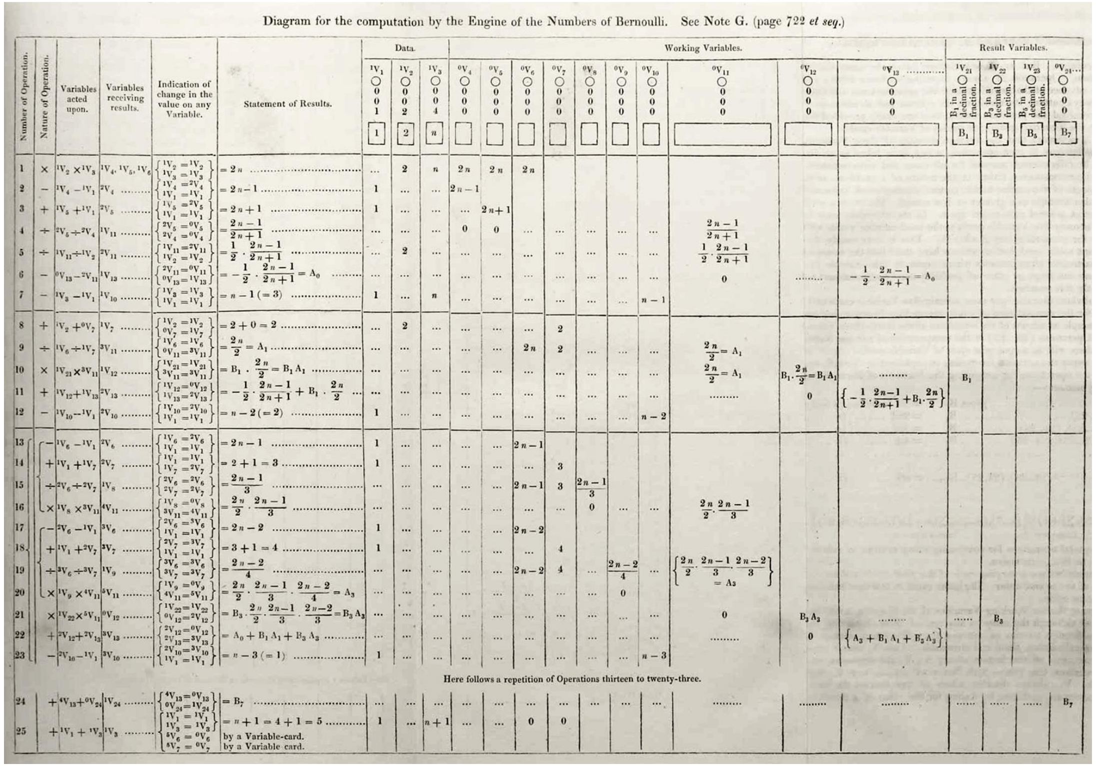
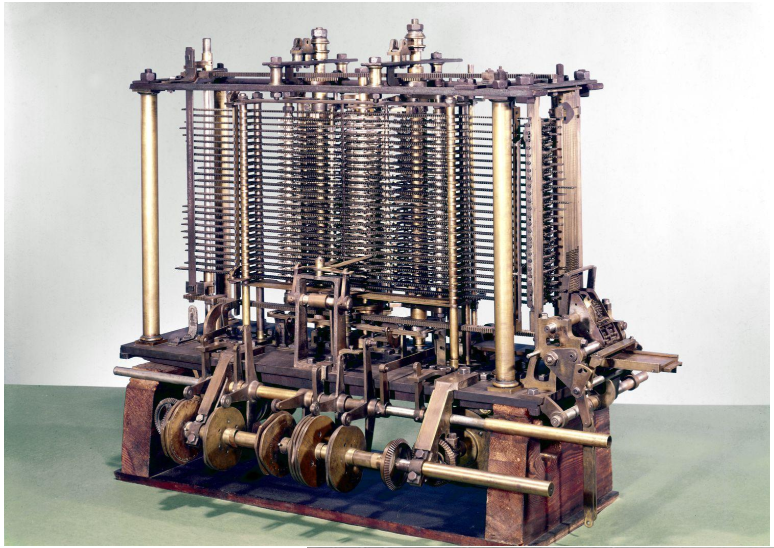


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.
1	×	$^1V_2 \times {}^1V_3$	$^1V_4, {}^1V_5, {}^1V_6$	$\left\{\begin{matrix}^1V_2 = {}^1V_2 \\ ^1V_3 = {}^1V_3\end{matrix}\right\}$	$= 2n$
2	−	$^1V_4 - {}^1V_1$	$^2V_4$	$\left\{\begin{matrix}^1V_4 = {}^2V_4 \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= 2n - 1$
3	+	$^1V_5 + {}^1V_1$	$^2V_5$	$\left\{\begin{matrix}^1V_5 = {}^2V_5 \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= 2n + 1$
4	÷	$^2V_5 \div {}^2V_4$	$^1V_{11}$	$\left\{\begin{matrix}^2V_5 = {}^0V_5 \\ ^2V_4 = {}^0V_4\end{matrix}\right\}$	$= \frac{2n-1}{2n+1}$
5	÷	$^1V_{11} \div {}^1V_2$	$^2V_{11}$	$\left\{\begin{matrix}^1V_{11} = {}^2V_{11} \\ ^1V_2 = {}^1V_2\end{matrix}\right\}$	$= \frac{1}{2} \cdot \frac{2n-1}{2n+1}$
6	−	$^0V_{13} - {}^2V_{11}$	$^1V_{13}$	$\left\{\begin{matrix}^2V_{11} = {}^0V_{11} \\ ^0V_{13} = {}^1V_{13}\end{matrix}\right\}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$
7	−	$^1V_3 - {}^1V_1$	$^1V_{10}$	$\left\{\begin{matrix}^1V_3 = {}^1V_3 \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= n - 1 (= 3)$
8	+	$^1V_2 + {}^0V_7$	$^1V_7$	$\left\{\begin{matrix}^1V_2 = {}^1V_2 \\ ^0V_7 = {}^1V_7\end{matrix}\right\}$	$= 2 + 0 = 2$
9	÷	$^1V_6 \div {}^1V_7$	$^3V_{11}$	$\left\{\begin{matrix}^1V_6 = {}^1V_6 \\ ^0V_{11} = {}^3V_{11}\end{matrix}\right\}$	$= \frac{2n}{2} = A_1$
10	×	$^1V_{21} \times {}^3V_{11}$	$^1V_{12}$	$\left\{\begin{matrix}^1V_{21} = {}^1V_{21} \\ ^3V_{11} = {}^3V_{11}\end{matrix}\right\}$	$= B_1 \cdot \frac{2n}{2} = B_1 A_1$
11	+	$^1V_{12} + {}^1V_{13}$	$^2V_{13}$	$\left\{\begin{matrix}^1V_{12} = {}^0V_{12} \\ ^1V_{13} = {}^2V_{13}\end{matrix}\right\}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}$
12	−	$^1V_{10} - {}^1V_1$	$^2V_{10}$	$\left\{\begin{matrix}^1V_{10} = {}^2V_{10} \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= n - 2 (= 2)$
13	−	$^1V_6 - {}^1V_1$	$^2V_6$	$\left\{\begin{matrix}^1V_6 = {}^2V_6 \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= 2n - 1$
14	+	$^1V_1 + {}^1V_7$	$^2V_7$	$\left\{\begin{matrix}^1V_1 = {}^1V_1 \\ ^1V_7 = {}^2V_7\end{matrix}\right\}$	$= 2 + 1 = 3$
15	÷	$^2V_6 \div {}^2V_7$	$^1V_8$	$\left\{\begin{matrix}^2V_6 = {}^2V_6 \\ ^2V_7 = {}^2V_7\end{matrix}\right\}$	$= \frac{2n-1}{3}$
16	×	$^1V_8 \times {}^3V_{11}$	$^4V_{11}$	$\left\{\begin{matrix}^1V_8 = {}^0V_8 \\ ^3V_{11} = {}^4V_{11}\end{matrix}\right\}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3}$
17	−	$^2V_6 - {}^1V_1$	$^3V_6$	$\left\{\begin{matrix}^2V_6 = {}^3V_6 \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= 2n - 2$
18	+	$^1V_1 + {}^2V_7$	$^3V_7$	$\left\{\begin{matrix}^2V_7 = {}^3V_7 \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= 3 + 1 = 4$
19	÷	$^3V_6 \div {}^3V_7$	$^1V_9$	$\left\{\begin{matrix}^3V_6 = {}^3V_6 \\ ^3V_7 = {}^3V_7\end{matrix}\right\}$	$= \frac{2n-2}{4}$
20	×	$^1V_9 \times {}^4V_{11}$	$^5V_{11}$	$\left\{\begin{matrix}^1V_9 = {}^0V_9 \\ ^4V_{11} = {}^5V_{11}\end{matrix}\right\}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{4} = A_3$
21	×	$^1V_{22} \times {}^5V_{11}$	$^0V_{12}$	$\left\{\begin{matrix}^1V_{22} = {}^1V_{22} \\ ^0V_{12} = {}^2V_{12}\end{matrix}\right\}$	$= B_3 \cdot \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3} = B_3 A_3$
22	+	$^2V_{12} + {}^2V_{13}$	$^3V_{13}$	$\left\{\begin{matrix}^2V_{12} = {}^0V_{12} \\ ^2V_{13} = {}^3V_{13}\end{matrix}\right\}$	$= A_0 + B_1 A_1 + B_3 A_3$
23	−	$^2V_{10} - {}^1V_1$	$^3V_{10}$	$\left\{\begin{matrix}^2V_{10} = {}^3V_{10} \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= n - 3 (= 1)$
					Here follows a repetition of Operations thirteen to twenty-three.
24	+	$^4V_{13} + {}^0V_{24}$	$^1V_{24}$	$\left\{\begin{matrix}^4V_{13} = {}^0V_{13} \\ ^0V_{24} = {}^1V_{24}\end{matrix}\right\}$	$= B_7$
25	+	$^1V_1 + {}^1V_3$	$^1V_3$	$\left\{\begin{matrix}^1V_1 = {}^1V_1 \\ ^1V_3 = {}^1V_3 \\ ^5V_6 = {}^0V_6 \\ ^5V_7 = {}^0V_7\end{matrix}\right\}$	$= n + 1 = 4 + 1 = 5$; by a Variable-card. by a Variable card.



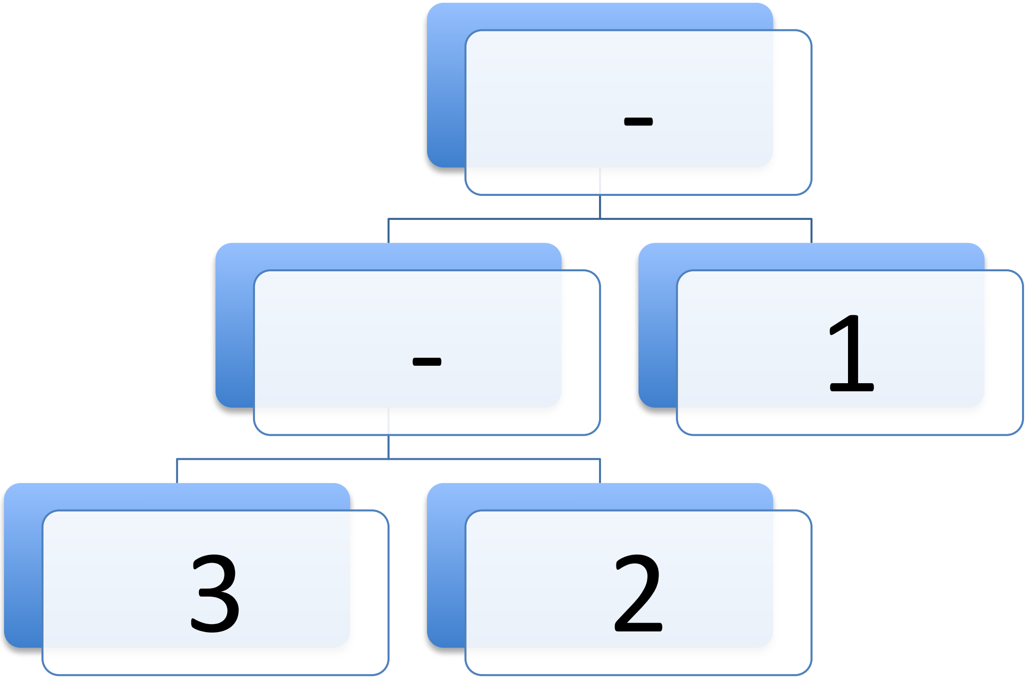
… La nota G estaba dedicada a los números de Bernoulli; en este apartado Ada describe con detalle las operaciones mediante las cuales las tarjetas perforadas "tejerían" una secuencia de números en la máquina analítica. Este código está considerado como el primer algoritmo específicamente diseñado para ser ejecutado por un ordenador, aunque nunca fue probado ya que la máquina nunca llegó a construirse.
(Wikipedia)

3 - 2 - 1

Árbol Sintáctico Abstracto

3-2-1

(3-2)-1

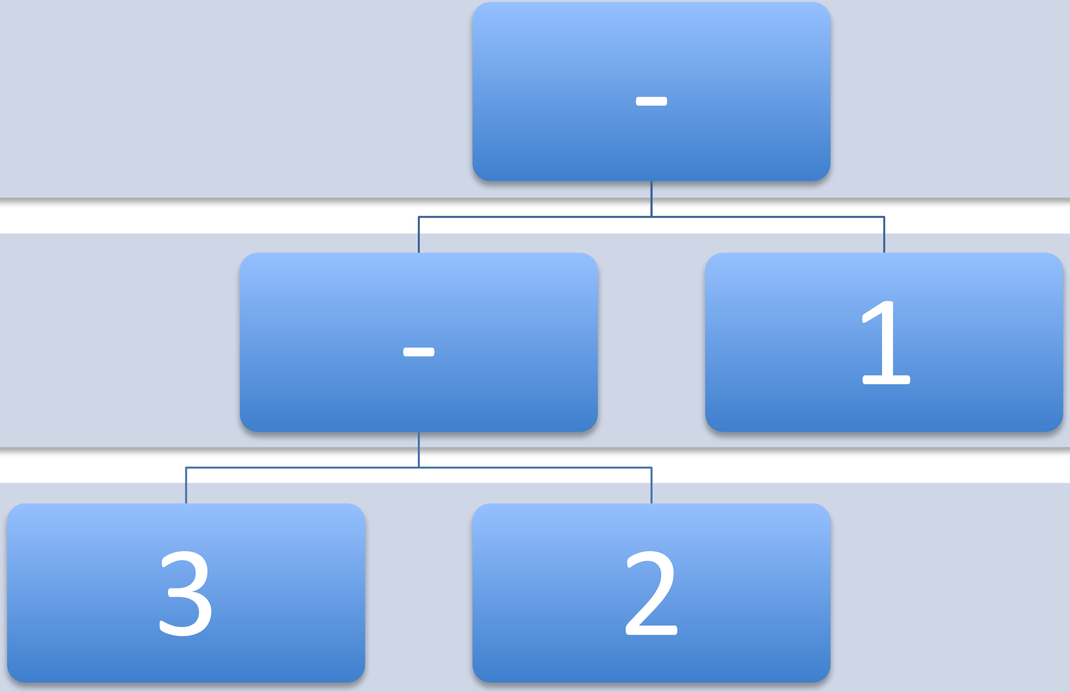


Semántica 3 - 2 - 1

0 = 1 - 1

1 = 3 -2

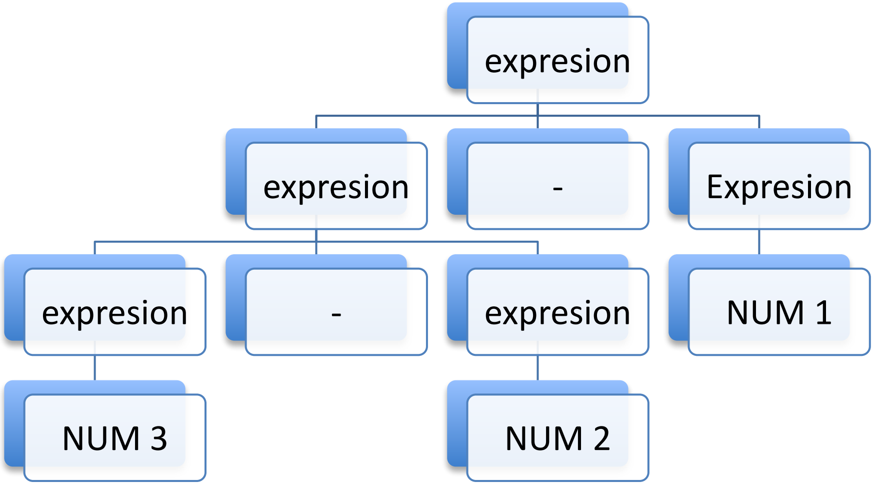
‘3’



Gramática Independiente del Contexto

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

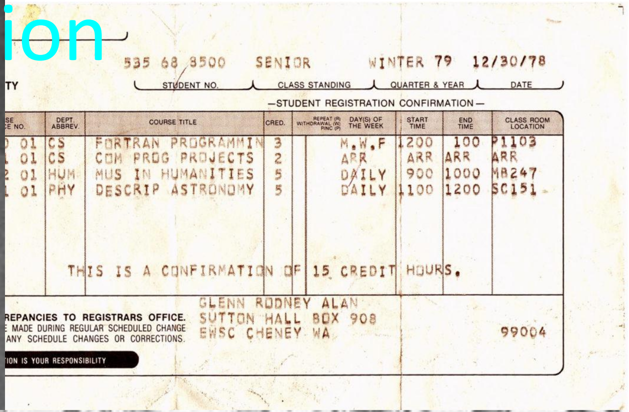
3-2-1



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) PNC (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.

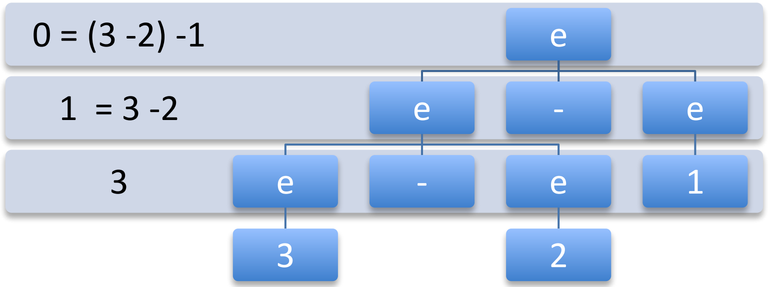
GLENN RODNEY ALAN
SUTTON HALL BOX 908
EWSC CHENEY WA 99004

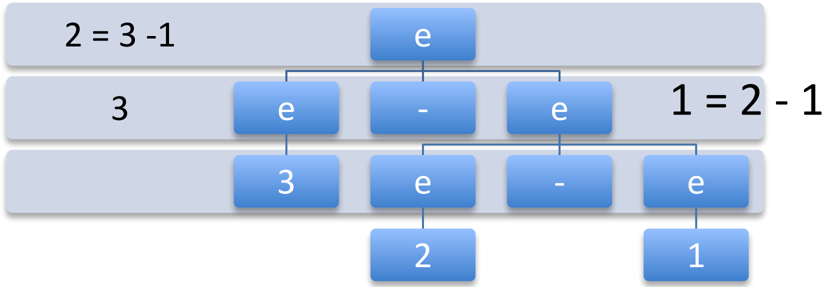
ION IS YOUR RESPONSIBILITY

The Algol 60 people

Gramática Ambigua

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



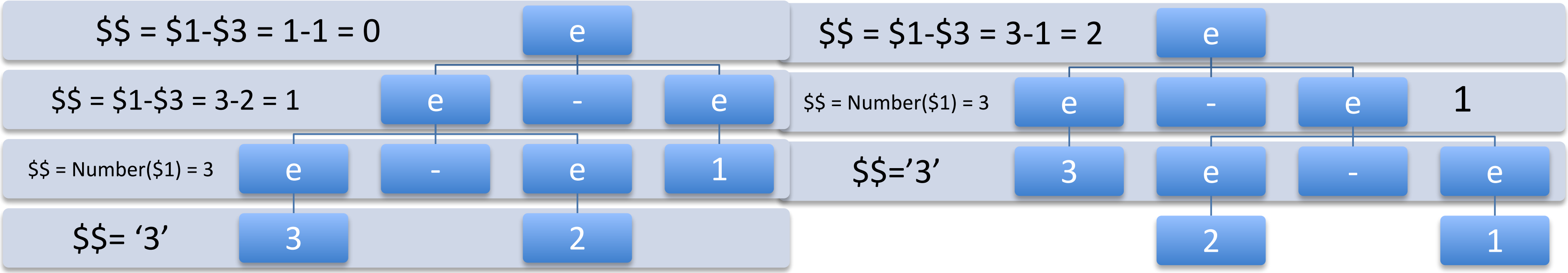


Esquema de Traducción (yacc)

`e → e '-' e    { $$ = $1 - $3; }`

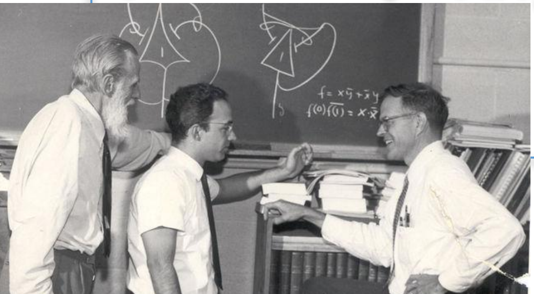
`e → NUM    { $$ = Number($1); }`

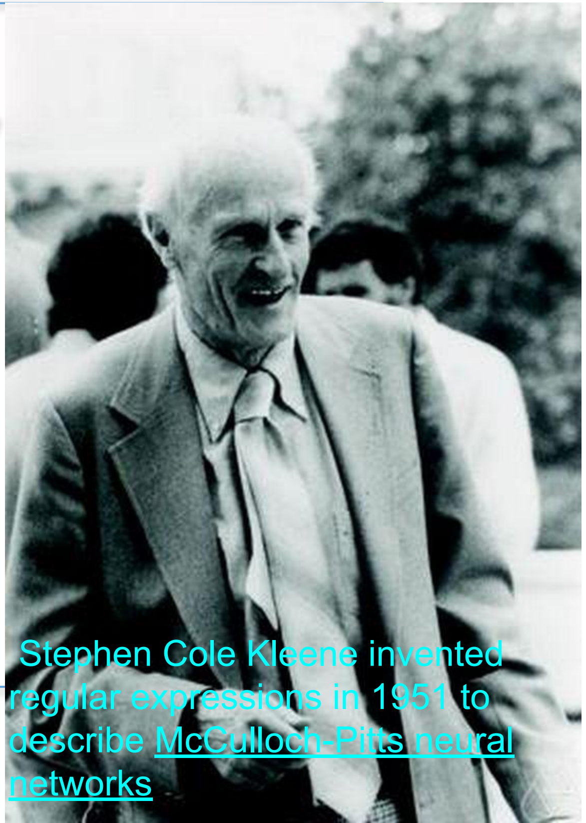
3–2-1



Análisis Léxico y Expresiones Regulares

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

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

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





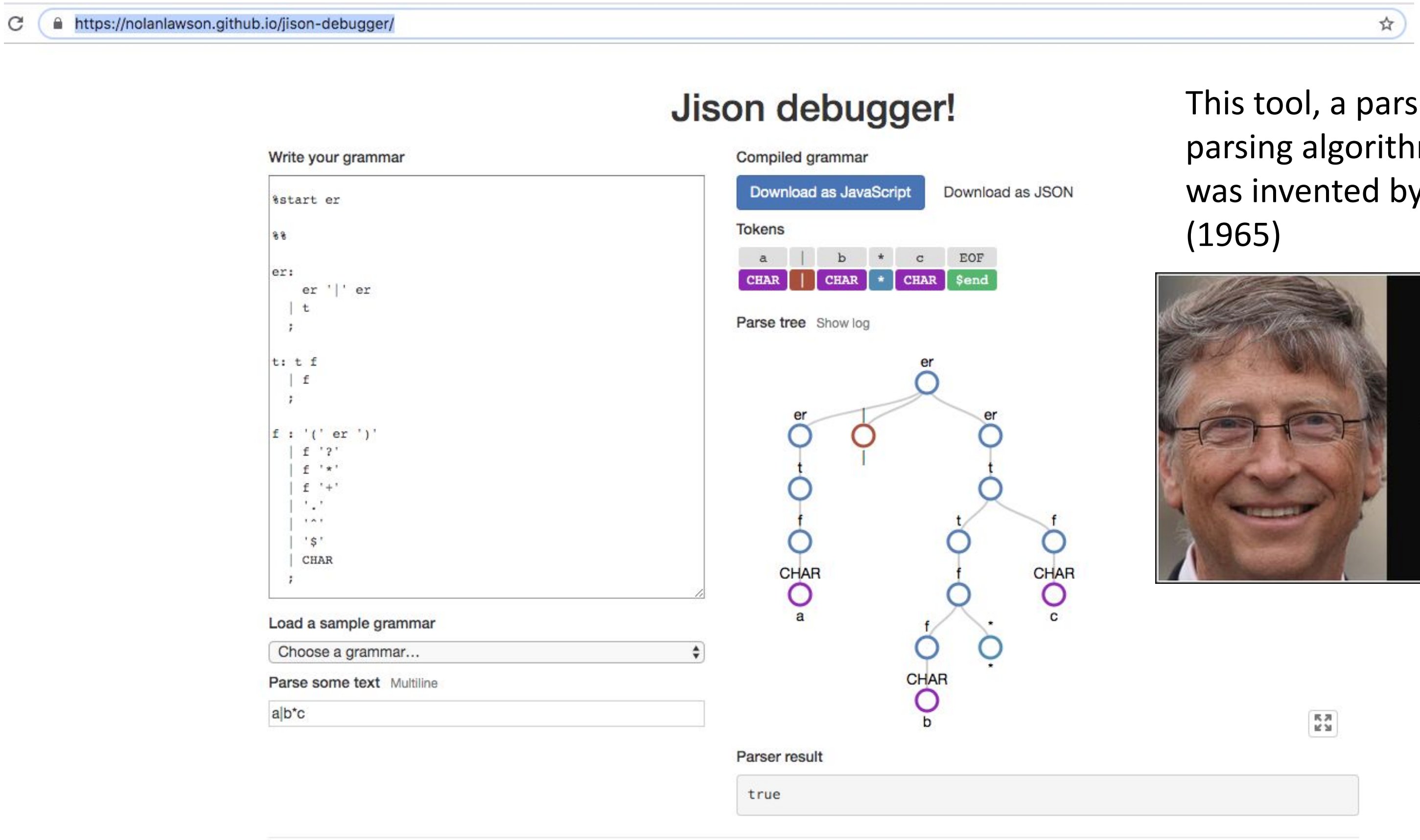
Stephen Cole Kleene invented regular expressions in 1951 to describe [McCulloch-Pitts neural networks]()

Un Programa que Evalúa Expresiones

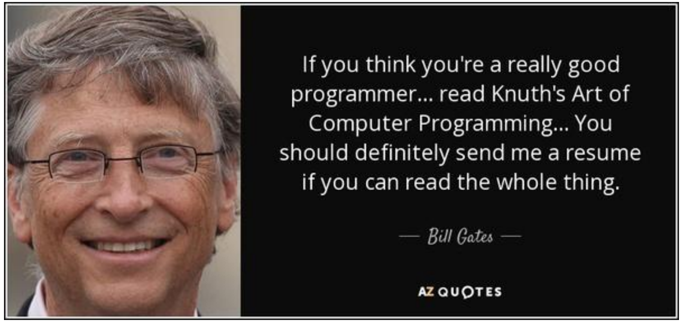
https://nolanlawson.github.io/jison-debugger/

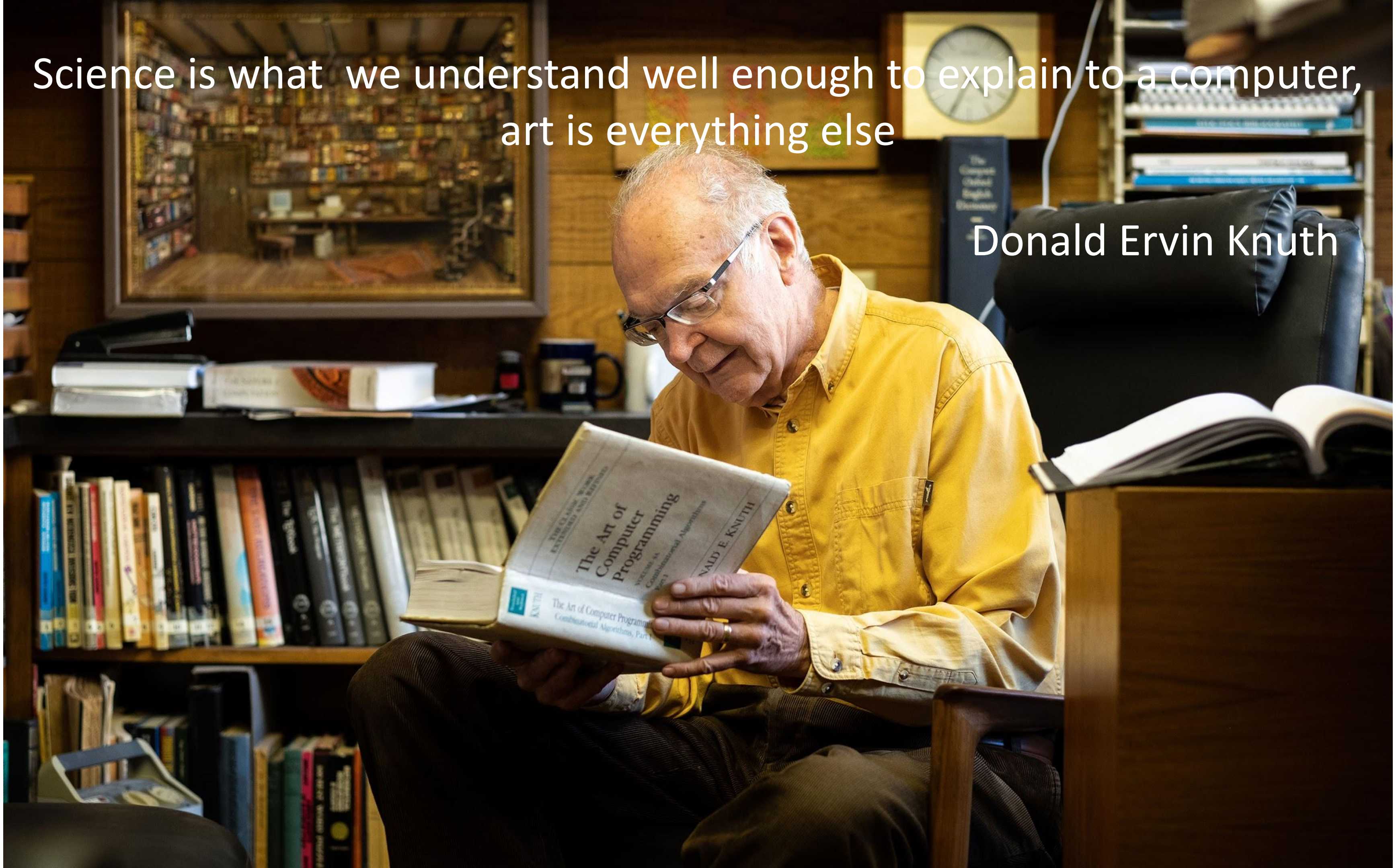
```
%lex
%%
[0-9]+              return 'NUMBER'
"-"                 return '-'
.                   return 'INVALID'
/lex

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

Parser Generators: an example

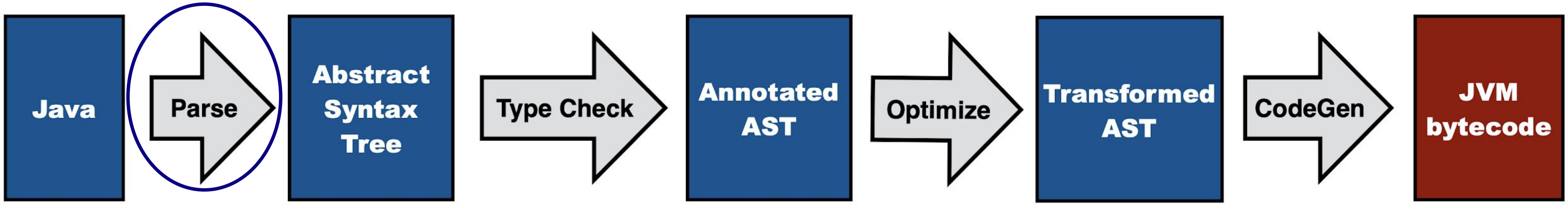


This tool, a parser generator uses a parsing algorithm known as LALR that was invented by Donald Ervin Knuth (1965)




Science is what we understand well enough to explain to a computer,
art is everything else
Donald Ervin Knuth
The Art of Computer Programming

The Phases of a Translator



A programming language translator usually consists of a sequence of stages

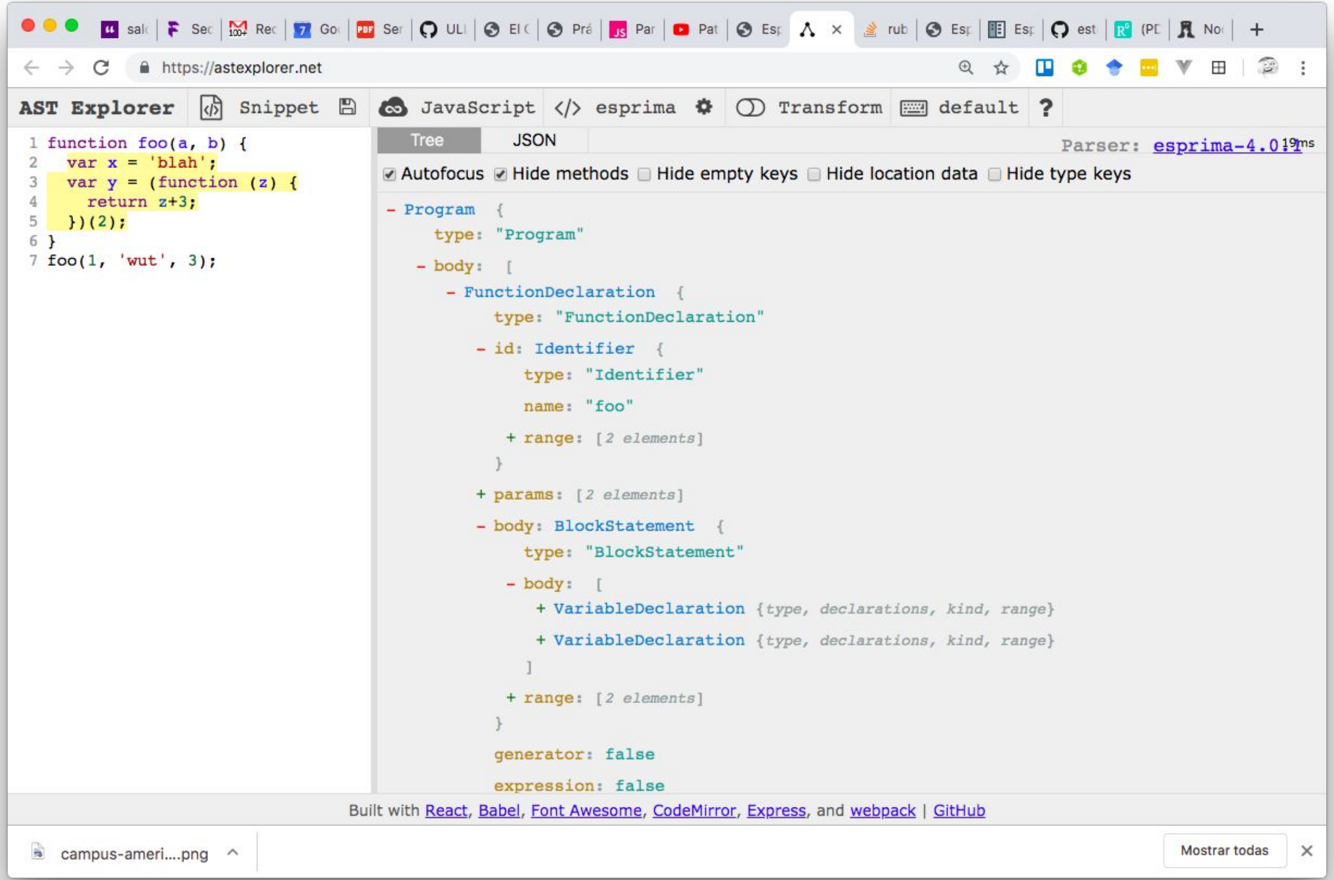
Lexer:

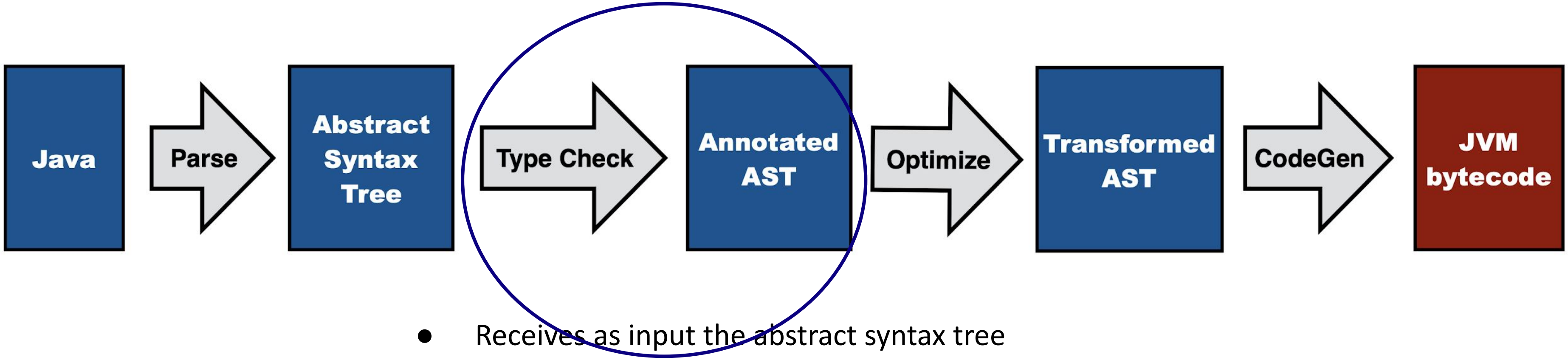
- Skips the comments and whitespaces and produces the stream of tokens for numbers, identifiers, reserved words, etc

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/



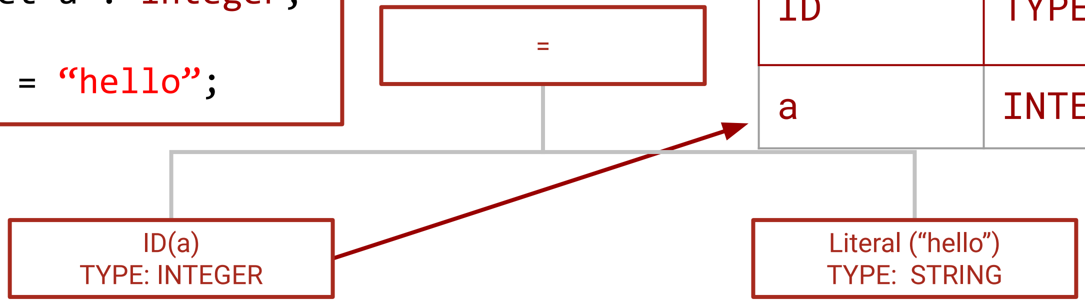


- Receives as input the abstract syntax tree
- Checks that the program complies with the static semantic rules of the language
- Performs name analysis, relating uses of names to declarations of names
- Checks that the types of arguments of operations are consistent with their specification

Input Program

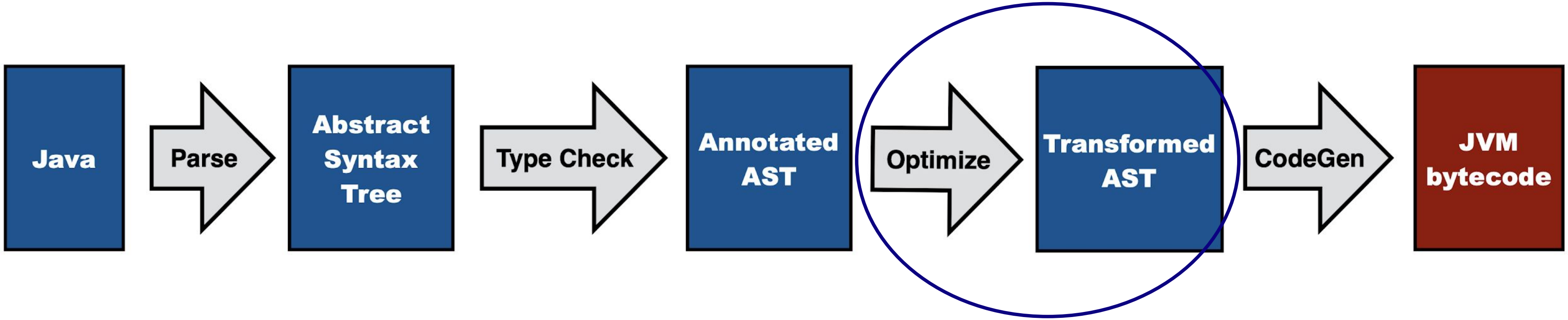
```
let a : integer;

a = "hello";
```

AST



Symbol Table

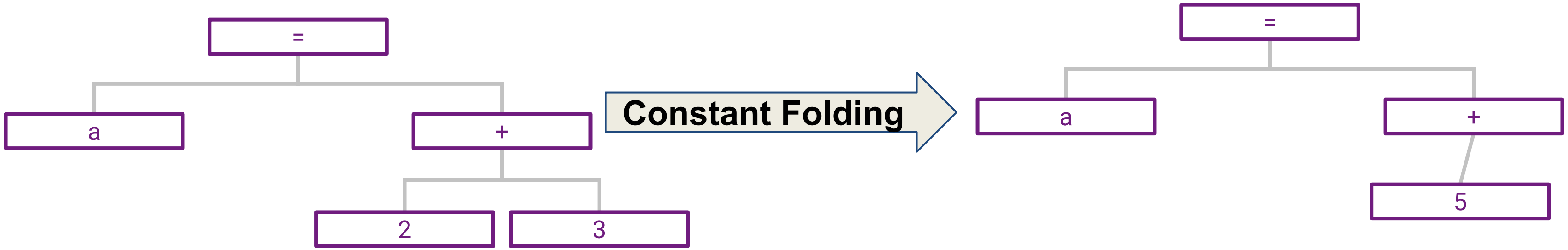
ID	TYPE
a	INTEGER

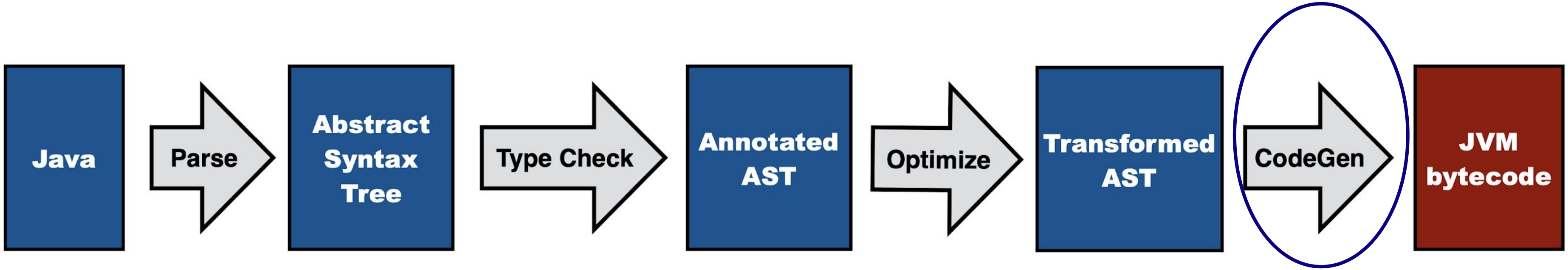


- Applies transformations that improve the program in various goals
- Goals: execution time, memory consumption, energy consumption, etc.
- Examples of transformations: Constant folding, Constant propagation, Loop invariants

Input Program

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

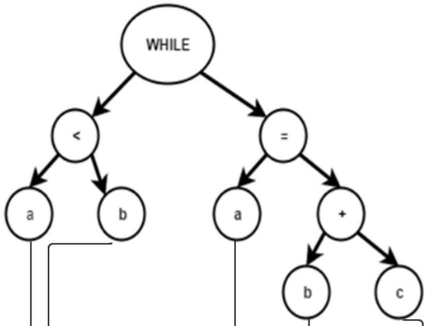




- Transforms abstract syntax tree to instructions for a particular computer architecture

Input Program

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



SYMBOL TABLE

VARIABLE NAME	MEMORY ADDRESS
a	4096
b	4097
c	4098

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:
```