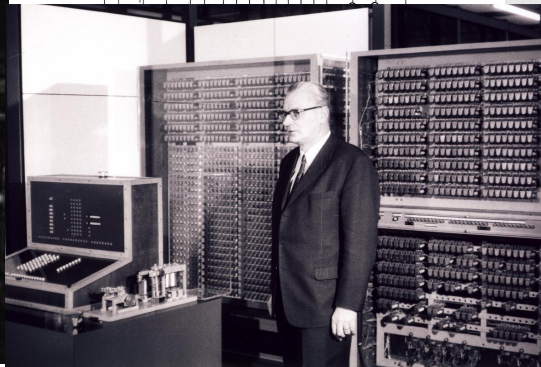
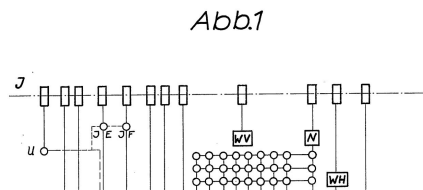


En 1935 Zuse construye la Z1. Leía las instrucciones desde una cinta perforada de 35 mm. No era una máquina Turing completa



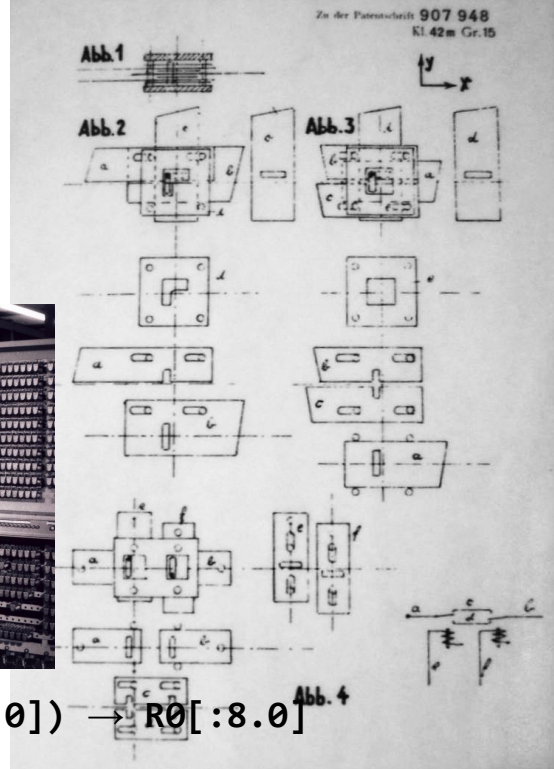
The Henschel Hs 129B ground attack aircraft

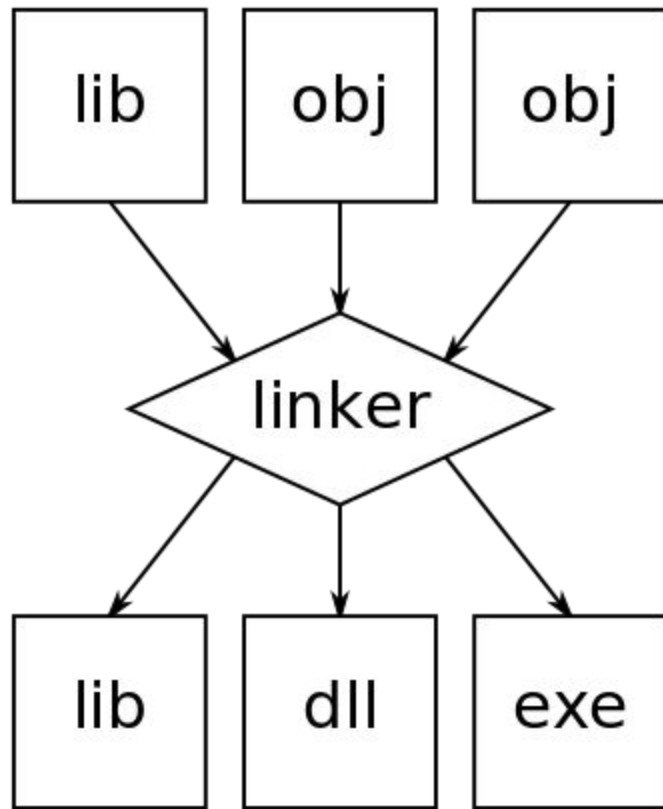
Zu der Patentschrift: 924 107
Kl.42m Gr.15



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

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





Rear Admiral (then Commodore) Grace M. Hopper, 1984. By 1952, Hopper had finished her program linker. The term compiler was coined by Hopper. It was written for the A-0 System.

The Algol 60 people



GLENN RODNEY ALAN
STUDENT'S NAME

535 68 8500 SENIOR WINTER 79 12/30/78
STUDENT NO. CLASS STANDING QUARTER & YEAR DATE

EASTERN WASHINGTON UNIVERSITY

—STUDENT REGISTRATION CONFIRMATION—

COURSE SEQUENCE NO.	COURSE TITLE	DEPT ABBREV.	CRED.	REPEAT OR WITHDRAWAL OR PENDING	DAYS OF THE WEEK	START TIME	END TIME	CLASS ROOM LOCATION
"HAPPINESS IS WINTER QUARTER AT EASTERN WASHINGTON UNIVERSITY"	14 230 01 CS	FORTRAN PROGRAMMIN	3		M, W, F	1200	100	P1103
	14 231 01 CS	COM PROG PROJECTS	2		ARR	ARR	ARR	ARR
	54 212 01 HUM	MUS IN HUMANITIES	5		DAILY	900	1000	MB247
	62 121 01 PHY	DESCRIP ASTRONOMY	5		DAILY	1100	1200	SC151

THIS IS A CONFIRMATION OF 15 CREDIT HOURS.

GLENN RODNEY ALAN
SUTTON HALL BOX 908
EWSC CHENEY WA 99004

CHECK CAREFULLY — REPORT ANY DISCREPANCIES TO REGISTRARS OFFICE. ALL CORRECTIONS TO YOUR REGISTRATION MUST BE MADE DURING REGULAR SCHEDULED CHANGE PERIOD. THIS FORM MUST BE PRESENTED FOR ANY SCHEDULE CHANGES OR CORRECTIONS.

YOUR CORRECT REGISTRATION IS YOUR RESPONSIBILITY

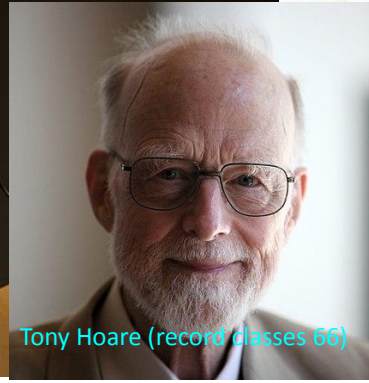
[OO History: Simula and Smalltalk](#)



Kristin Nygaard and Ole-Johan Dahl at the Norwegian Computing Center. Simula67

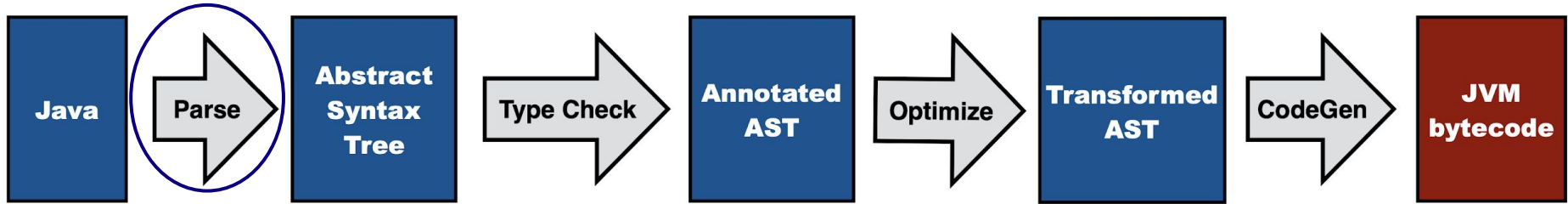


Alan Kay (Smalltalk)



Tony Hoare (record classes 66)

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:

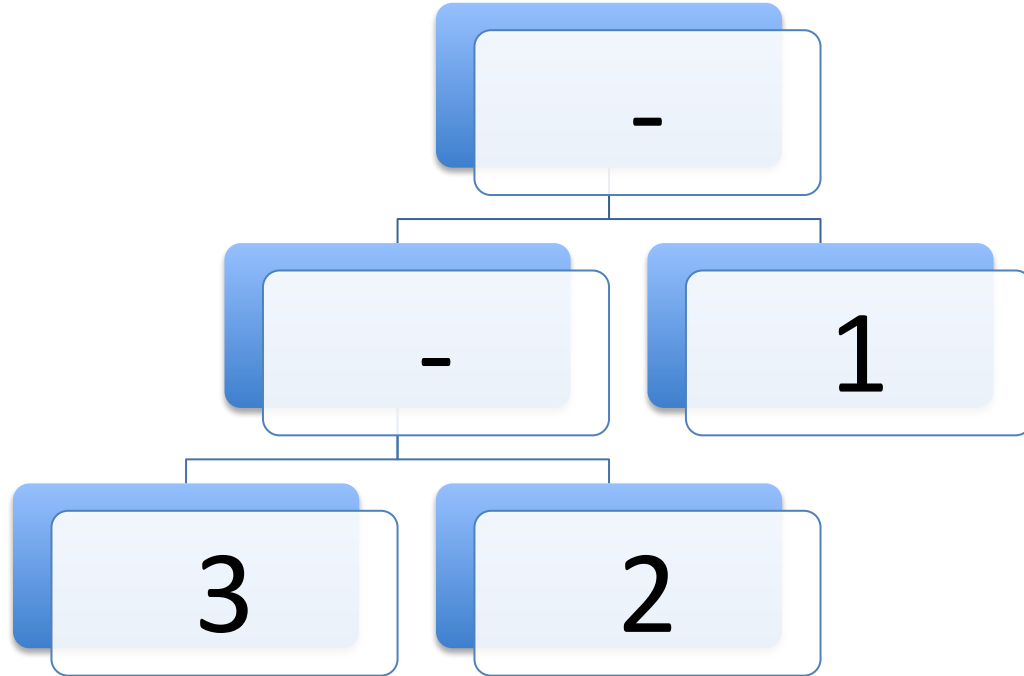
- Reads the stream of tokens, check that it complies with the syntactic rules and produces the *Abstract Syntax Tree*: a data structure representing the underlying syntactic structure of the input program

3 - 2 - 1

Árbol Sintáctico Abstracto

3-2-1

(3-2)-1



Semántica 3 - 2 - 1

$0 = 1 - 1$

-

$1 = 3 - 2$

-

1

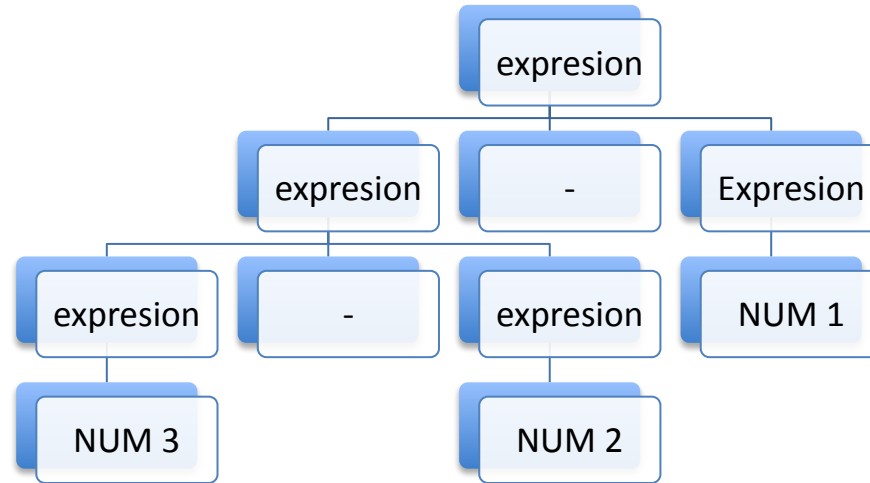
'3'

3

2

Gramática Independiente del Contexto

- $\text{expresion} \rightarrow \text{expresion} \text{ '-' } \text{expresion}$
- $\text{expresion} \rightarrow \text{NUMERO}$



3-2-1

Context Free Grammars

- expression \rightarrow expression '-' expression
- expression \rightarrow NUMERO

535 68 8500 SENIOR WINTER 79 12/30/78

STUDENT NO. CLASS STANDING QUARTER & YEAR DATE

—STUDENT REGISTRATION CONFIRMATION—

COURSE NO.	DEPT ABBREV.	COURSE TITLE	CRED.	SECTIONS WITHDRAWN OR IN PROG.	DAYS OF THE WEEK	START TIME	END TIME	CLASS ROOM LOCATION
01	CS	FORTRAN PROGRAMMIN	3		M,W,F	1200	100	P1103
01	CS	COM PRG PROJCTS	2		ARR	ARR	ARR	ARR
01	HUM	MUS IN HUMANITIES	5		DAILY	900	1000	HR247
01	PHY	DESCRIP ASTRONOMY	5		DAILY	1100	1200	SC151

THIS IS A CONFIRMATION OF 15 CREDIT HOURS.

GLENN RODNEY ALAN
SUTTON HALL BOX 908
EWING CHENEY WA. 99004

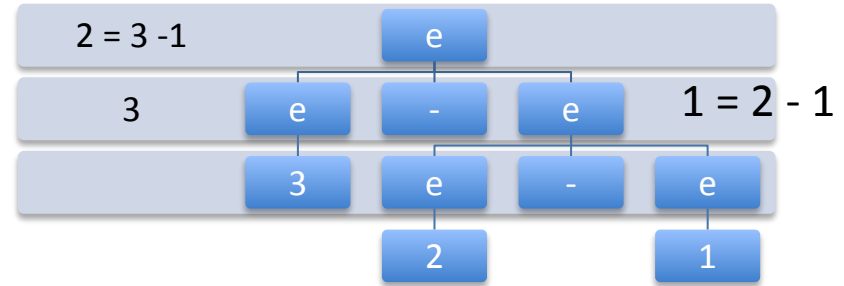
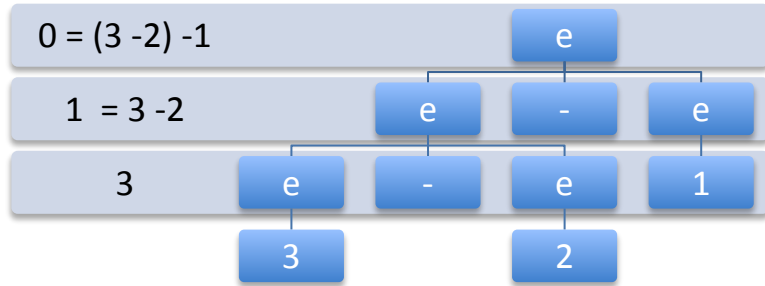
DISCREPANCIES TO REGISTRARS OFFICE
MADE DURING REGULAR SCHEDULED CHANGE
ANY SCHEDULE CHANGES OR CORRECTIONS
ARE YOUR RESPONSIBILITY

Noam Chomsky teaching linguistics
(1956)



Gramática Ambigua

- $\text{expression} \rightarrow \text{expression} \text{ '-' } \text{expression}$
- $\text{expression} \rightarrow \text{NUMERO}$

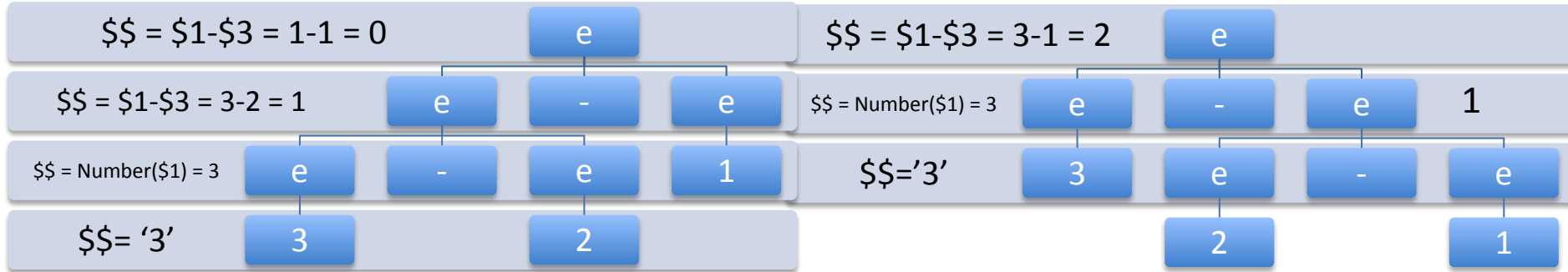


Esquema de Traducción (yacc)

$e \rightarrow e \text{ '-' } e \quad \{ \$\$ = \$1 - \$3; \}$

$e \rightarrow \text{NUM} \quad \{ \$\$ = \text{Number}(\$1); \}$

3-2-1

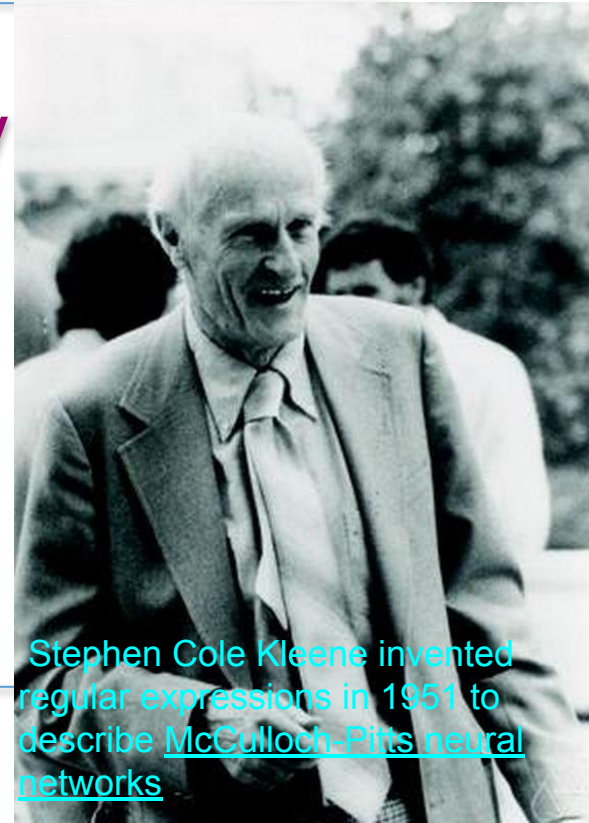
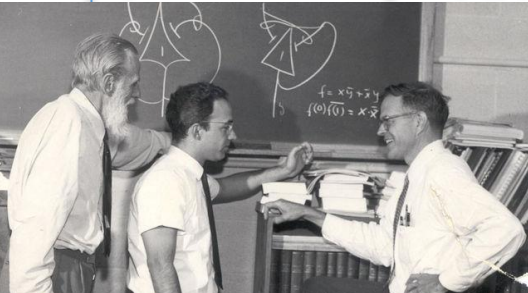


Análisis Léxico y Expresiones Regulares

`[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

<https://noianlawson.github.io/jison-debugger/>



Jison debugger!

Write your grammar

```
%start er
%%
er:
  er '|' er
  | t
  ;

t: t f
  | f
  ;

f: '(' er ') '
  | f '?'
  | f '*'
  | f '+'
  | '.'
  | '^'
  | '$'
  | CHAR
  ;
```

Load a sample grammar

Choose a grammar...

Parse some text Multiline

alb*c

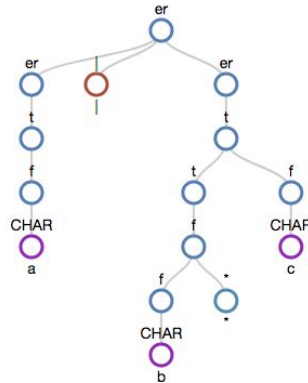
Compiled grammar

Download as JavaScript Download as JSON

Tokens

a | b * c EOF
CHAR CHAR * CHAR \$end

Parse tree Show log



Parser result

true

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



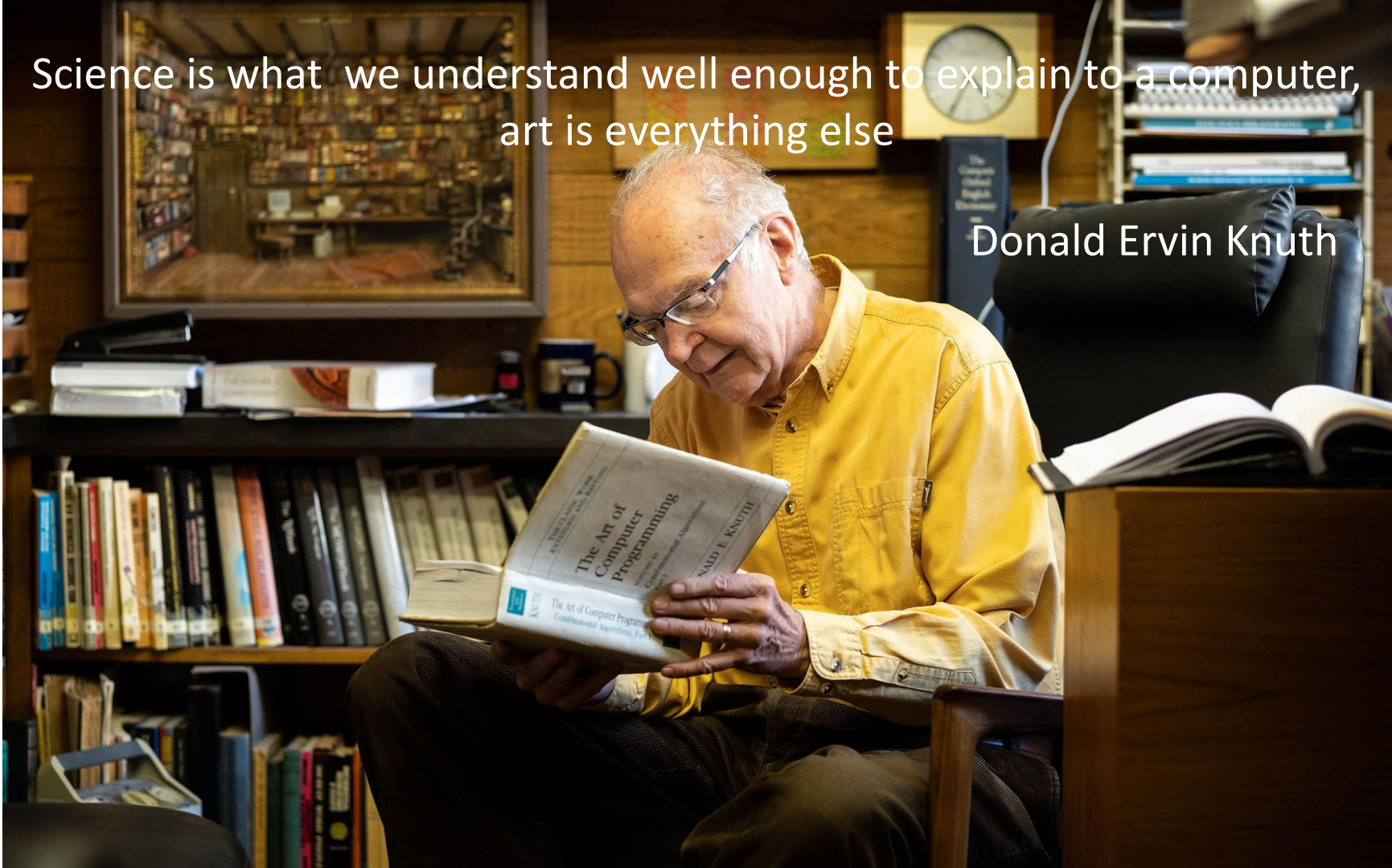
If you think you're a really good programmer... read Knuth's Art of Computer Programming... You should definitely send me a resume if you can read the whole thing.

— Bill Gates —

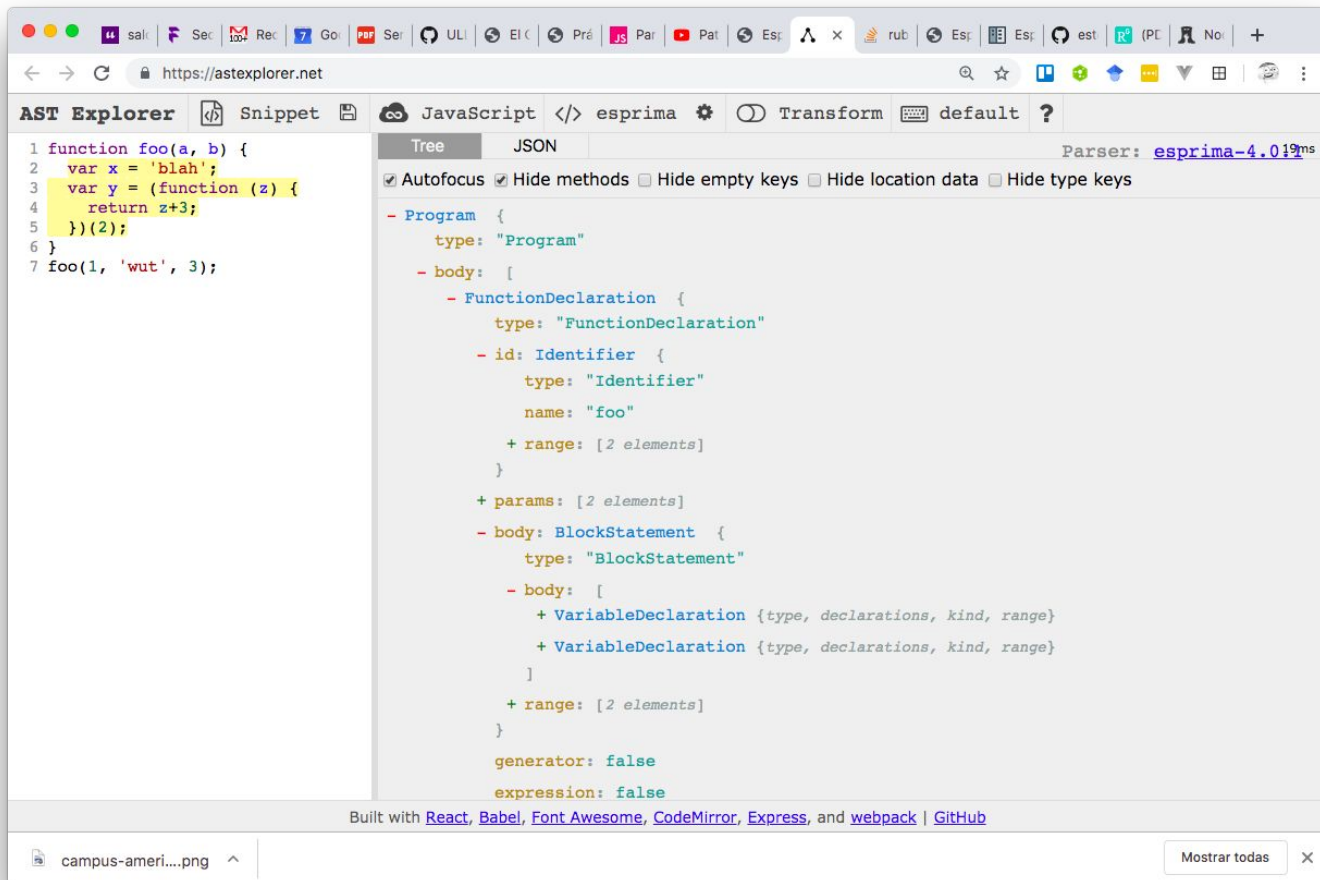
AZ QUOTES

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

Donald Ervin Knuth



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



The screenshot shows the AST Explorer interface. On the left, the source code is displayed with line numbers 1 through 7. The code is a function named 'foo' that takes two arguments, 'a' and 'b'. Inside the function, there is a variable 'x' assigned the value 'blah', and a variable 'y' assigned a function that takes 'z' and returns 'z+3'. The function 'foo' is called with arguments 1 and 'wut', and the result is passed to a function that takes '3' as an argument.

```
1 function foo(a, b) {
2   var x = 'blah';
3   var y = (function (z) {
4     return z+3;
5   })(2);
6 }
7 foo(1, 'wut', 3);
```

On the right, the AST is displayed in a tree view. The root node is 'Program', which has a 'body' array containing a 'FunctionDeclaration' node. The 'FunctionDeclaration' node has an 'id' of 'foo', 'params' of 'a, b', and a 'body' array containing a 'BlockStatement' node. The 'BlockStatement' node has a 'body' array containing two 'VariableDeclaration' nodes. The first 'VariableDeclaration' node has an 'id' of 'x' and a 'value' of 'blah'. The second 'VariableDeclaration' node has an 'id' of 'y' and a 'value' of a function expression. The function expression has a 'body' array containing a 'ReturnStatement' node with a 'value' of 'z+3'. The 'FunctionDeclaration' node also has a 'range' property of [2, 10]. The 'BlockStatement' node has a 'range' property of [2, 10]. The 'VariableDeclaration' nodes have 'range' properties of [2, 10] and [11, 10]. The 'ReturnStatement' node has a 'range' property of [11, 10].

Parser: [esprima-4.0.19ms](#)

Autofocus Hide methods Hide empty keys Hide location data Hide type keys

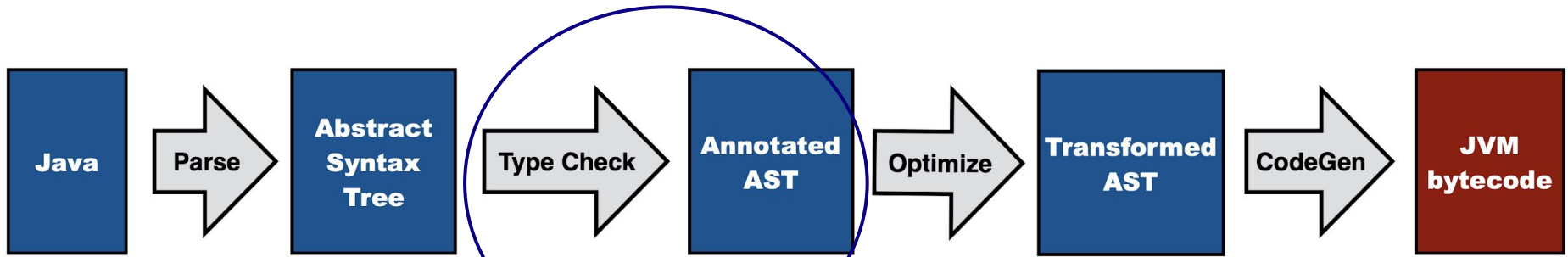
Tree JSON

```
- Program {
  type: "Program"
  - body: [
    - FunctionDeclaration {
      type: "FunctionDeclaration"
      - id: Identifier {
        type: "Identifier"
        name: "foo"
      }
      + params: [2 elements]
    }
    + body: BlockStatement {
      type: "BlockStatement"
      - body: [
        + VariableDeclaration {type, declarations, kind, range}
        + VariableDeclaration {type, declarations, kind, range}
      ]
      + range: [2 elements]
    }
    generator: false
    expression: false
  ]
}
```

Built with [React](#), [Babel](#), [Font Awesome](#), [CodeMirror](#), [Express](#), and [webpack](#) | [GitHub](#)

campus-ameri...png ^

Mostrar todas X



- 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

```

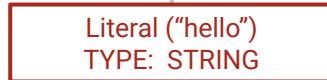
let a : integer;
a = "hello";
  
```

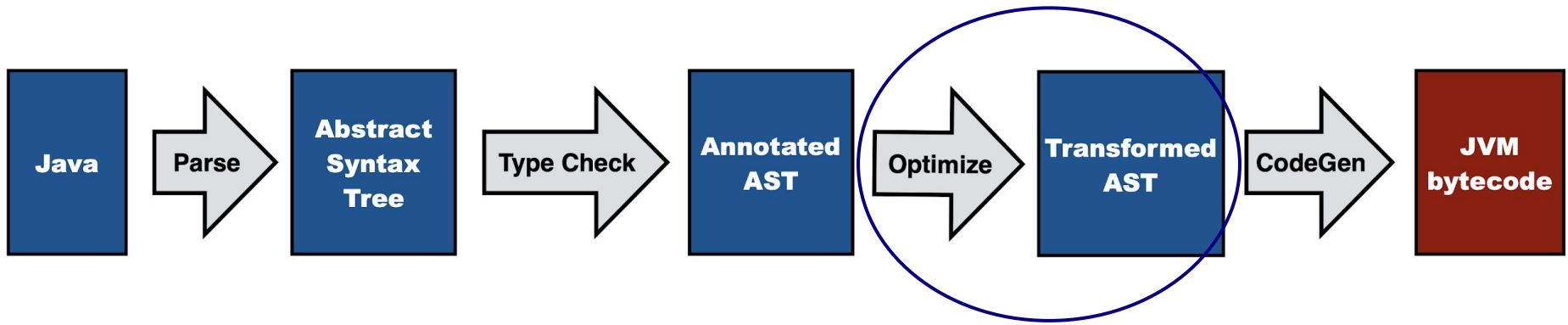
AST



Symbol Table

ID	TYPE
a	INTEGER

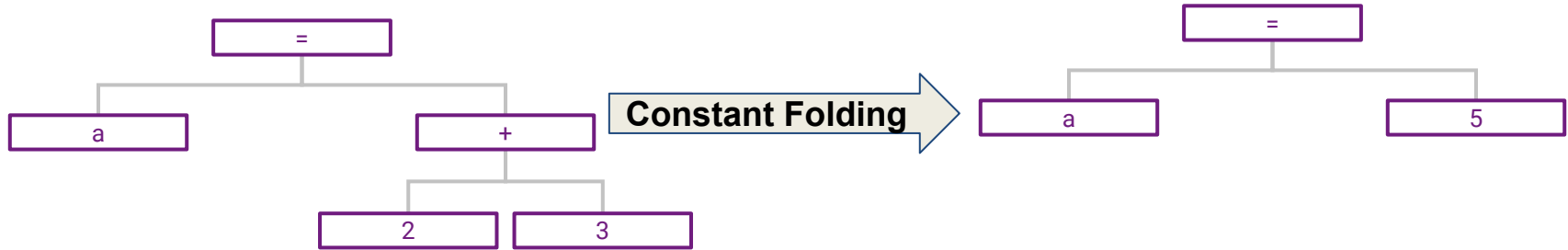


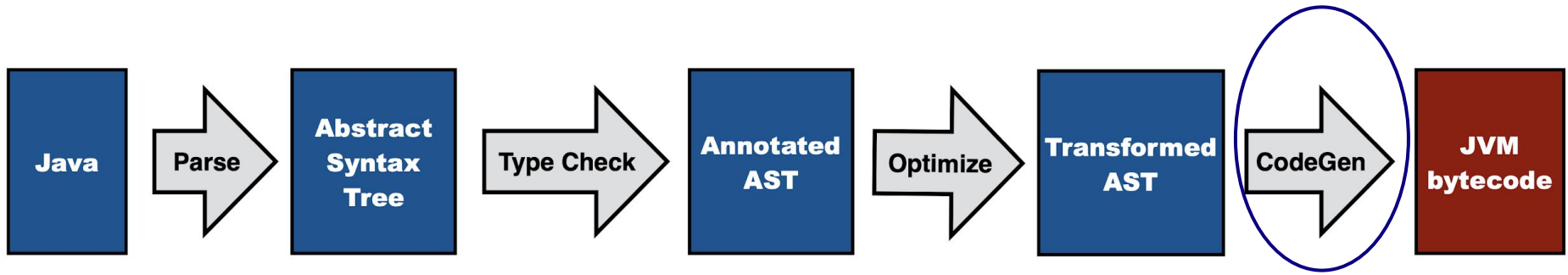


- Applies transformations that improve the program in various goals
- Goals: execution time, memory consumption, energy consumption, etc.
- Examples of transformations: Constant folding, Constant propagation, Loop invariants

Input Program

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



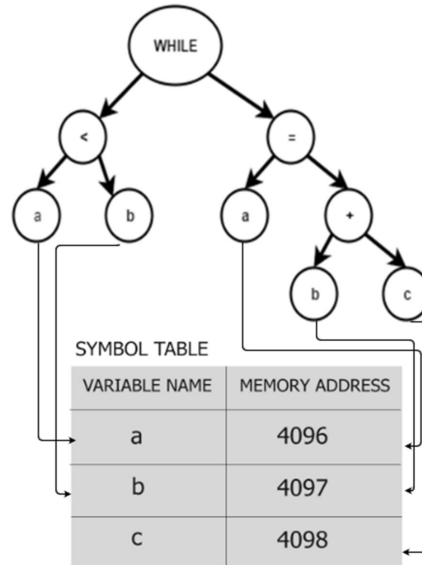


- Transforms abstract syntax tree to instructions for a particular computer architecture

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:
  
```