From Brainfuck to Domino Computers
A trip into Esoteric Languages, Turing Machines, Cellular Automata and the Nature of Computation

Nicolas Seriot
October 27th, 2017
1. Esoteric Languages
2. Luring Machines
3. 2D Cellular Automata
4. 1D Cellular Automata
5. Computers Everywhere
Brainfuck syntax

Move the pointer to the right
Move the pointer to the left
Increment the memory cell under the pointer
Decrement the memory cell under the pointer
Output the character signified by the cell at the pointer
Input a character and store it in the cell at the pointer
Jump past the matching ] if the cell under the pointer is 0
Jump back to the matching [ if the cell under the pointer is nonzero
STEP: 6
PROG: +++++[>++++++<-]>.
    ^ 6
DATA: 06 00 00 00 00 00 00 00 00 00 00
     ^ 0

STEP: 7
PROG: +++++[>++++++<-]>.
    ^ 7
DATA: 06 00 00 00 00 00 00 00 00 00 00
     ^ 0

SUMMARY: program stopped after 75 step(s) with output:
HEX: 24
STR: $
What is Whitespace?

Most modern programming languages do not consider white space characters (spaces, tabs and newlines) syntax, ignoring them, as if they weren't there. We consider this to be a gross injustice to these perfectly friendly members of the character set. Should they be ignored, just because they are invisible? Whitespace is a language that seeks to redress the balance. Any non whitespace characters are ignored; only spaces, tabs and newlines are considered syntax.

What are the advantages of Whitespace?

Some things which are difficult in other languages are made much easier in Whitespace. For example, literate programming is simply a matter of writing your helpful comments in between program instructions. It's also easy to encrypt your programs. Simply write a misleading comment!

Whitespace is a particularly useful language for spies. Imagine you have a top secret program that you don't want anyone to see. What do you do? Simply print it out and delete the file, ready to type in at a later date. Nobody will know that your blank piece of paper is actually vital computer code!

What does a typical Whitespace program look like?

Below is an extract from a program which asks for a name then outputs it (see here for the full script.

---

```
Glashideast — Stores > All Popular Polls 4 Deals nit
Devices Build Entertainment Technology OpenSource Science YRO
bh Become a fan of Slashdot on Facebook

New Whitespace-Only Programming Language he source code is written in Haskell, or you can get a Linux binary. You.

Posted by timothy on Tuesday April 01, 2003 @03:35AM from the drag-your-mouse-down-the-examples dept.

he language was designed by two people who shouldn't have had so
foobarbazquux writes either. Thanks also to the residents of #compsoc for their helpful
“Introducing whitespace, a language designed to compensate for the "white-space doesn't count" culture of contemporary programming
languages. Amaze your friends by hiding programs in your web-pages! Astound colleagues by putting a virus in your text file!
(And for those who prefer obfuscation to invisibility, Koshatul writes
“This article in the Sydney Morning Herald, tells of a new programming language which 'makes it impossible to express a security
vulnerability in a program's source code.”
```
https://whitespace.kauaveel.ee/
Piet

- looks like abstract paintings
- start upper left, to the right
- Algorithm to find the next color block
- color changes yield stack commands

Primes tester

http://www.dangermouse.net/esoteric/piet.html

Brainfuck interpreter
Velato

- uses MIDI files as source code
- some freedom so that it doesn't sound totally random
- http://esolangs.org/wiki/velato

https://www.youtube.com/watch?v=Fhni-6Q5z80
PHP, 54 bytes

```php
function a($a){return $a && !strpos(print_r($a,1), '0');}
```

Haskell, 48 bytes

```haskell
f x = or [elem c ''[,,]'' | c: '0': <- scanr (:) [] x] < (x < '[]''[]'')
```

Python, 38 bytes

```python
lambda a: (a > '[]'') * ('O' in a or '0' in a)
```

JavaScript, 24 bytes

```javascript
a => !!a[0] & !/0/.test(a)
```

Ruby, 22 bytes

```ruby
-> a { a[0] && a * ?!! ~ /\b0/ }
```

Perl, 15 bytes

```perl
S = !/0|*..$/
```

OSAB1E, 8 bytes

```osab1e
¢ explicitely made for code golfs
yQ2~08~_
```

Jelly, 3 bytes

```jelly
tPA
```

¢ https://github.com/DennisMitchell/jelly
Ben Olmstead, 1998, designed to be as difficult to program in as possible

Malebolge is the 8th level of hell in Dante's Inferno

based on trits (base 3 digits) (2222222222 base 3 == 59048)
C: Code pointer, D: data pointer, A: accumulator

<table>
<thead>
<tr>
<th>$(C + [C])$</th>
<th>Description</th>
<th>Pseudocode</th>
<th>Op</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Set code pointer to the value pointed to by the current data pointer.</td>
<td>$C = [D]$</td>
<td>i</td>
</tr>
<tr>
<td>5</td>
<td>Output the character in A, modulo 256, to standard output.</td>
<td>PRINT(A%256)</td>
<td>&lt;</td>
</tr>
<tr>
<td>23</td>
<td>Input a character from standard input and store it in A.</td>
<td>A = INPUT</td>
<td>/</td>
</tr>
<tr>
<td>39</td>
<td>Tritwise rotate right.</td>
<td>A = [D] = ROTATE_RIGHT([D])</td>
<td>*</td>
</tr>
<tr>
<td>40</td>
<td>Set data pointer to the value pointed to by the current data pointer.</td>
<td>D = [D]</td>
<td>j</td>
</tr>
<tr>
<td>62</td>
<td>Tritwise &quot;crazy&quot; operation (see table below).</td>
<td>A = [D] = CRAZY_OP(A, [D])</td>
<td>p</td>
</tr>
<tr>
<td>68</td>
<td>No operation.</td>
<td>NOP</td>
<td>o</td>
</tr>
<tr>
<td>81</td>
<td>Stop execution of the current program.</td>
<td>STOP</td>
<td>v</td>
</tr>
</tbody>
</table>

**Crazy Op**

<table>
<thead>
<tr>
<th>A</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

After each instruction, mem location pointed by C is encrypted if in range 33-126

After encryption, C and D are incremented modulo 310 (59049 decimal).

Reference implementation is known to be buggy...
Welcome to Esolang, the esoteric programming languages wiki!

This wiki is dedicated to the fostering and documentation of programming languages designed to be unique, difficult to program in, or just plain weird.

For readers

You'll probably want to find out what an esoteric programming language is in the first place. Then, you might want to explore the complete list of languages, or find something more specific with the categories.

You could also visit the joke language list, which lists languages that can't even be programmed in.

Failing that, you could take a look at a completely random page.

• All of these languages do basically the same thing

• They can solve the very same set of problems

https://esolangs.org
Turing Machine

- Alan Turing, 1936, Great-Britain
- Idealized computing device
- Infinite tape holding 1 symbol per cell, r/w
- Finite number of symbols
- Tape holds input, computation and output
- Indicator to hold machine state
Example: Binary Increment

|     | 0 | 1 | 1 |   |

<table>
<thead>
<tr>
<th>State</th>
<th>Read</th>
<th>Write</th>
<th>Move</th>
<th>New State</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND_END</td>
<td>0</td>
<td>0</td>
<td>R</td>
<td>FIND_END</td>
</tr>
<tr>
<td>FIND_END</td>
<td>1</td>
<td>1</td>
<td>R</td>
<td>FIND_END</td>
</tr>
<tr>
<td>FIND_END</td>
<td>_</td>
<td>_</td>
<td>L</td>
<td>ZERO_TO_LEFT</td>
</tr>
<tr>
<td>ZERO_TO_LEFT</td>
<td>1</td>
<td>0</td>
<td>L</td>
<td>ZERO_TO_LEFT</td>
</tr>
<tr>
<td>ZERO_TO_LEFT</td>
<td>0</td>
<td>1</td>
<td>L</td>
<td>HALT</td>
</tr>
</tbody>
</table>
Turing Machines

Lego

[Video Link]

wood

[Video Link]

electro-mechanical

[Video Link]
A numbered list of instructions:

1. $\text{dec A 3 2}$
2. $\text{inc B 1}$
3. $\text{halt}$

- Increment(register, #instruction)
- Decrement(register, #instruction)
- Halt
Tag System

- 3-tuple \((m, A, P)\)
  - \(m\): number of symbols to delete
  - \(A\): finite alphabet of symbols, including a special halting symbol
  - \(P\): production rules from \(A\) symbols to words in \(A\)

<table>
<thead>
<tr>
<th>Tag system</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m): 2</td>
<td>211</td>
</tr>
<tr>
<td>(A): {1,2,3,H}</td>
<td>1331</td>
</tr>
<tr>
<td>(P):</td>
<td>313321H</td>
</tr>
<tr>
<td>1 --&gt; 3321H</td>
<td>3321H33</td>
</tr>
<tr>
<td>2 --&gt; 331</td>
<td>21H3333</td>
</tr>
<tr>
<td>3 --&gt; 33</td>
<td>H3333331 (halt)</td>
</tr>
</tbody>
</table>
Turing Completeness

- Intuitively: Turing completeness requires:
  - an infinite loop
  - a conditional jump (if-then)
  - data storage

- In practice: prove that some calculation system can simulate a Turing machine, or any machine known to be Turing-complete
Brainfuck Turing Machine

++4+>+4+>>>+[>>, [>+++4+4+<] [->]<<]<[>]>>-[<<t++4+4+>>-[<<---->>-[->]<]<]<[<-<]<[<]+<+[>]<<+><->>>]<<[<[<]>[<-[>+++4+4++<->]>[<+>-]+<<<+++>+>

[<<+>->-][<<[-]>>-][<<++>+>-][<<-->->>+++<->][<<+>+>>--<-][<<->->-][<<++++>+>>+-]<[-<][<<->>>4+++<->[<<->>>--<->]<[<<++++>+>>+-][<<[-]>->>++<->][<<+++++>+>>--<->]<->>++<[<<->>][<<->>][<<+++>>>-<->][<<---->>>++<->][<<++>>>+<-][->]<-[<<->]
Accidentally Turing-Complete

Wang Tiles

mov is Turing-complete

Stephen Dolan
Computer Laboratory, University of Cambridge
stephen.dolan@cl.cam.ac.uk


CSS

Minecraft
Is Unicode Turing-Complete?

- Can we implement a tag system with bidi algorithms? [http://www.unicode.org/reports/tr9/]
- Can we implement logical gates with case folding rules?
Halting Problem

Turing machines cannot do anything! Eg. will a given prog. ever halt? Proof:

```python
func halts(f) -> Bool {
    // ...
}
```

```python
func g() {
    if halts(g) {
        while true {}
    }
}
```

- `halts(f)`
  - if true, `g()` runs forever
  - if false, `g()` stops
  - ⚡ contradiction!
- `halts()` cannot exist
<table>
<thead>
<tr>
<th>1. Esoteric Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Luring Machines</td>
</tr>
<tr>
<td>3. 2D Cellular Automata</td>
</tr>
<tr>
<td>4. 1D Cellular Automata</td>
</tr>
<tr>
<td>5. Computers Everywhere</td>
</tr>
</tbody>
</table>
2D Cellular Automata

• 1970: Conway developed Game of Life, a simple 2D grid state model

• 2 or 3 neighbours -> survive

• 0, 1 or 4+ neighbours -> death

• 3 neighbours -> birth
Oscillators

- Blinker (2)
- Star (3)
- Cross (3)
- French kiss (3)
- Clock (4)
- Pinwheel (4)

- Octagon (5)
- Fumarole (5)
- Pentoad (5)
- Kok’s galaxy (8)
- Pentadecathlon (15)
• Conway conjecture: no pattern can grow without limit

• Gliders can **move information** and interact

• We can develop **logic gates** to **simulate digital circuits** and solve any computational problem
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.
Tetris over Metapixel over GoL
Half Adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Sum (XOR)</th>
<th>Carry (AND)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Langton's Ant

- At white square turn 90° right
  flip color
  step forward

- At black square turn 90° left
  flip color
  step forward

- Gajardo et al (2000) build logical circuits
  -> TM -> capable of universal computation
Wireworld

- Brian Silverman, late 80s
- CA that simulates digital circuits easily

<table>
<thead>
<tr>
<th>State</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty</td>
<td>empty</td>
</tr>
<tr>
<td>head</td>
<td>tail</td>
</tr>
<tr>
<td>tail</td>
<td>wire</td>
</tr>
<tr>
<td>wire</td>
<td>if 1 or 2 heads among the 8 neighbours then: head else: wire</td>
</tr>
</tbody>
</table>
http://karlscherer.com/Wireworld.html

https://github.com/nst/Wireworld
MEM

CELL
www.quinapalus.com

demo in Golly

https://www.logre.eu/wiki/Projet_Wireworld/en
1. Esoteric Languages
2. luring Machines
3. 2D Cellular Automata
4. 1D Cellular Automata
5. Computers Everywhere
1D Cellular Automata

0 1 0 0 0 0 1 0 0 0 0 1 0 1 0 0
A simple and interactive tool to explore Elementary Cellular Automata, in Swift

https://github.com/nst/ECAExplorer
Rule 99
Rule 110
def next_row(row, rule_bits):
    r = row[-1] + row + row[-1]
    return [rule_bits[i] + r[i] + r[i-1] for i, b in enumerate(row)]

def save_png_data(rows, filename):
    ll = [[255 - 'b * 255 for b in row for row in rows]]
    png.fromarray(ll, 'L').save(filename)
    print filename

def run():
    rule = 110 # or 0 or b
    rule_bits = [int(x) for x in '1: 68b'.format(rule)]
    (WIDTH, HEIGHT) = (1088, 18GB)
    row = [0] * WIDTH
    row[500] = 1
    png_data = [row]
    for i in range(HEIGHT-1):
        row = next_row(row, rule_bits)
        png_data.append(row)
    save_png_data(png_data, "rule_50_50.png" % (rule, WIDTH, HEIGHT))

if __name__ == '__main__':
    run()

https://gist.github.com/nst/7f123d951355041044b7fiae9d33de3e
• Rule 110 can be made to emulate a cyclic tag system

• Wolfram ANKS, p. 662

Universality in Elementary Cellular Automata

Matthew Cook
Department of Computation and Neural Systems,
Caltech, Mail Stop 136-93,
Pasadena, California 91125, USA

The purpose of this paper is to prove a conjecture made by Stephen Wolfram in 1985, that an elementary one dimensional cellular automaton known as "Rule 110" is capable of universal computation. I developed this proof of his conjecture while assisting Stephen Wolfram on research for A New Kind of Science [1].

2004

When central bit is 1, acts as a NAND gate.

When central bit is 0, can move information horizontally.

Rule 110
- Emulate logical gates
- Wolfram ANKS, p. 662
• Emulate random access memory

• Wolfram ANKS, p. 663
• A 16 states automaton
• Computes primes
• Wolfram ANKS, p. 640
Fluid Simulation

https://gist.github.com/nst/90e2cf3d18e1ab56bfc3a52ba1a27e90
1. Esoteric Languages

2. 2D Cellular Automata

3. 1D Cellular Automata

4. Computers Everywhere
Mechanical Binary Adder

https://www.youtube.com/watch?v=GcDshWmhF4A
Minesweeper

\[ z = x \text{ OR } y \]

\[ \text{NOT} (x \text{ OR NOT } y) \]

http://www.formauri.es/personal/pgimeno/compurec/Minesweeper.php
Domino Logic

Domino OR
The OR gate is fairly easy to make from dominoes - we need two input chains, either of which will set off the output chain of dominoes. A domino OR gate looks like this:

Try making one from dominoes!

Domino XOR
The XOR gate can be very elegantly made from dominoes: we need either of which will set off the output chain of dominoes, but not both. This can be achieved by making the two inputs pass along the same section of the chain, if they're both running they will stop each other. This can be achieved like this:

Try and build this yourself!

Domino AND
The AND gate is more difficult to make from dominoes - we need the output chain to fall only when both input chains fail. One way to do this is to split an input chain so it can knock out itself, and use the second input to knock out the first input's knock-out chain. If both inputs were falling, the knocker out gets knocked out!

Can you see any problems which might occur with this gate? Why do parts of the chain need to be longer?

Try and build this yourself!

Domino Logic

https://www.youtube.com/watch?v=SudixyugiX4
Paper / Pneumatic Binary Adder

https://www.youtube.com/watch?v=yvANcR4mQ7M
Turing Tumble

https://www.kickstarter.com/projects/871405126/turing-tumble-gaming-on-a-mechanical-computer
404 Nature of computation not found

Computation is nowhere and everywhere

Is the Universe computation? 🙄🤔
The Simulation Hypothesis

- Future generations will certainly use super-super-computers to run many detailed simulations.

- If the simulations create consciousness, then the majority of minds belong to simulated universes rather than the original universe.

- So, we're more likely to be part of a simulated universe than the original one. Old skeptical hypothesis that reality is an illusion
Is brain a computer?
Are thoughts computation?
What is consciousness?
Can we be alive if we can avoid death?
Should life be only biological?