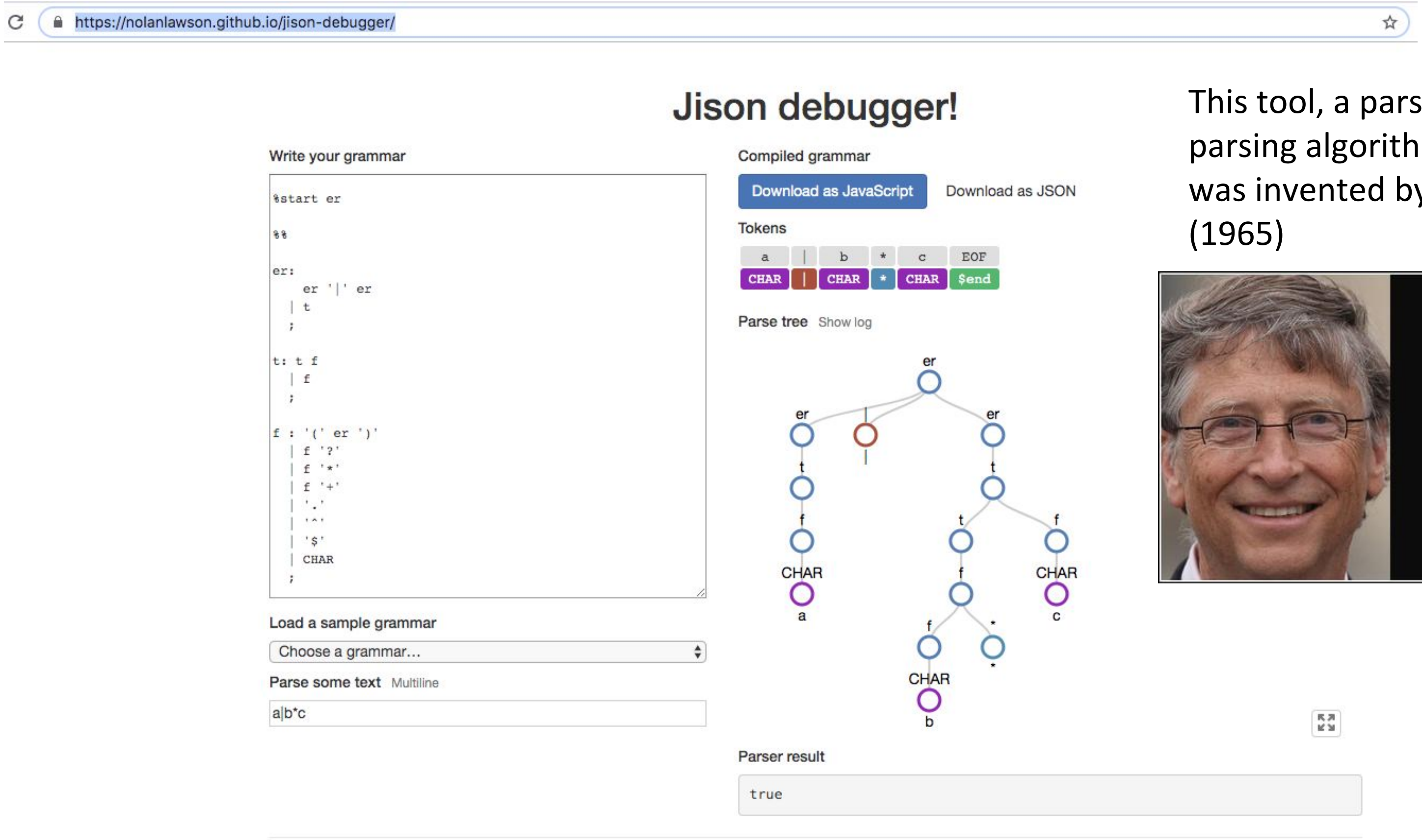
Stephen Cole Kleene invented regular expressions in 1951 to describe [McCulloch-Pitts neural networks]()

Un Programa que Evalúa Expresiones

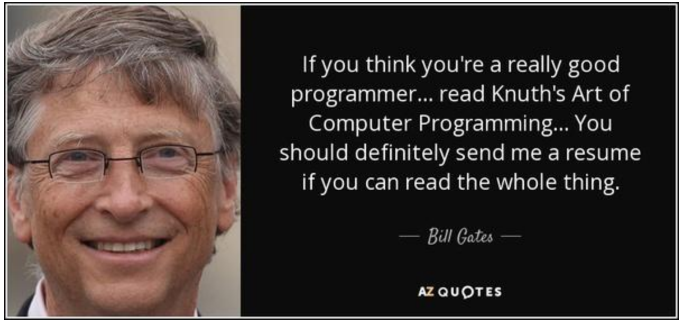
https://nolanlawson.github.io/jison-debugger/

```
%lex
%%
[0-9]+              return 'NUMBER'
"-"                 return '-'
.                   return 'INVALID'
/lex

%%
es: e              {return $1} ;
e : e '-' e        {$$ = $1-$3}
  | NUMBER         {$$ = Number($1)} ;
```

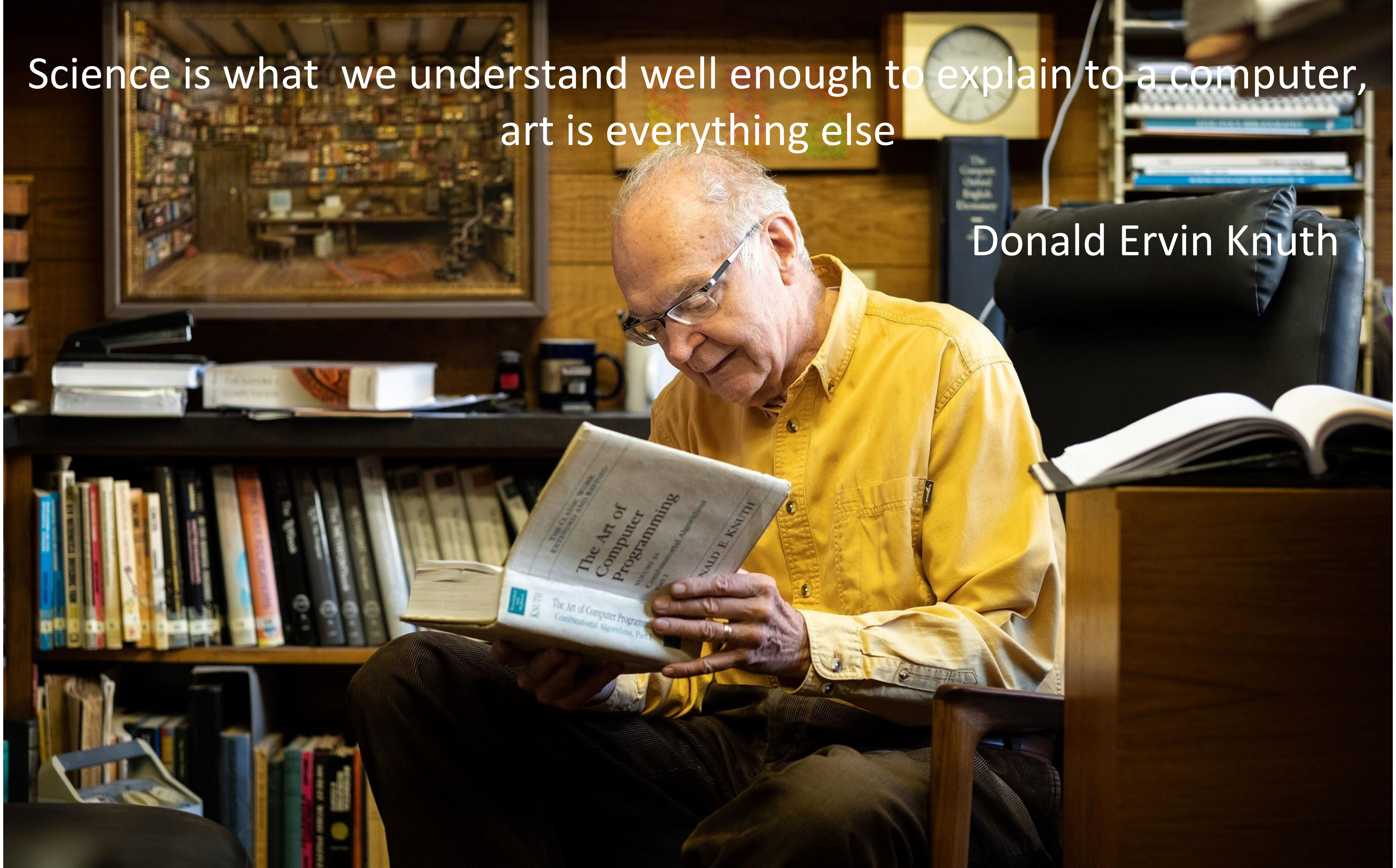
Parser Generators: an example



This tool, a parser generator uses a parsing algorithm known as LALR that was invented by Donald Ervin Knuth (1965)

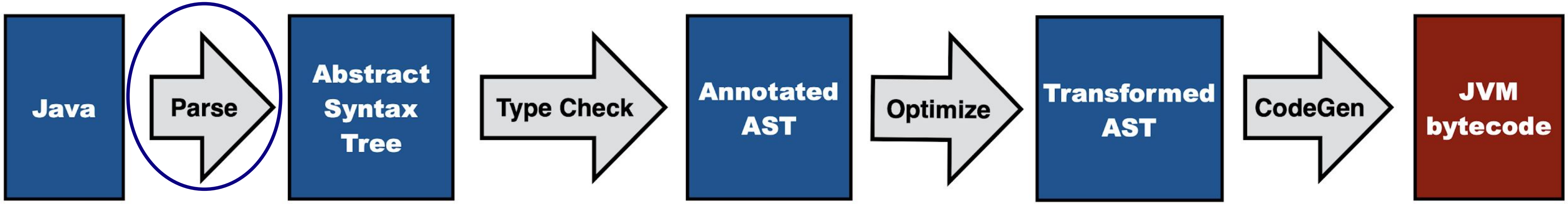


Science is what we understand well enough to explain to a computer, art is everything else



Donald Ervin Knuth

The Phases of a Translator



A programming language translator usually consists of a sequence of stages

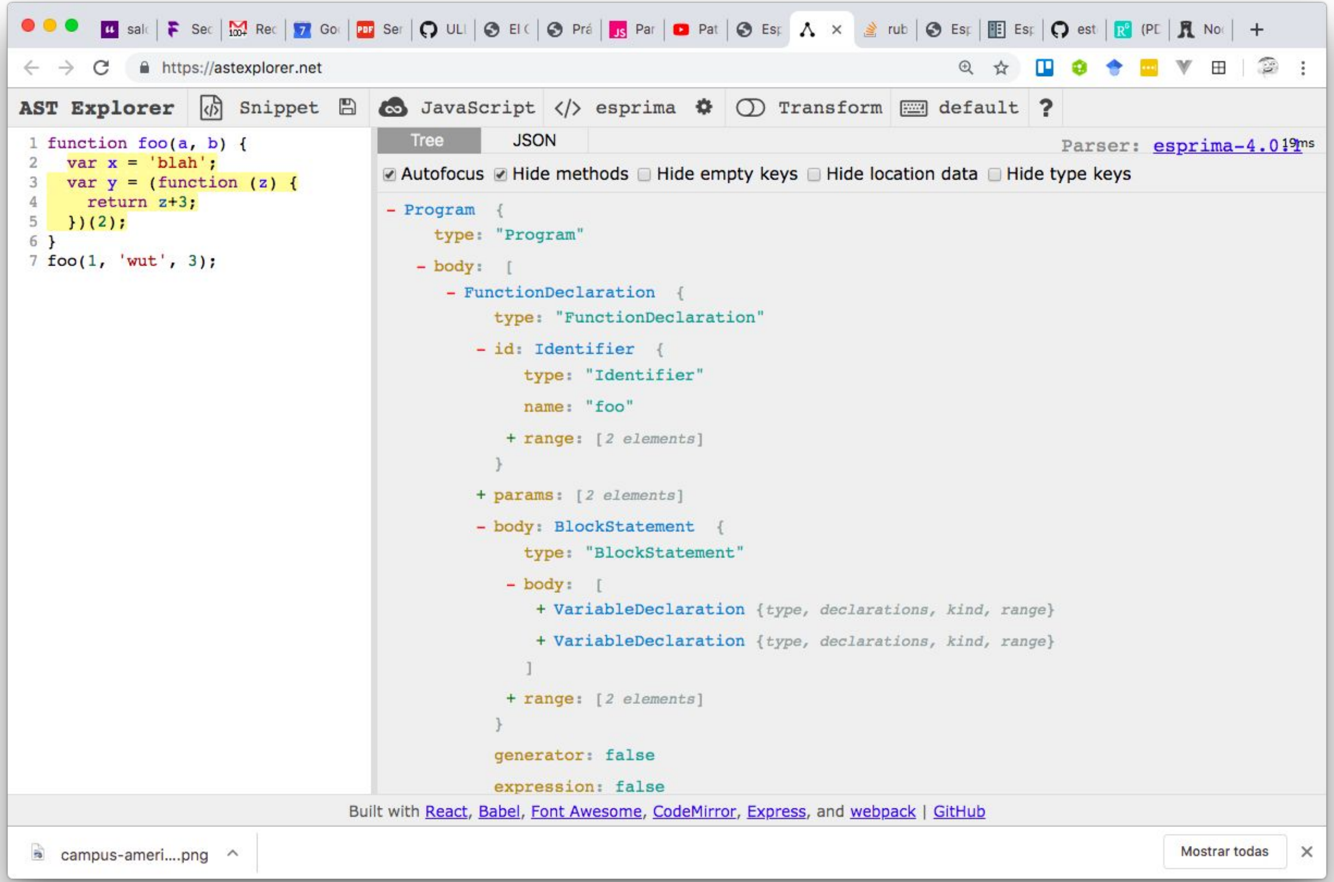
Lexer:

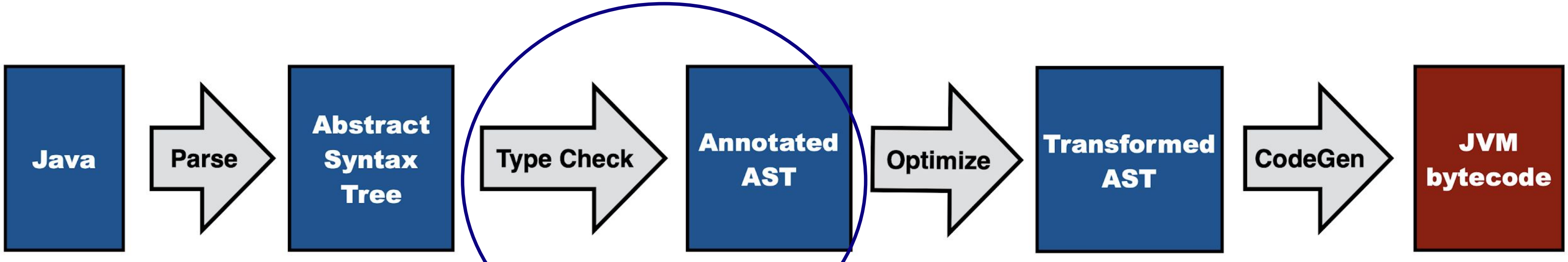
- Skips the comments and whitespaces and produces the stream of tokens for numbers, identifiers, reserved words, etc

Parser:

- Reads the stream of tokens, check that it complies with the syntactic rules and produces the *Abstract Syntax Tree*: a data structure representing the underlying syntactic structure of the input program

The *Abstract Syntax Tree*: a data structure representing the underlying syntactic structure of the input program: https://astexplorer.net/



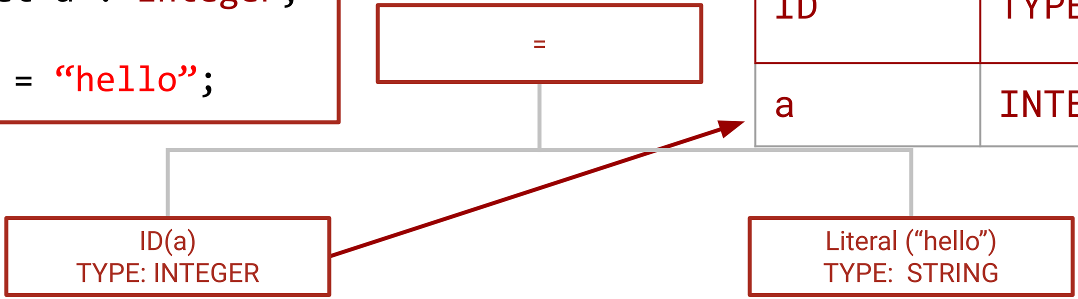


- Receives as input the abstract syntax tree
- Checks that the program complies with the static semantic rules of the language
- Performs name analysis, relating uses of names to declarations of names
- Checks that the types of arguments of operations are consistent with their specification

Input Program

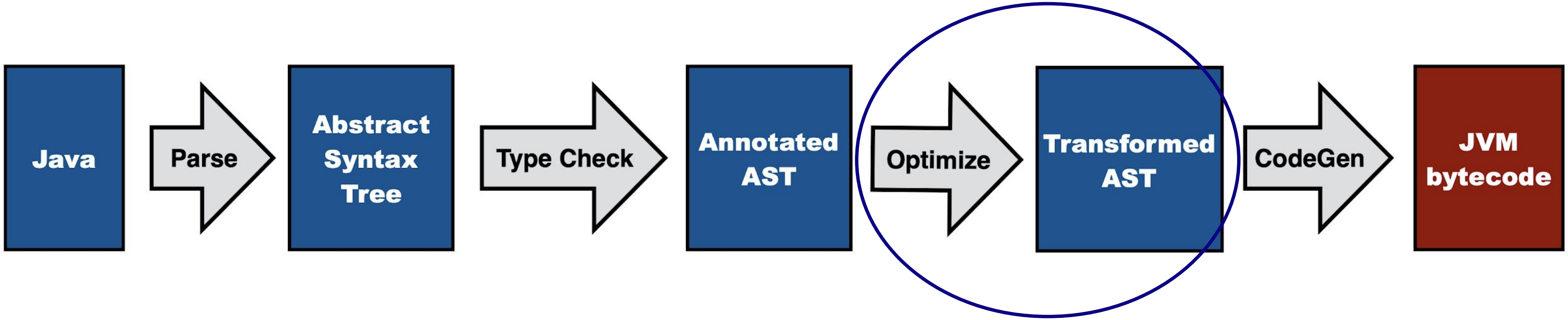
```
let a : integer;

a = “hello”;
```

AST



Symbol Table

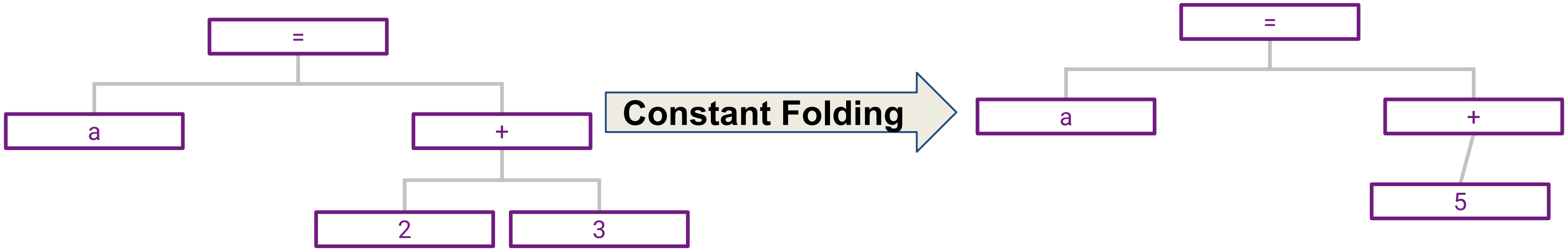
ID	TYPE
a	INTEGER

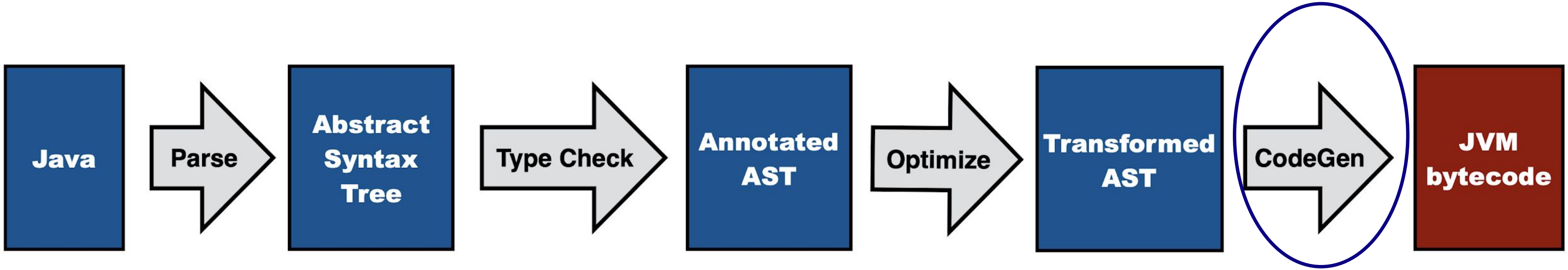


- Applies transformations that improve the program in various goals
- Goals: execution time, memory consumption, energy consumption, etc.
- Examples of transformations: Constant folding, Constant propagation, Loop invariants

Input Program

```
a = 2+3;
```

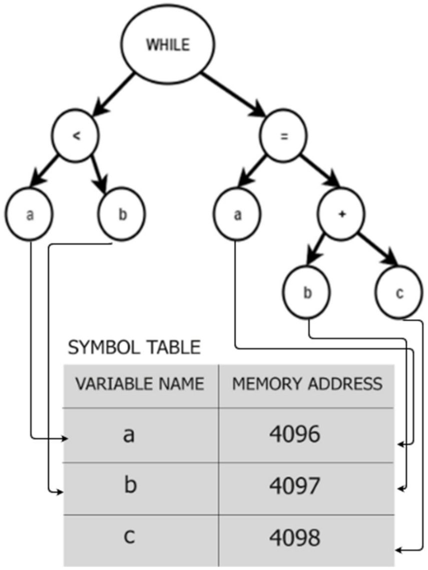




- Transforms abstract syntax tree to instructions for a particular computer architecture

Input Program

```
while (a < b) do
  a = b + c
end while
```

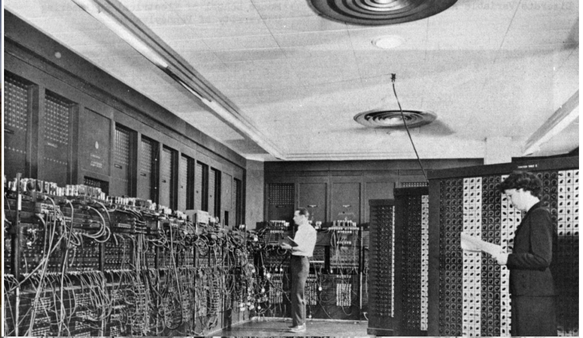


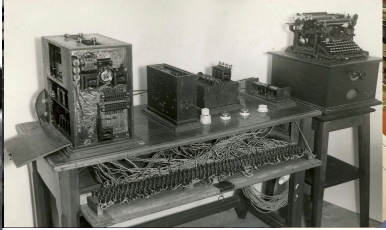
CODE GENERATION

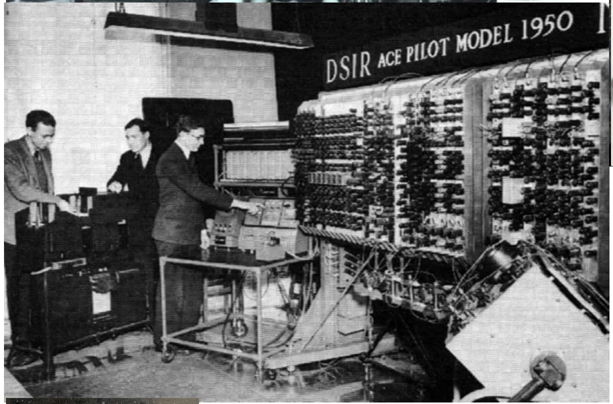
```
//Translating guard
L1:
MOV R0,[4096]
MOV R1,[4097]
LT R0,R1
JZ R0,L2
```

```
//Translating body
MOV R0,[4097]
MOV R1,[4098]
ADD R0,R1
MOV [4096],R0
JMP L1
L2:
```

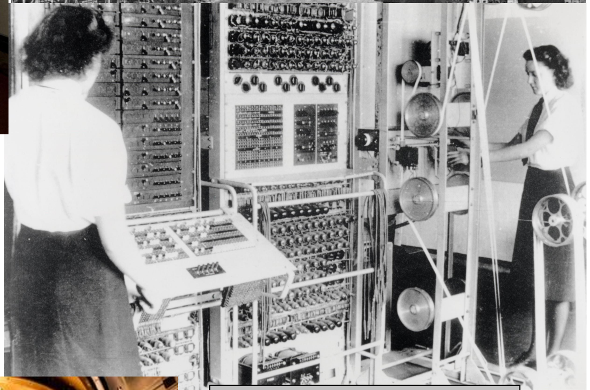

Storage
Control and
Arithmetic

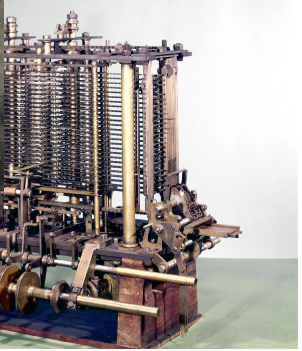


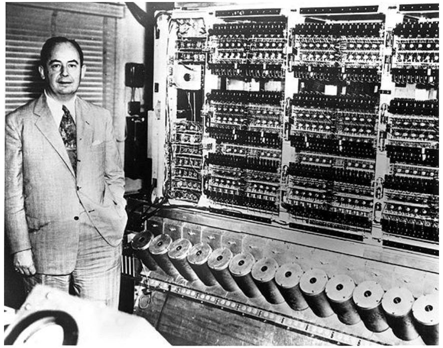


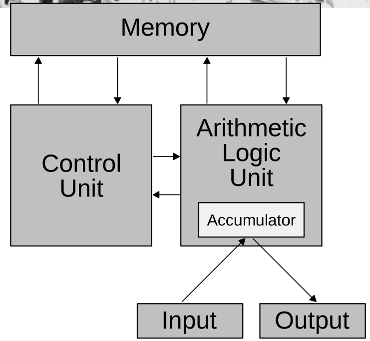

DSIR ACE PILOT MODEL 1950

¡Gracias!








Memory
Control
Unit
Arithmetic
Logic
Unit
Accumulator
Input
Output