Survey of Imperative Style Turing Complete proof techniques and an application to prove Proteus Turing Complete

By: Isaiah Martinez

AUTONOMY RESEARCH CENTER FOR STEAHM

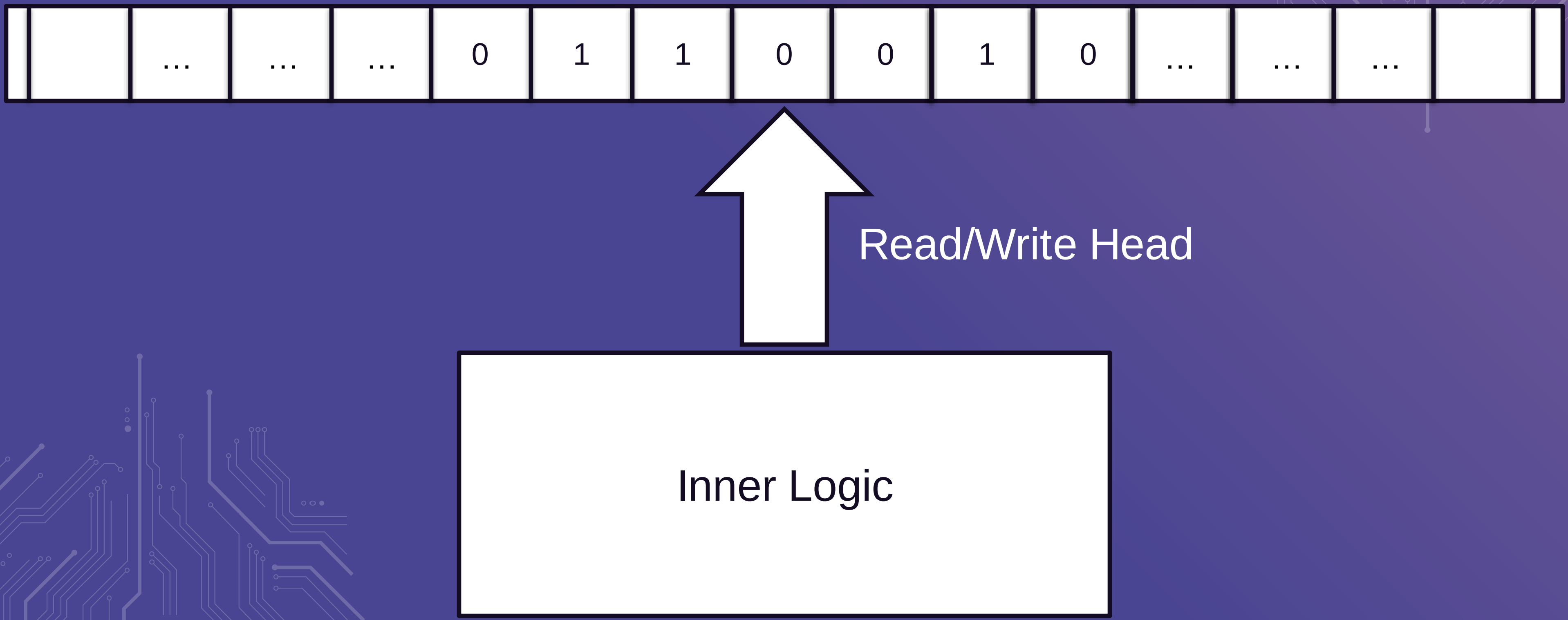
Table of contents

01

Introduction

- Turing Machines (TM)
- The Church-Turing Thesis (CTT)
- Rice’s Theorem
- Turing Completeness

The Turing Machine



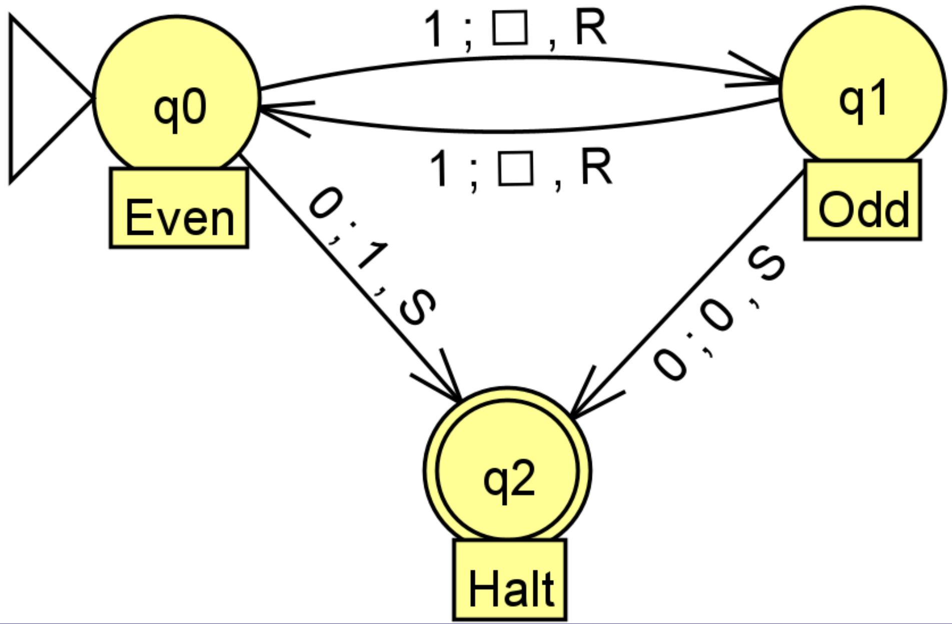
Example of a TM

Given any amount of 1’s followed by a 0.

Goal is to find the parity of the amount of 1’s.

1110 => 0 (odd)

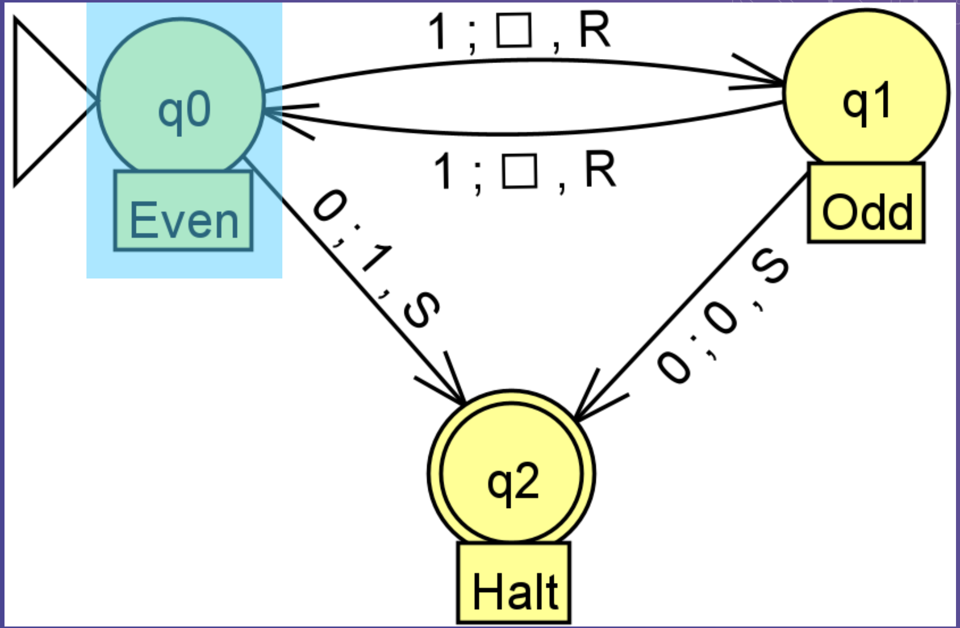
110 => 1 (even)



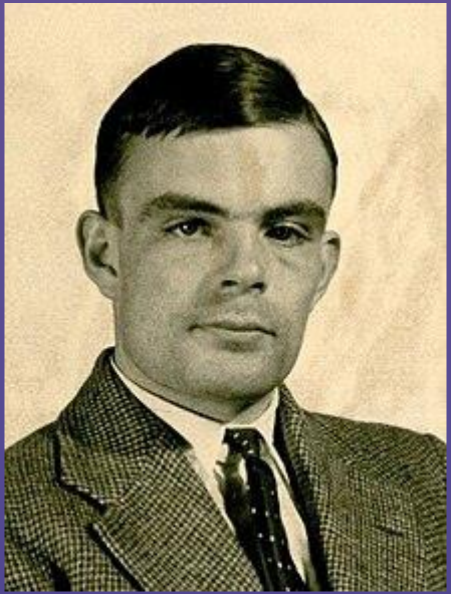
Walkthrough of Sample TM

110

Even!



The Church–Turing Thesis



Every effectively calculable function can be computed by a Turing Machine

Rice's Theorem

Let φ be an admissible numbering of partial computable functions. Let P $\subseteq \mathbb{Z}$.

Suppose that:

1. P is non-trivial: P is neither empty nor $\mathbb{N}$ itself.
2. P is extensional: $\forall$ m,n $\in \mathbb{N}$, if $\varphi_m = \varphi_n$ then m $\in$ P $\Leftrightarrow$ n $\in$ P.

Then P is undecidable.

The only decidable index sets are $\emptyset$ and $\mathbb{N}$.

Rice's Theorem applied to Programming Languages

Programs are undecidable.

Only able to determine simplistic things in nature but does not have an answer to complex problems like The Halting Problem.

Turing Completeness

For a given system to be Turing Complete, it must be capable of performing any computation that a TM can perform.

02

Proteus

- Actor Model
- Use Cases

Actor Model

Actors: Individuals within the overall group

States: Status of the individual

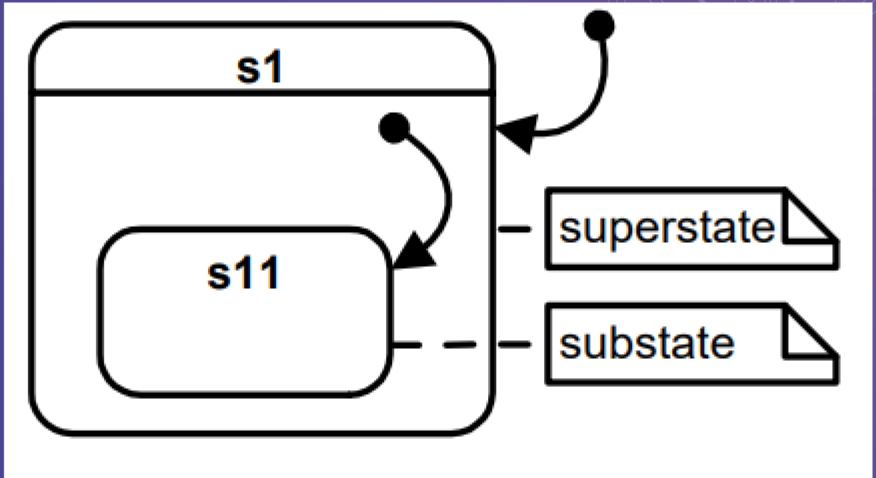
Events: Messages sent between individuals

Different teams in a company

Proteus Language Design Details

$$3/2 = 1$$

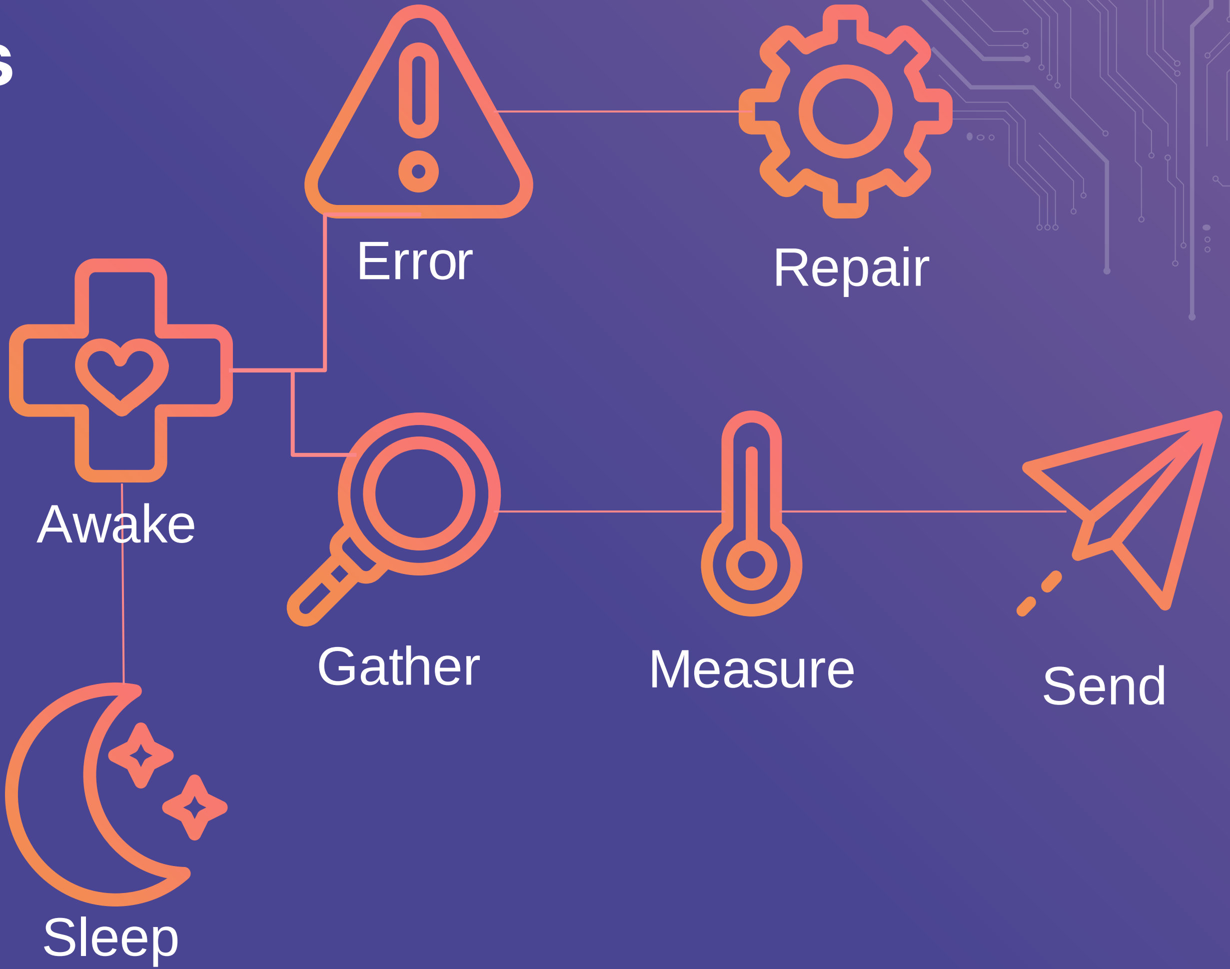
Truncation to remain closure in $\mathbb{Z}$



States can be HSMs

Cannot create/destroy State Machines dynamically

Use Cases



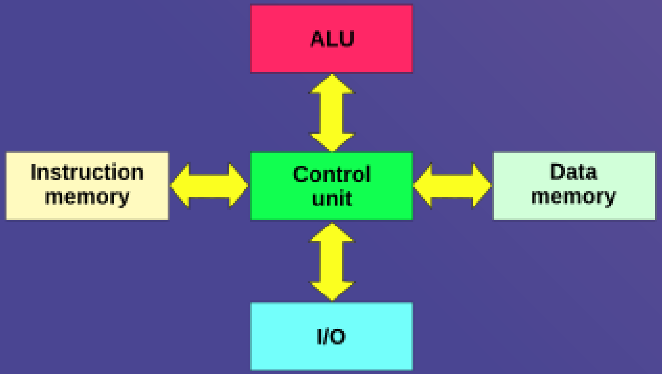
03 Turing Complete Proofs

- Computer Engineering
- Computer Science – Automata Theory
- Software Engineering
- Mathematics

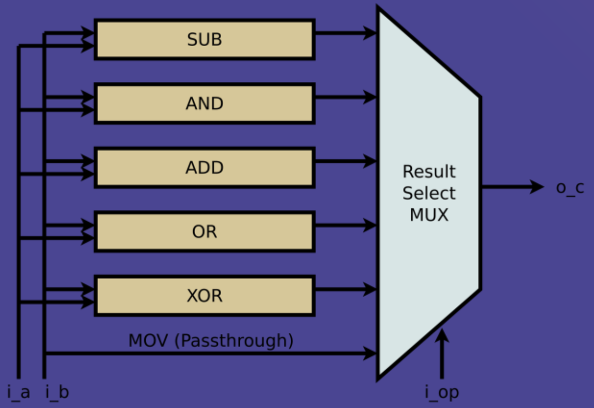
Computer Engineering

Logic Gates:
- AND
- OR
- XOR

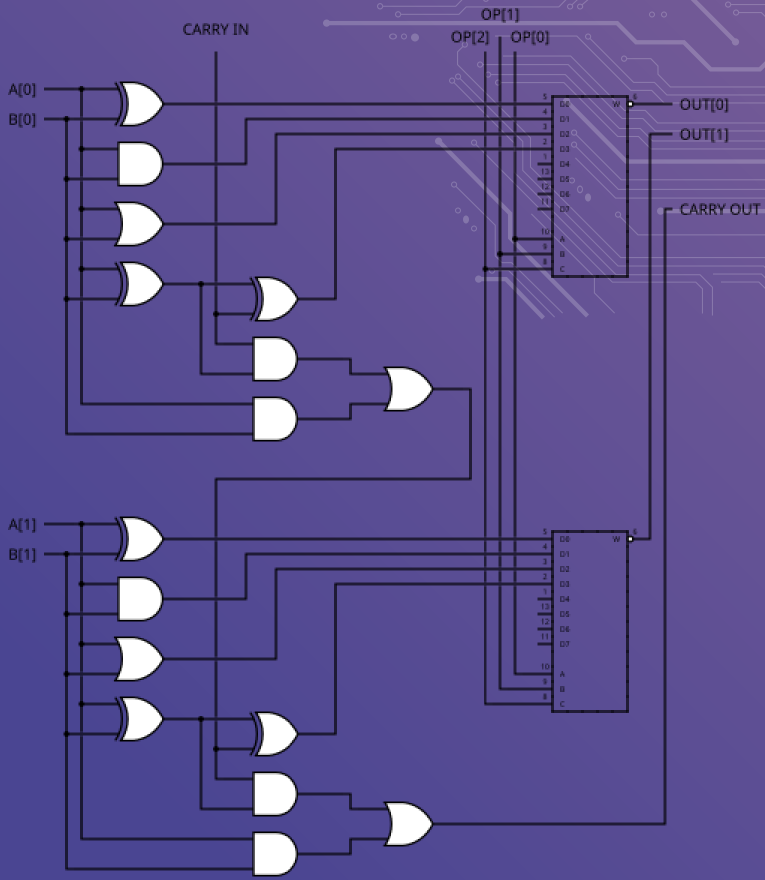
Circuitry:
- ADD
- SUB
- MUX



Harvard Architecture



Simplified 2-Bit ALU



2-Bit ALU Schematic

Computer Science – Automata Theory

The theoretical definition of a TM, reliant on an undecidable input

Used to represent TMs in a more concrete manner using mathematical notation

Many equivalent definitions

Definition 1 *A Turing Machine M is defined by:*

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

where:

Q *is the set of internal states,*

Σ *is the input alphabet,*

Γ *is the finite set of symbols called the tape alphabet,*

δ *is the transition function,*

$\square \in \Gamma$ *is a special symbol called the blank,*

$q_0 \in Q$ *is the initial state,*

$F \subseteq Q$ *is the set of final states.*

The transition function δ is defined as

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}.$$

Example of a TM – Revisited

$Q = \{q_0, q_1, q_2\}$ with associated labels {Even, Odd, Halt}

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

$\Gamma = \{0, 1, \square\}$

$F = \{q_2\}$

$q_0 \in$ Q as the initial state

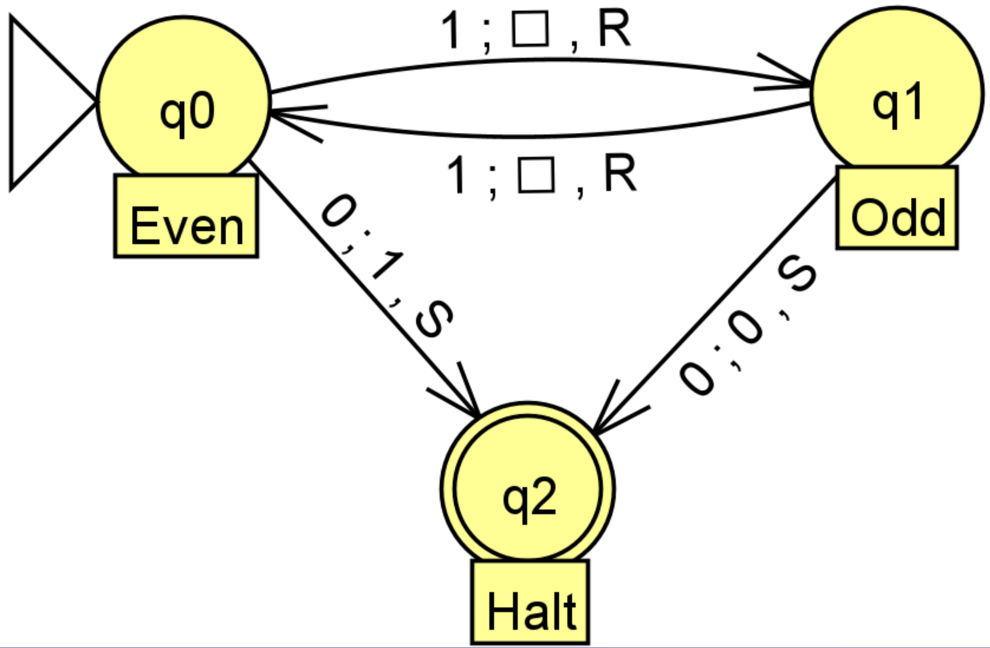
and the following delta transitions

$$\delta(q_0, 0) = (q_2, 1, S),$$

$$\delta(q_0, 1) = (q_1, \square, R),$$

$$\delta(q_1, 0) = (q_2, 0, S),$$

$$\delta(q_1, 1) = (q_0, \square, R).$$



Sample TM

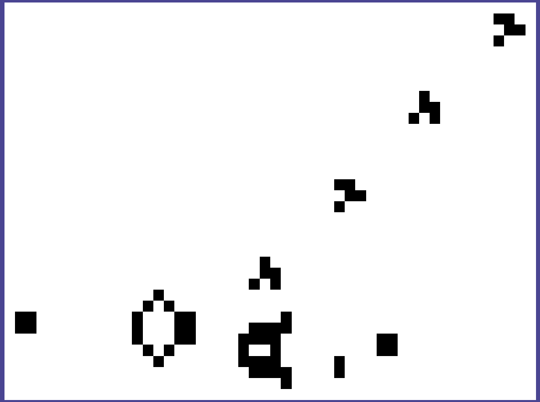
Software Engineering

Implement a known TC system:

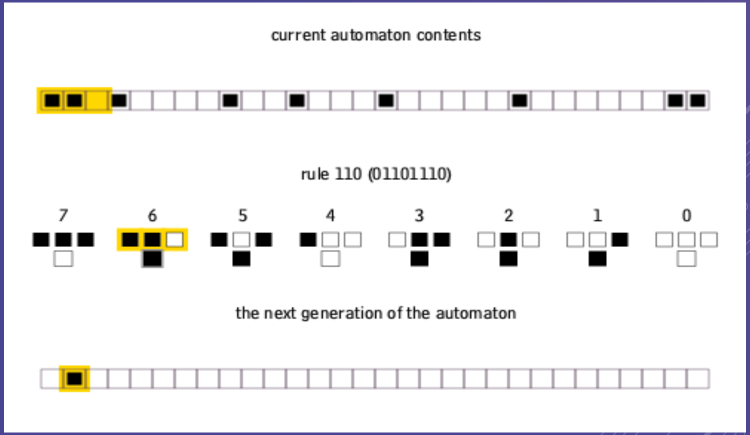
- Programmable Calculators
- Programming languages
- Cellular Automata

>	Increments the data pointer by one. (This points to the next cell on the right).
<	Decrement the data pointer by one. (This points to the next cell on the left).
+	Increments the byte at the data pointer by one.
−	Decrements the byte at the data pointer by one.
.	Output the byte at the data pointer.
,	Accept one byte of input, storing its value in the byte at the data pointer.
[	If the byte at the data pointer is zero, then instead of moving the instruction forward to the next command, go to the matching ']' command. (Jump forwards).
]	If the byte at the data pointer is non-zero, then instead of moving the instruction forward to the next command, go to the matching '[' command. (Jump backwards).

Brainfuck Instruction Set

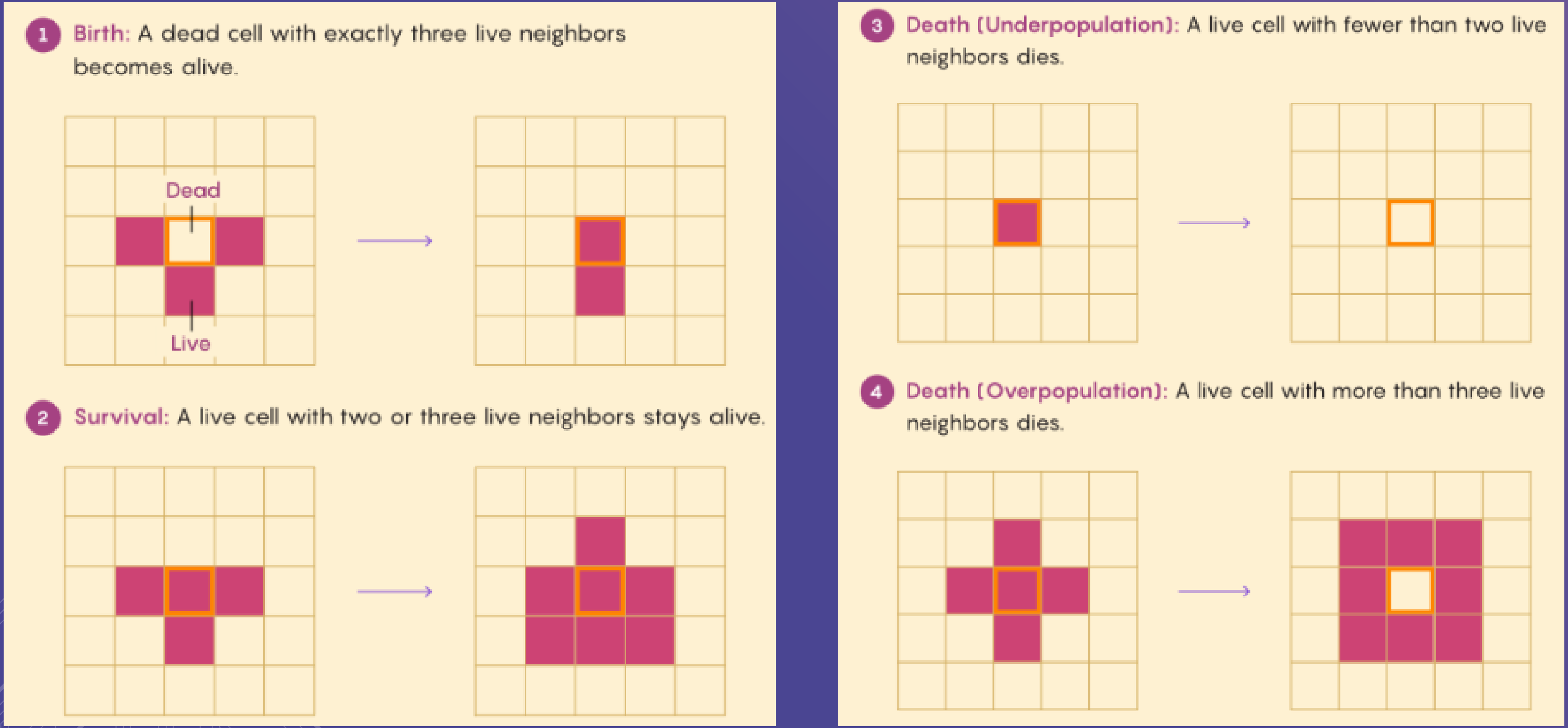


Conway's Game of Life



Rule 110

Software Engineering – Conway’s Game of Life



Rules for Conway’s Game of Life

Mathematics

Lambda Calculus:

1. x : A variable as a function parameter

2. λx.M : A lambda abstraction that is function definition with bound variable x as input and returns the body M

3. (M N) An application where it applies the function M to argument N

Lambda Calculus in Practice

$$0 \equiv \lambda sz.s(z)$$

$$1 \equiv \lambda sz.s(s(z))$$

$$2 \equiv \lambda sz.s(s(s(z)))$$

Using SUCCession to create ℕ

$$PLUS := \lambda mnfx.nf(mfx)$$

$$\equiv \lambda mn.n\ SUCC\ m$$

$$SUB := \lambda mn.n\ PRED\ m$$

Addition and Subtraction

$$PAIR := \lambda xyf.fxy$$

$$CAR := \lambda p.p\ TRUE$$

$$CDR := \lambda p.p\ FALSE$$

$$NIL := \lambda x.\ TRUE$$

List Operations

$$TRUE := \lambda xy.x \equiv K$$

$$FALSE := \lambda xy.y \equiv 0$$

Booleans

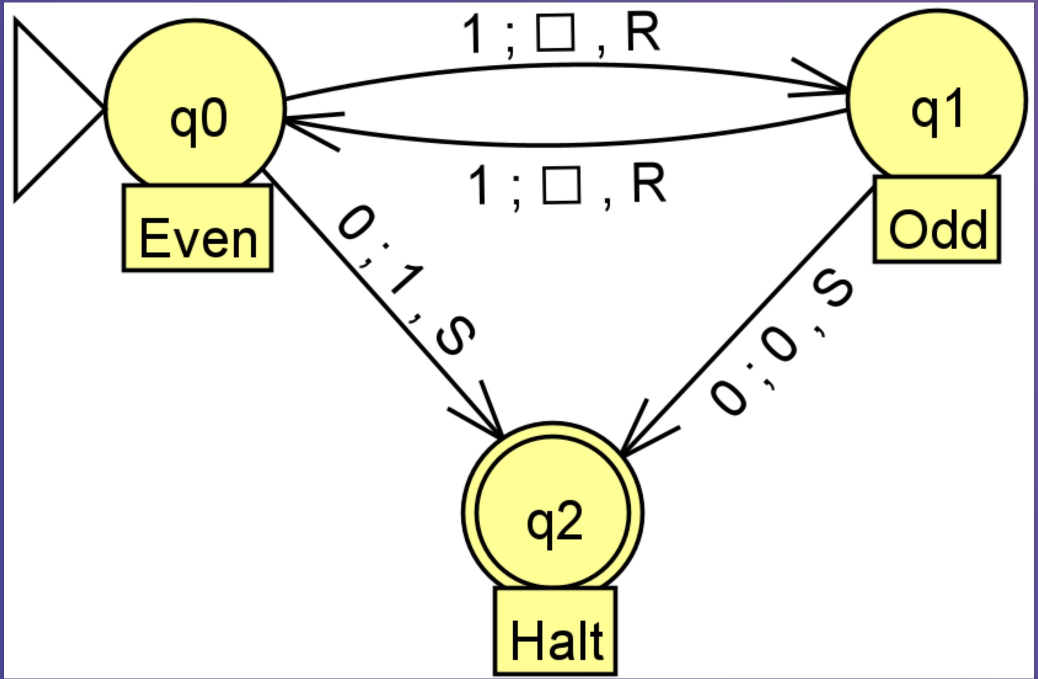
Example of a TM – Revisited

$$MOD\ (SUB\ (LENGTH\ (\mathbf{input})\ 1))\ 2$$

Simplified Lambda Calculus

Input: 11…10

Output: {0,1}



Sample TM

Lambda Calculus Expanded

MOD (**INPUT** 2)

$$(\lambda m.(\lambda g.(\lambda x.g(\lambda a.x(x)(a)))(\lambda x.g(\lambda a.x(x)(a))))(\lambda f.(\lambda n.(\lambda a.(\lambda b.(\lambda p.(\lambda a.(\lambda b.p(b)(a))))$$
$$((\lambda m.(\lambda n.(\lambda n.n(\lambda x.(\lambda a.(\lambda b.b)))(\lambda a.(\lambda b.a)))((\lambda m.(\lambda n.n((\lambda n.(\lambda f.(\lambda x.n(\lambda g.(\lambda h.h(g(f)))$$
$$(\lambda u.x)(\lambda u.u)))))(m)))(m)(n))))(b)(a)))(n)(\lambda f.(\lambda x.f(f(x))))(\lambda j.(\lambda n.n(\lambda x.(\lambda a.(\lambda b.b)))$$
$$(\lambda a.(\lambda b.a))(n))(\lambda j.f(j))((\lambda m.(\lambda n.n((\lambda n.(\lambda f.(\lambda x.n(\lambda g.(\lambda h.h(g(f))))(\lambda u.x)(\lambda u.u)))))(m)))$$

Expansion of a simpler lambda calculus expression

04 Proteus Proofs

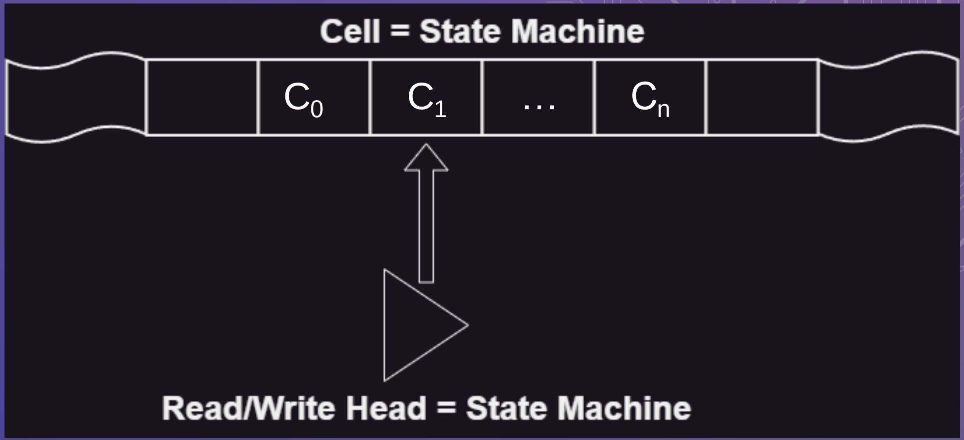
- Formal definition using Automata Theory
- Conway's Game of Life
- Rule 110

Automata Theory

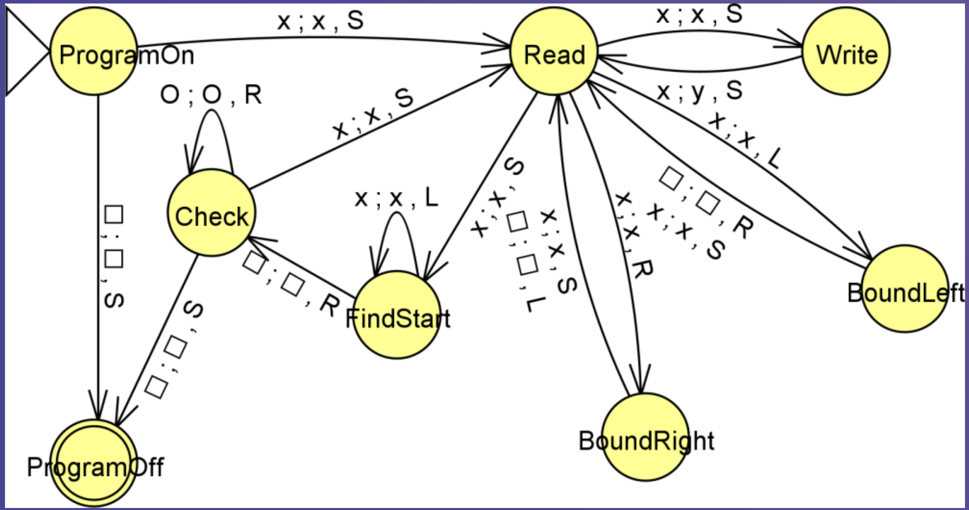
Using Rice's Theorem, Proteus programs are undecidable

Must define a TM using Proteus

Note: all cells are state machines

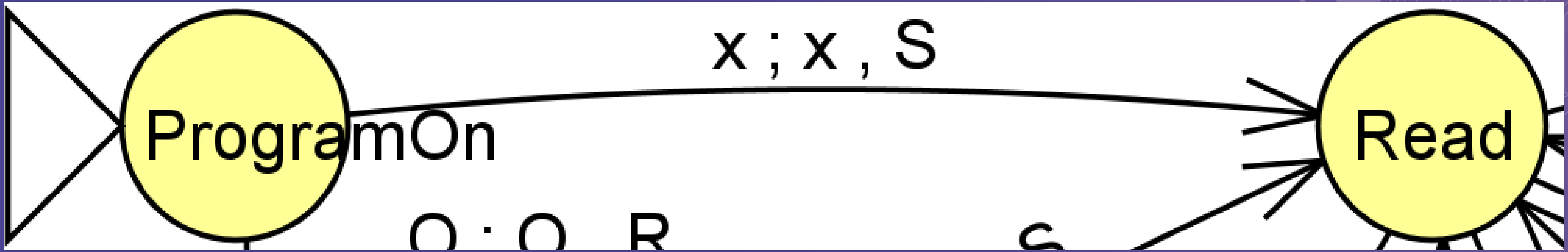


Theoretical Design of TM



TM State Diagram

TM made in Proteus – Program Start



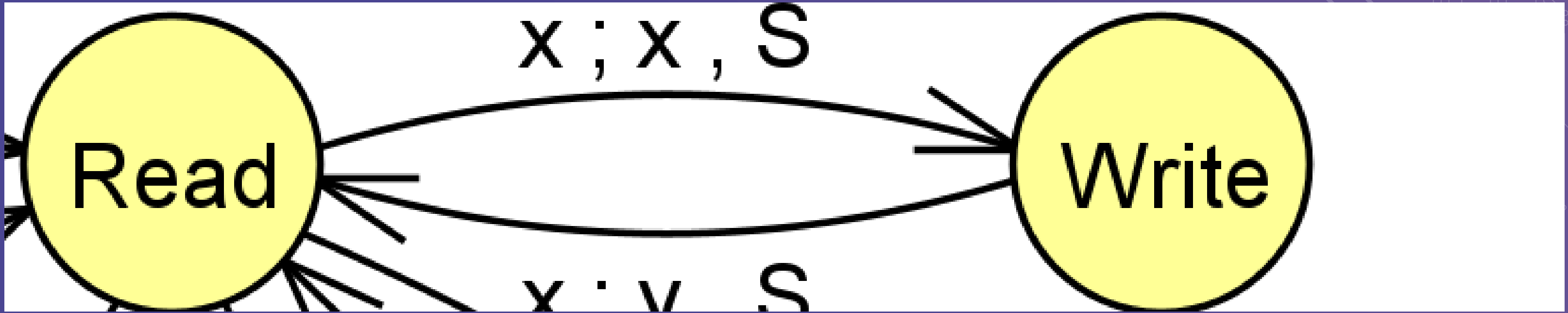
Program Start in the partial TM State Diagram

Program has a predetermined list of cells that are in some state (On,Off, s_i)

Read/Write Head (RWH) is initialized to the ProgramOn state indicating the program has just begun.

The RWH transitions to Read state for processing further operations from the program.

TM made in Proteus – Writing



Writing logic in the partial TM State Diagram

From the Read state, the program determines what to do

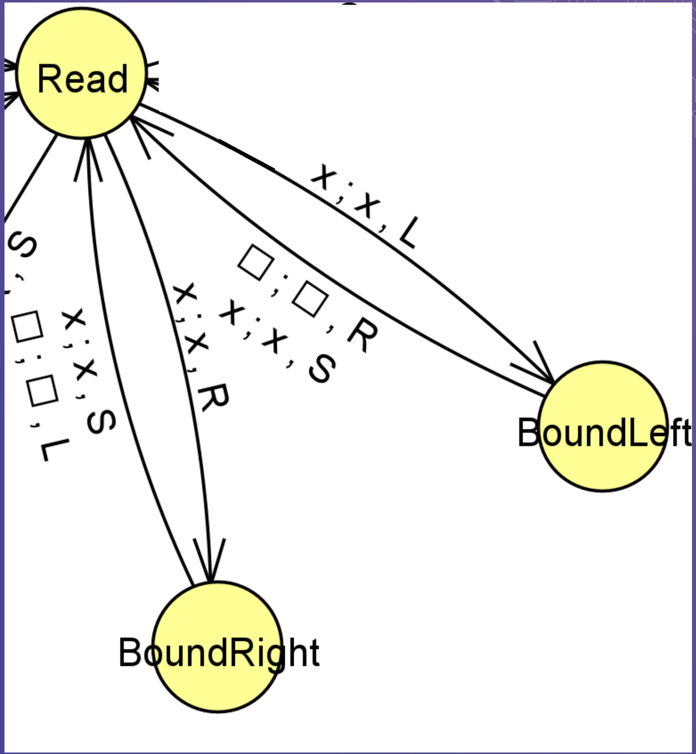
Suppose it is to write a new value (state) to the current cell (state machine). It enters the Write state and stays at the current cell. Then it updates the data at the current cell and stays at the current cell. The RWH then goes back to the Read state.

TM made in Proteus – Moving

If the RWH wants to move in a direction, it enters the BoundLeft/BoundRight state.

WLOG let the RWH want to move Left. If the RWH moves onto a blank (past c_0) then it moves one cell in the <u>opposite direction</u> (right) to remain within the confines of the predetermined tape (some c_0).

If the move remains on some <u>c_i</u> then the move is <u>valid</u>.



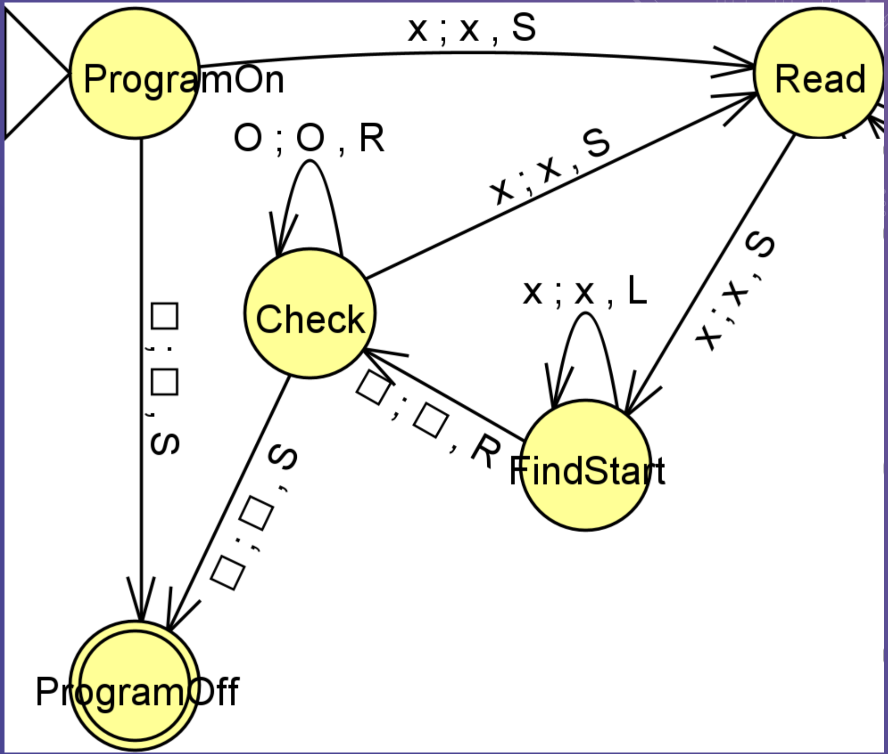
Moving logic in the partial TM State Diagram

TM made in Proteus – Halting

If the program wants to halt, it must ensure that all defined cells on the tape are in the “Off” state (i.e. $\forall c_i$ c_i = “Off”)

It begins by finding the first cell c_0 and checks each cell moving to the right one at a time.

If it encounters some c_i not in the “Off” state, then it enters the Read state.



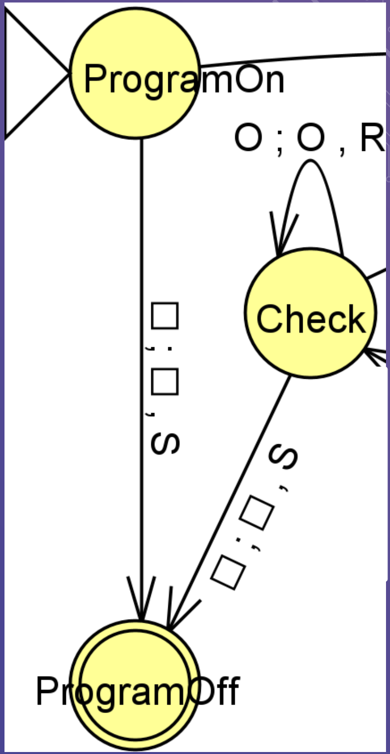
Halting logic in the partial TM State Diagram

TM made in Proteus – Halting

If instead it encounters a blank, then it has moved past the final predefined cell (c_n). This means that all cells are in the "Off" state (i.e. c_0, c_1, …, c_n are in the "Off" state).

In such a case, the program stays at the current cell, enters the "ProgramOff" state and halts.

If an empty program is given, then the RWH immediately moves to the "ProgamOff" state.



Halting logic in the partial TM State Diagram

Automata Theory – Formal Definition

$$Q = \{\text{'ProgramOn'}, \text{'ProgramOff'}, \text{'Read'}, \text{'Write'},$$
$$\text{'BoundLeft'}, \text{'BoundRight'}, \text{'FindStart'}, \text{'Check'}\}$$
$$F = \{\text{'ProgramOff'}\}$$
$$q_0 = \text{'ProgramOn'}$$
$$\Sigma = \{\text{'On'}, \text{'Off'}\}$$
$$\Gamma = \{\text{'On'}, \text{'Off'}, s_0, \ldots, s_n, \square\} \text{ for } n \in \mathbb{Z}_{\geq 0}$$

Formal Automata Definition for a TM

let $x, y \in \Gamma$

$$\delta(\text{'ProgramOn'}, x) = (\text{'Read'}, x, S)$$
$$\delta(\text{'ProgramOn'}, \square) = (\text{'ProgramOff'}, \square, S)$$
$$\delta(\text{'Read'}, x) = (\text{'Write'}, x, S)$$
$$\delta(\text{'Read'}, x) = (\text{'BoundLeft'}, x, L)$$
$$\delta(\text{'Read'}, x) = (\text{'BoundRight'}, x, R)$$
$$\delta(\text{'FindStart'}, x) = (\text{'BoundLeft'}, x, S)$$
$$\delta(\text{'Write'}, x) = (\text{'BoundLeft'}, y, S)$$
$$\delta(\text{'BoundLeft'}, x) = (\text{'Read'}, x, S)$$
$$\delta(\text{'BoundLeft'}, \square) = (\text{'Read'}, \square, R)$$
$$\delta(\text{'BoundRight'}, x) = (\text{'Read'}, x, S)$$
$$\delta(\text{'BoundRight'}, \square) = (\text{'Read'}, \square, L)$$
$$\delta(\text{'FindStart'}, x) = (\text{'FindStart'}, x, L)$$
$$\delta(\text{'FindStart'}, \square) = (\text{'Check'}, \square, R)$$
$$\delta(\text{'Check'}, x) = (\text{'Read'}, x, S)$$
$$\delta(\text{'Check'}, \text{'Off'}) = (\text{'Check'}, \text{'Off'}, R)$$
$$\delta(\text{'Check'}, \square) = (\text{'ProgramOff'}, \square, S)$$

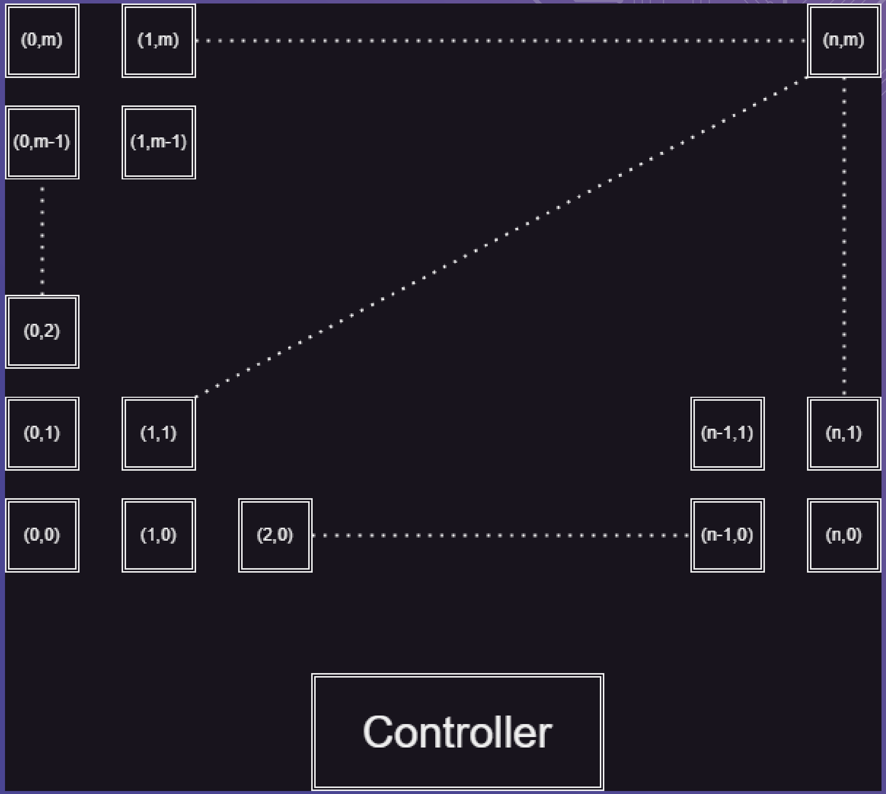
Conway's Game of Life

Create the cartesian plane (2D) of cells using state machines

Additional non-planar HSM called the Controller for sending messages

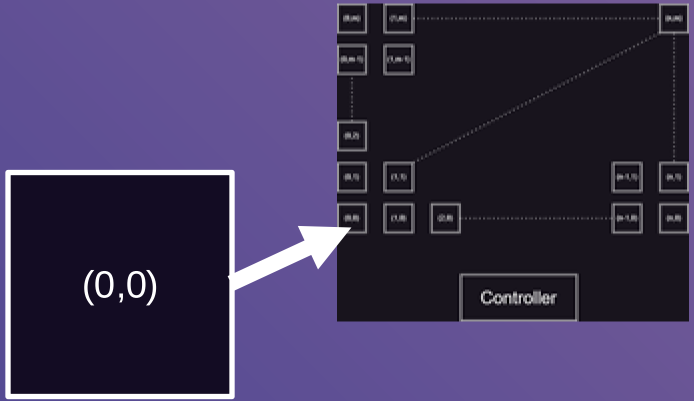
Messages:

1. getDisplay(myName)
2. calculateNextState()
3. updateAllCells()
4. initializeCellValue(Value)



System Design

Conway’s Game of Life – Cells



Each cell holds the following data:

1. Local Variables:
    1. myName :: String
    2. Xcoord :: Int
    3. Ycoord :: Int
    4. currCellIsOn :: Boolean
    5. nextStateCurrCellIsOn :: Boolean
2. States:
    1. Display
    2. CalculateNext
    3. Update

Local Variables:

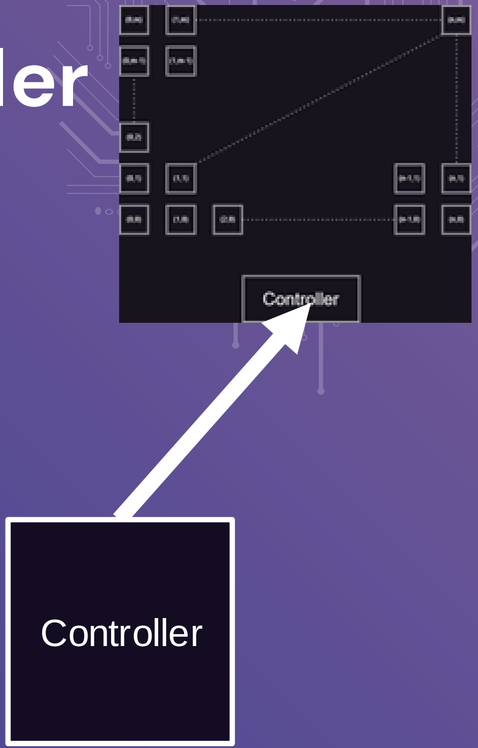
- cell00
- 0
- 0
- false
- false

Initialized to Display state

Conway’s Game of Life – Controller

The controller holds the following data:

1. States:
   1. Setup
      1. Sends initializeCell with initial state
   2. nextStage
      1. Broadcasts to all cells to calculate their next state
      2. Once all cells have updated values, broadcasts to all cells to update to the newly calculated state



Initialized to Setup state

Conway's Game of Life written in Proteus

```
actor cellXY{
    string myName = "cellXY";
    int Xcoord = [X];
    int Ycoord = [Y];
    bool currCellIsOn = false;
    bool nextStateCurrCellIsOn = false;
    statemachine {
        initial Display;
        state Display {
            on getDisplay {otherCellName} {otherCellName ! currCellIsOn}
            on calculateNextState {} {go calculateNext {}}
            on updateAllCells {} {go Update {}}
            on initializeCell {Value} {currCellIsOn = Value}
        }
        state calculateNext {
            int neighborTop = [Y]coord + 1;
            int neighborBot = [Y]coord - 1;
            int neighborLeft = [X]coord - 1;
            int neighborRight = [X]coord + 1;
            string neighborTopName = "cell" + Xcoord + neighborTop;
            string neighborBotName = "cell" + Xcoord + neighborBot;
            string neighborLeftName = "cell" + neighborLeft + Ycoord;
            string neighborRightName = "cell" + neighborRight + Ycoord;
            int count = 0;
            if (neighborTopName ! getDisplay {myName}) {
                count += 1;
            }
            if (neighborBotName ! getDisplay {myName}) {
                count += 1;
            }
            if (neighborLeftName ! getDisplay {myName}) {
                count += 1;
            }
            if (neighborRightName ! getDisplay {myName}) {
                count += 1;
            }
            if ((!(currCellIsOn)) && (count == 3)) {
                nextStateCurrCellIsOn = true;
            } else if ((currCellIsOn) && ((count == 2) || (count == 3))) {
                nextStateCurrCellIsOn = true;
            } else if ((currCellIsOn) && (count < 2)) {
                nextStateCurrCellIsOn = false;
            } else if ((currCellIsOn) && (count > 3)) {
                nextStateCurrCellIsOn = false;
            }
            go Display {}
        }
        state Update {
            currCellIsOn = nextStateCurrCellIsOn;
            go Display {}
        }
    }
}
```

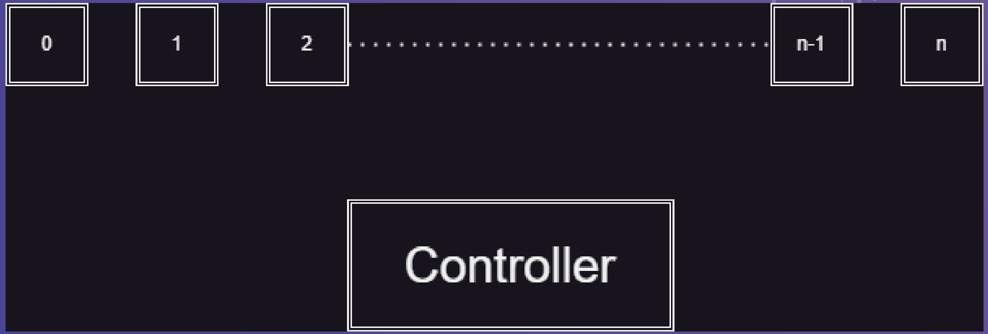
Rule 110

Create the 1D grid of cells using state machines

Additional non-planar HSM called the Controller for sending messages

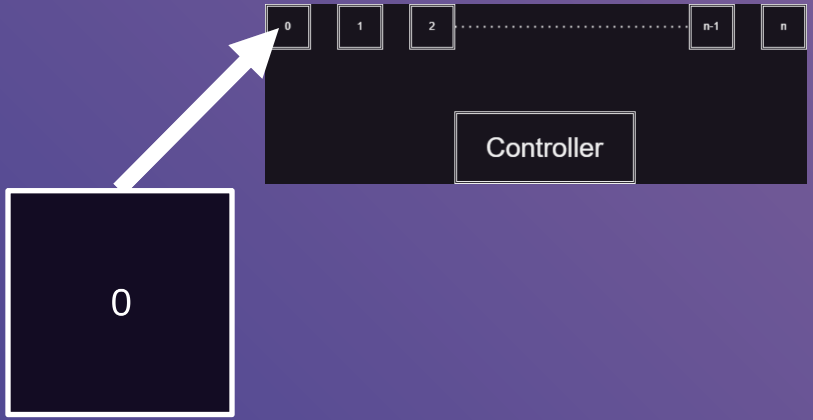
Messages:

1. getDisplay(myName)
2. calculateNextState()
3. updateAllCells()
4. initializeCellValue(Value)



System Design

Rule 110 – Cells



Each cell holds the following data:

1. Local Variables:
    2. myName :: String
    3. coord :: Int
    1. currCellIsOn :: Boolean
    2. nextStateCurrCellIsOn :: Boolean
4. States:
    1. Display
    2. CalculateNext
    3. Update

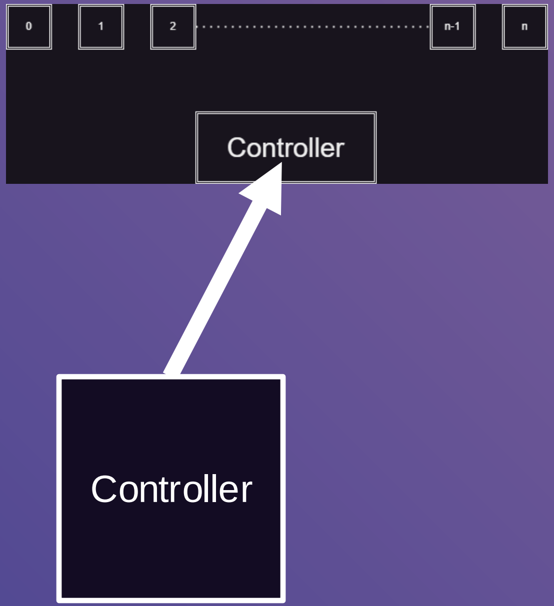
Local Variables:
- cell00
- 0
- false
- false

Initialized to Display state

Rule 110 – Controller

The controller holds the following data:

1. States:
    1. Setup
        1. Sends initializeCell with initial state
    2. nextStage
        1. Broadcasts to all cells to calculate their next state
        2. Once all cells have updated values, broadcasts to all cells to update to the newly calculated state



Initialized to Setup state

Rule 110 written in Proteus

```
actor cellXY{
    string myName = "cellX";
    int coord = [X];
    bool currCellIsOn = false;
    bool nextStateCurrCellIsOn = false;
    statemachine {
        initial Display;
        state Display {
            on getDisplay {otherCellName} {otherCellName ! currCellIsOn}
            on calculateNextState {} {go calculateNext {}}
            on updateAllCells {} {go Update {}}
            on initializeCell {Value} {currCellIsOn = Value}
        }
        state calculateNext {
            int neighborLeft = coord - 1;
            int neighborRight = coord + 1;
            string neighborLeftName = "cell" + neighborLeft;
            string neighborRightName = "cell" + neighborRight;
            bool valNeighborLeft = neighborLeftName ! getDisplay {myName};
            bool valNeighborRight = neighborRightName ! getDisplay {myName};
            if ((valNeighborLeft) && (currCellIsOn)
                    && (valNeighborRight)) {
                nextStateCurrCellIsOn = false;
            } else if ((valNeighborLeft) && (currCellIsOn)
                    && (!(valNeighborRight))) {
                nextStateCurrCellIsOn = true;
            } else if ((valNeighborLeft) && (!(currCellIsOn))
                    && (valNeighborRight)) {
                nextStateCurrCellIsOn = true;
            } else if ((valNeighborLeft) && (!(currCellIsOn))
                    && (!(valNeighborRight))) {
                nextStateCurrCellIsOn = false;
            } else if ((!(valNeighborLeft)) && (currCellIsOn)
                    && (valNeighborRight)) {
                nextStateCurrCellIsOn = true;
            } else if ((!(valNeighborLeft)) && (currCellIsOn)
                    && (!(valNeighborRight))) {
                nextStateCurrCellIsOn = true;
            } else if ((!(valNeighborLeft)) && (!(currCellIsOn))
                    && (valNeighborRight)) {
                nextStateCurrCellIsOn = true;
            } else {
                nextStateCurrCellIsOn = false;
            }
            go Display {}
        }
        state Update {
            currCellIsOn = nextStateCurrCellIsOn;
            go Display {}
        }
    }
}
```

05

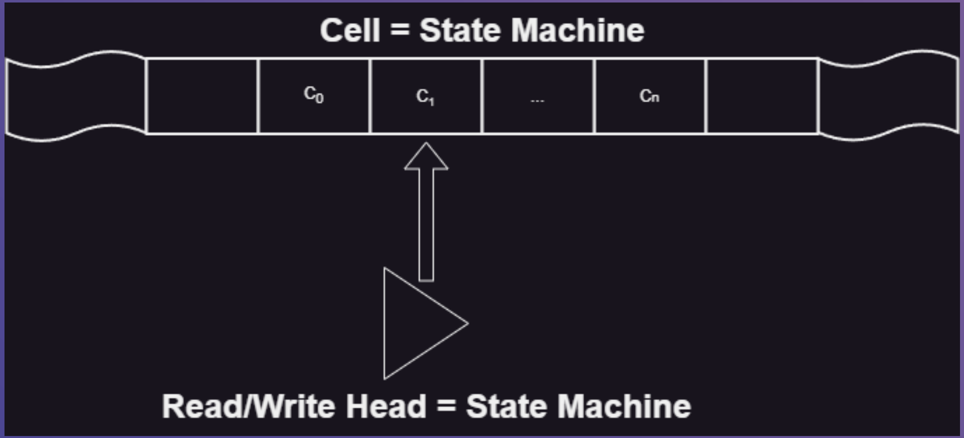
Conclusion

- Summary
- Future Thoughts
- Final Remarks

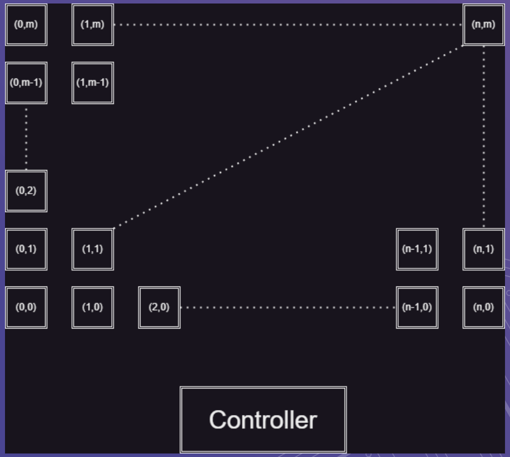
Summary

Proteus is TC, as shown by the 3 different examples:

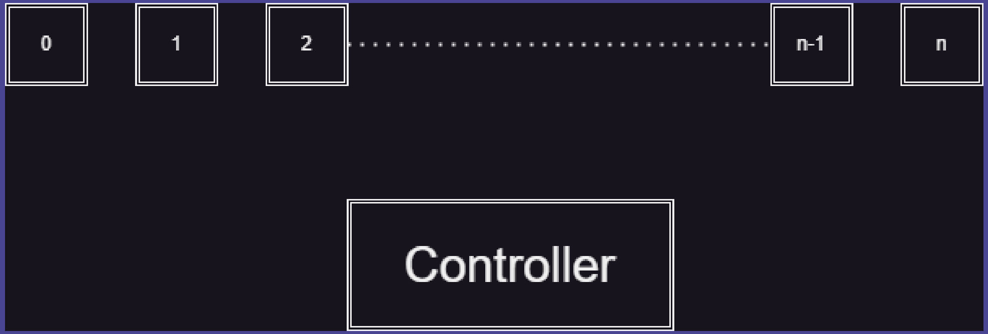
1. Automata Theory with Undecidable Input
2. Implementation of CGoL
3. Implementation of Rule 110



Theoretical Design of TM



System Design for CGoL



System Design for Rule 110

Future Thoughts

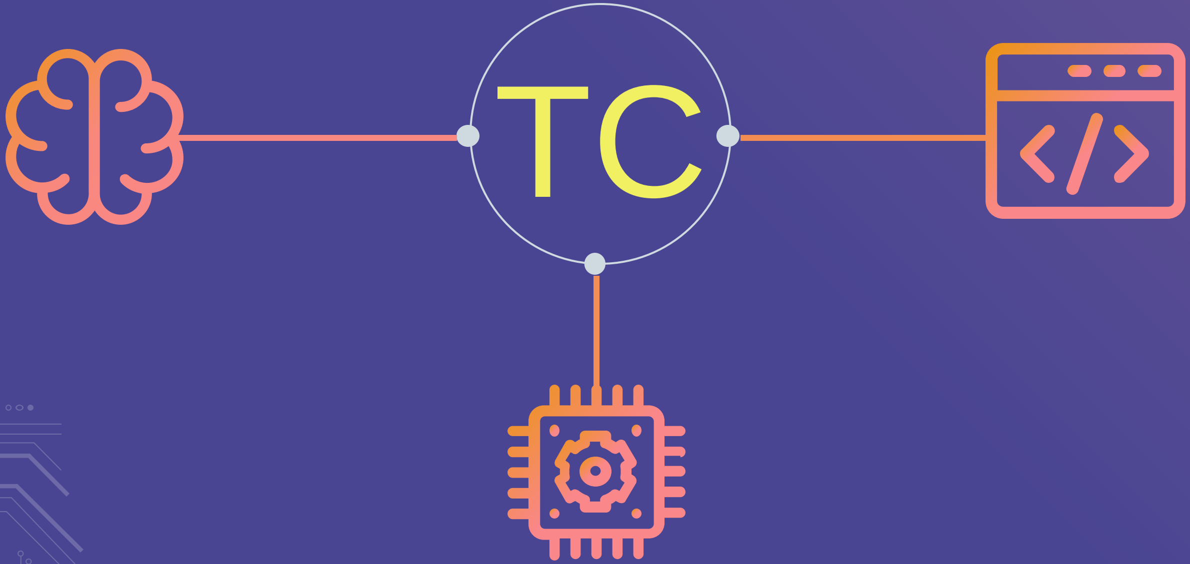
- No way to implement interactive user input for a brainfuck interpreter
  - Pre-feeding data may fix issue
- Truncation to maintain closure of $\mathbb{Z}$

  $\Rightarrow$ there is no way to implement a proper calculator directly
- Lambda Calculus is highly theoretical and requires definitions for restructuring the language
- Very tedious to create a complete architecture for a functional computer, one logical gate at a time.

Final Remarks

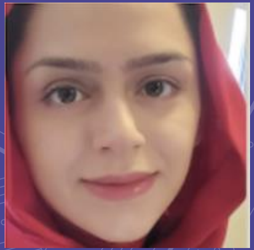
1. Different disciplines approaches to Turing Completeness

2. Proteus being useful IRL



Thank You!

Questions?



CREDITS: This presentation template was created by **Slidesgo,** and includes icons by **Flaticon,** and infographics & images by **Freepik**

References

- R. Gandy, “Church’s Thesis and Principles for Mechanisms,” The Kleene Symposium, pp. 123 –148, Jun. 1980.
- Sakharov, Alex. “Rice’s Theorem.” From MathWorld–A Wolfram Web Resource, created by Eric W. Weisstein, Wolfram.com, 2024. https://mathworld.wolfram.com/RicesTheorem.html
- Wikipedia Contributors, “Rice’s theorem,” Wikipedia, Sep. 24, 2019. https://en.wikipedia.org/wiki/Rice%27s theorem
- “Brainfuck,” Wikipedia, Oct. 05, 2021. https://en.wikipedia.org/wiki/Brainfuck
- 262588213843476, “Basics Of BrainFuck,” Gist, Oct. 29, 2024. https://gist.github.com/roachhd/dce54bec8ba55fb17d3a
- A. Churchill, S. Biderman, and A. Herrick, “Magic: The Gathering is Turing Complete.”
- “I Made A Working Computer With Just Redstone!,” www.youtube.com. https://www.youtube.com/watch?v=CW9N6kGbu2I
- G. Branwen, “Surprisingly Turing-Complete,” gwern.net, Dec. 2012, Available: https://gwern.net/turing-complete
- “Accidentally Turing-Complete,” beza1e1.tuxen.de. https://beza1e1.tuxen.de/articles/accidentally_turing_complete.html
- B. McClelland, “Adding Runtime Verification to the Proteus Language,” CSUN, May 2021. P. Rendell, ”A Universal Turing Machine in Conway’s Game of Life,” 2011 International Conference on High Performance Computing & Simulation, Istanbul, Turkey, 2011, pp. 764-772, doi: 10.1109/HPCSim.2011.5999906.
- P. Linz, An Introduction to Formal Languages and Automata. Jones & Bartlett Learning, 2016.
- Jeffrey Outlaw Shallit, A second course in formal languages and automata theory. Cambridge; New York: Cambridge University Press, 2009.

References

- M. Bic¸er, F. Albayrak and U. Orhan, ”Automatic Automata Grading System Using JFLAP,” 2023 Innovations in Intelligent Systems and Applications Conference (ASYU), Sivas, Turkiye, 2023, pp. 1-4, doi: 10.1109/ASYU58738.2023.10296744.
- Dezani-Ciancaglini Mariangiola And J. R. Hindley, “Lambda-Calculus,” Wiley Encyclopedia of Computer Science and Engineering, pp. 1–8, Sep. 2008, doi: https://doi.org/10.1002/9780470050118.ecse212.
- M. Dezani-Ciancaglini and J. R. Hindley, “Lambda-Calculus,” Nov. 2007, Available: https://www.researchgate.net/publication/228107078 Lambda-Calculus
- R. Rojas, “A Tutorial Introduction To The Lambda Calculus,” 2015, Available: https://arxiv.org/pdf/1503.09060
- “Home — nand2tetris,” nand2tetris, 2017. https://www.nand2tetris.org/
- Noam Nisan, ELEMENTS OF COMPUTING SYSTEMS : building a modern computer from first principles. 2020.
- B. McClelland et al., “Towards a Systems Programming Language Designed for Hierarchical State Machines,” pp. 23–30, Jul. 2021, doi: https://doi.org/10.1109/smc-it51442.2021.00010.
- speeder, “How does the Brainfuck Hello World actually work?,” Stack Overflow, May 30, 2013. https://stackoverflow.com/questions/16836860/how-does-the-brainfuck-hello-world- actually-work/19869651#19869651
- “brainfuck-Esolang,” Esolangs.org, 2023. https://esolangs.org/wiki/Brainfuck#Self-interpreters
- Brainfuck.org, 2024. https://brainfuck.org/dbfi.b
- Wikipedia Contributors, “Cellular automaton,” Wikipedia, Dec. 05, 2019. https://en.wikipedia.org/wiki/Cellular automaton
- E. W. Weisstein, “Cellular Automaton,” From Mathworld–A Wolfram Web Resource, math-world.wolfram.com. https://mathworld.wolfram.com/CellularAutomaton.html

References

- A. Ilachinski, Cellular Automata. World Scientific, 2001.
- “Math’s ‘Game of Life’ Reveals Long-Sought Repeating Patterns — Quanta Magazine,” Quanta Magazine, Jan. 18, 2024. https://www.quantamagazine.org/maths-game-of-life-reveals- long-sought-repeating-patterns-20240118/
- Code & Optimism, “How to Write A Turing-Complete Programming Language In 40 Minutes In Ruby Using Bable-Bridge,” YouTube, Sep. 19, 2012. https://www.youtube.com/watch?v= Uoyufkb5lk
- M. Kenyon, “How to Write a Brainfuck Interpreter in C#,” Thesharperdev.com, Oct. 12, 2019. https://thesharperdev.com/how-to-write-a-brainfuck-interpreter-in-c/
- “Compiling to Brainf#ck - Meep.,” InJuly, 2024. https://injuly.in/blog/bfinbf/index.html
- srijan-paul, “GitHub - srijan-paul/meep: A programming language that compiles to brainfuck.,” GitHub, 2020. https://github.com/srijan-paul/meep
- M. Ueding, “Creating a Brainfuck interpreter,” Martin Ueding, Apr. 19, 2023. https://martinueding.de/posts/creating-a-brainfuck-interpreter/
- “Functional Programming,” learn.saylor.org. https://learn.saylor.org/mod/book/tool/print/-index.php?id=33044&chapterid=13087
- Wikipedia Contributors, “Lambda calculus,” Wikipedia, Jan. 02, 2020. https://en.wikipedia.org/wiki/Lambda calculus
- “Collected Lambdas,” jwodder.freeshell.org. https://jwodder.freeshell.org/lambda.html
- Wikipedia Contributors, “SKI Combinator calculus,” Wikipedia, Oct. 28, 2024. https://en.wikipedia.org/wiki/SKI combinator calculus

References

- “Reddit - Dive into anything,” Reddit.com, 2017. https://www.reddit.com/r/ProgrammerHu-mor/comments/78z90f/when you need to know if a number is even or odd/
- “Reddit - Dive into anything,” Reddit.com, 2017. https://www.reddit.com/r/ProgrammerHu-mor/comments/78z90f/comment/doylzry/
- “The Excel Formula Language Is Now Turing-Complete,” InfoQ. https://www.infoq.com/articles/excel-lambda-turing-complete/
- “A deep dive into an NSO zero-click iMessage exploit: Remote Code Execu- tion,” Blogspot.com, 2021. https://googleprojectzero.blogspot.com/2021/12/a-deep-dive-into- nso-zero-click.html?m=1
- Wikipedia Contributors, “Actor model,” Wikipedia, Nov. 14, 2019. https://en.wikipedia.org/wiki/Actor model
- L. Nigro, “Parallel Theatre:
- An actor framework in Java for high performance computing,” Simulation Modelling Practice and Theory, vol. 106, p. 102189, Jan. 2021, doi: https://doi.org/10.1016/j.simpat.2020.102189.
- GeeksForGeeks, “Functional Programming Paradigm - GeeksforGeeks,” GeeksforGeeks, Jan. 02, 2019. https://www.geeksforgeeks.org/functional-programming-paradigm/
- coq, “GitHub - coq/coq: Coq is a formal proof management system. It provides a formal language to write mathematical definitions, executable algorithms and theorems together with an environment for semi-interactive development of machine-checked proofs.,” GitHub, Sep. 04, 2024. https://github.com/coq/coq?tab=readme-ov-file
- “A Simple ALU, Drawn From The ZipCPU,” zipcpu.com. https://zipcpu.com/zipcpu/2017/08/11/simple-alu.html
- Y. N. Patt and S. J. Patel, Introduction to computing systems : from bits and gates to C and beyond. Boston: Mcgraw-Hill\Higher Education, Cop, 2005.

Additional Slides

CREDITS: This presentation template was created by **Slidesgo,** and includes icons by **Flaticon,** and infographics & images by **Freepik**

Proteus Grammar – Pt. 1

Program: DefEvent* DefGlobalConst* DefFunc* DefActor+
DefActor: 'actor' ActorName '{' ActorItem* '}'
ActorItem: DefHSM | DefActorOn | DefMember | DefMethod
DefActorOn: 'on' EventMatch OnBlock
DefHSM: 'statemachine' '{' StateItem* '}'
DefState: 'state' StateName '{' StateItem* '}'
StateItem: DefOn | DefEntry | DefExit | DefMember | DefMethod | DefState | InitialState
DefOn: 'on' EventMatch OnBody
EventMatch: EventName '{' [VarName (',' VarName)*] '}'
OnBody: GoStmt | OnBlock
OnBlock: Block
DefEntry: 'entry' '{' Block '}'
DefExit: 'exit' '{' Block '}'
DefMember: Type VarName '=' ConstExpr ';'
DefMethod: 'func' FuncName FormalFuncArgs ['->' Type] Block
InitialState: 'initial' StateName ';'
Block: '{' Stmt* '}'

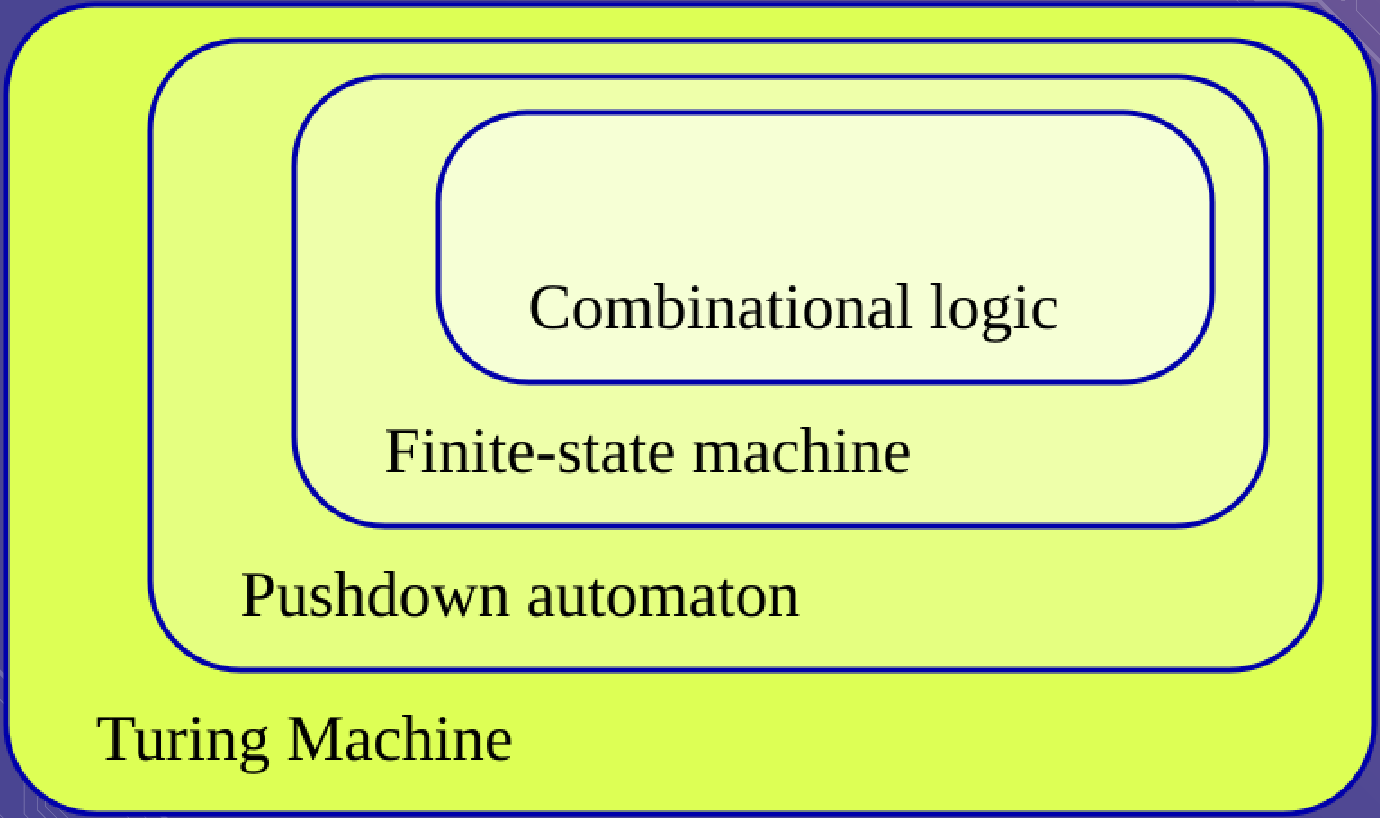
Stmt: IfStmt | WhileStmt | DecStmt | AssignStmt | ExitStmt | ApplyStmt | SendStmt | PrintStmt | PrintlnStmt
DefEvent: 'event' EventName '{' [Type (',' Type)*] }' ';'
DefFunc: 'func' FuncName FormalFuncArgs ['->' Type] Block
DefGlobalConst: 'const' Type VarName '=' ConstExpr ';'
ExitStmt: 'exit' '(' NUMBER ')' ';'
ReturnStmt: 'return' Expr ';'
DecStmt: Type VarName '=' Expr ';'
AssignStmt: VarName '=' Expr ';'
ApplyStmt: ApplyExpr ';'
SendStmt : HSMName '!' EventName ExprListCurly ';'
PrintStmt : 'print' ExprListParen ';'
PrintlnStmt : 'println' ExprListParen ';'
FormalFuncArgs : '(' [Type VarName (',' Type VarName)*] ')'
ExprListParen :'(' [Expr (',' Expr)*] ')'
ExprListCurly :'{' [Expr (',' Expr)*] '}'

Proteus Grammar – Pt. 2

Type: 'int' | 'string' | 'bool' | 'actorname' | 'statename' | 'eventname'
GoStmt: JustGoStmt | GoIfStmt
JustGoStmt: 'go' StateName Block
GoIfStmt: 'goif' ParenExpr StateName Block ['else' (GoIfStmt | ElseGoStmt)]
ElseGoStmt: 'go' StateName Block
IfStmt: 'if' ParenExpr Block ['else' (IfStmt | Block)]
WhileStmt: 'while' ParenExpr Block
ParenExpr: '(' Expr ')'
ConstExpr: IntExpr | BoolExpr | StrExpr
Expr: ValExpr | BinOpExpr | ApplyExpr
BinOpExpr: ValExpr BinOp Expr
BinOp: '*' | '/' | '%' | '+' | '-' | '<<' | '>>' | '<' | '>' | '<=' | '>=' | '==' | '!=' | '^' | '&&' | '||' | '*=' | '/=' | '%=' | '+=' | '-=' | '<<=' | '>>=' | '^='
ApplyExpr: FuncName ExprListParen
ValExpr: VarExpr | IntExpr | StrExpr | BoolExpr | ActorExpr | StateExpr | EventExpr | ParenExpr

VarExpr: VarName
IntExpr: NUMBER
StrExpr: STRING
BoolExpr: BOOL
ActorExpr: 'actor' ActorName
StateExpr: 'state' StateName
EventExpr: 'event' EventName
StateName: NAME
ActorName: NAME
FuncName: NAME
VarName: NAME
EventName: NAME

Automata Theory levels of computation



Proof Assistants

Initial idea was to use Proof Assistants for the proof

Impossible => Solution to the Halting Problem for all Proof assistants

All well-formed proofs written to a proof assistant must be valid. Thus, they all halt.

Impossible to use a Non-TC system to show that a system is TC

Brainfuck

One of the simplest known TC programming languages

30K byte cell array with an I/O mechanism

`>`	Increments the data pointer by one. (This points to the next cell on the right).
`<`	Decrement the data pointer by one. (This points to the next cell on the left).
`+`	Increments the byte at the data pointer by one.
`−`	Decrements the byte at the data pointer by one.
`.`	Output the byte at the data pointer.
`,`	Accept one byte of input, storing its value in the byte at the data pointer.
`[`	If the byte at the data pointer is zero, then instead of moving the instruction forward to the next command, go to the matching ']' command. (Jump forwards).
`]`	If the byte at the data pointer is non-zero, then instead of moving the instruction forward to the next command, go to the matching '[' command. (Jump backwards).

Brainfuck Instruction set

`>++++++++[<+++++++++>-]<.`	H	`>++++++[<+++++++++>-]<+.`	W
`>++++[<+++++++>-]<+.`	e	`<.`	o
`+++++++..`	l	`+++.`	r
`+++.`	l	`------.`	l
`>>++++++[<+++++++>-]<++.`	o	`--------.`	d
`------------.`	"space"	`>>>++++[<++++++++>-]<+.`	!

“Hello World!” in Brainfuck

SKI Combinator Calculus

SKI combinator calculus defines 3 rules:

1. Ix =x
2. Kxy = x
3. Sxyz = xz(yz)

Can be likened to a reduced version of Lambda Calculus

$$\mathrm{SKIK} \Rightarrow \mathrm{KK(IK)} \Rightarrow \mathrm{KKK} \Rightarrow \mathrm{K}$$

$\mathrm{SKI(KIS)}$

- $\mathrm{SKI(KIS)} \Rightarrow \mathrm{K(KIS)(I(KIS))} \Rightarrow \mathrm{KIS} \Rightarrow \mathrm{I}$
- $\mathrm{SKI(KIS)} \Rightarrow \mathrm{SKII} \Rightarrow \mathrm{KI(II)} \Rightarrow \mathrm{KII} \Rightarrow \mathrm{I}$

$\mathrm{KS(I(SKSI))}$

- $\mathrm{KS(I(SKSI))} \Rightarrow \mathrm{KS(I(KI(SI)))} \Rightarrow \mathrm{KS(I(I))} \Rightarrow \mathrm{KS(II)} \Rightarrow \mathrm{KSI} \Rightarrow \mathrm{S}$
- $\mathrm{KS(I(SKSI))} \Rightarrow \mathrm{S}$

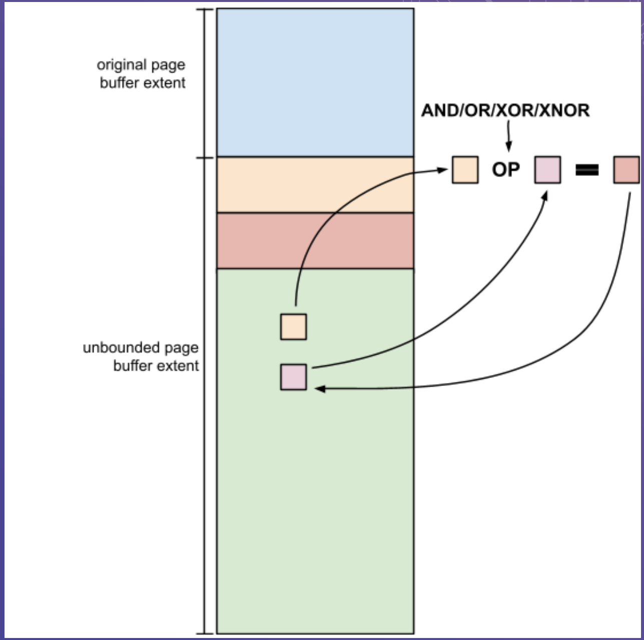
Examples of SKI Combinator Calculus reductions

Why does TC matter? – Security

If a TM can do anything*, then it if a bad actor can get ahold of it, then they will cause harm to the system/user

Important to be cognizant of the weapon that one wields when doing operations

Fire warms the hearth but can also raze a forest



iMessage vulnerability caused by memory overflow

Why does TC matter? – System Limits

To know the limits of the system.

A basic calculator is not TC. Therefore, it cannot be used to perform any malicious behavior.

A general-purpose programming language is TC. This means that programmers should be cognizant of the extent of operations the language can perform.

Smart vs Dumb Fridge have different technological requirements

What is Undecidability?

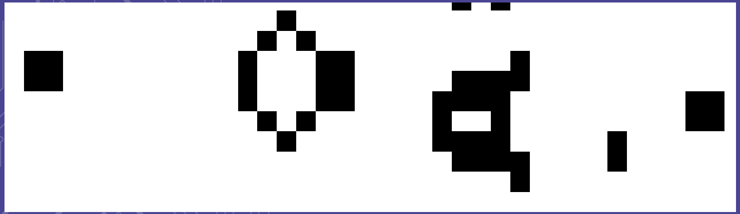
There is no algorithm to answer the question proposed, and it is impossible to create one.

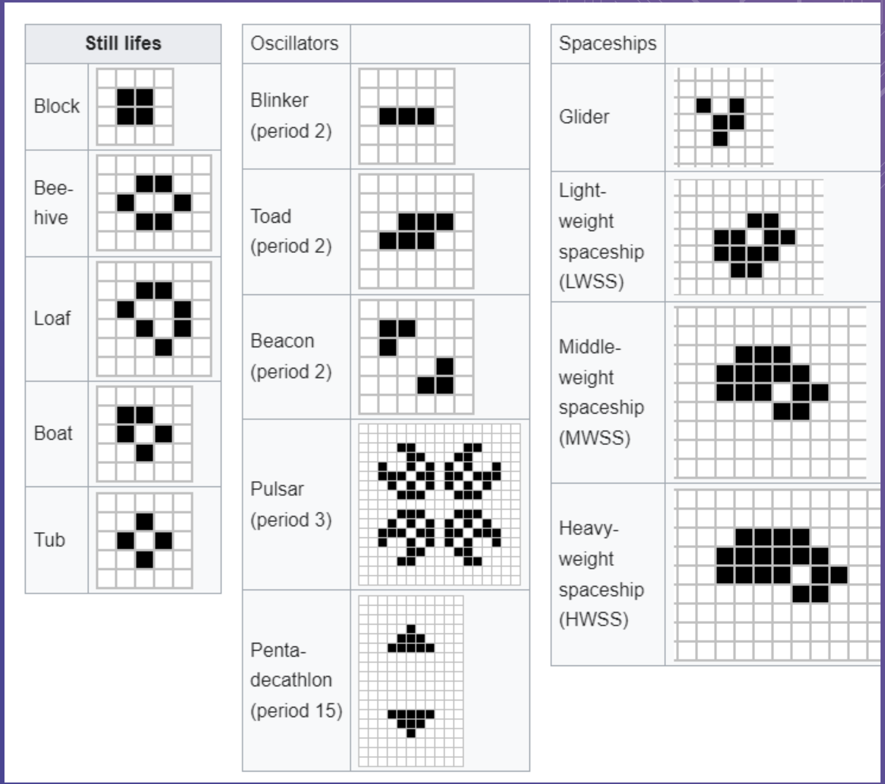
E.g. The Halting Problem: Given an arbitrary program, can one determine if the program will finish running (halt)?

How is CGoL TC?

Undecidable whether one state can reach another in general

Class 4 cellular automaton according to Wolfram, meaning that an initial config may have chaotic or oscillating behavior after an indeterminate amount of time



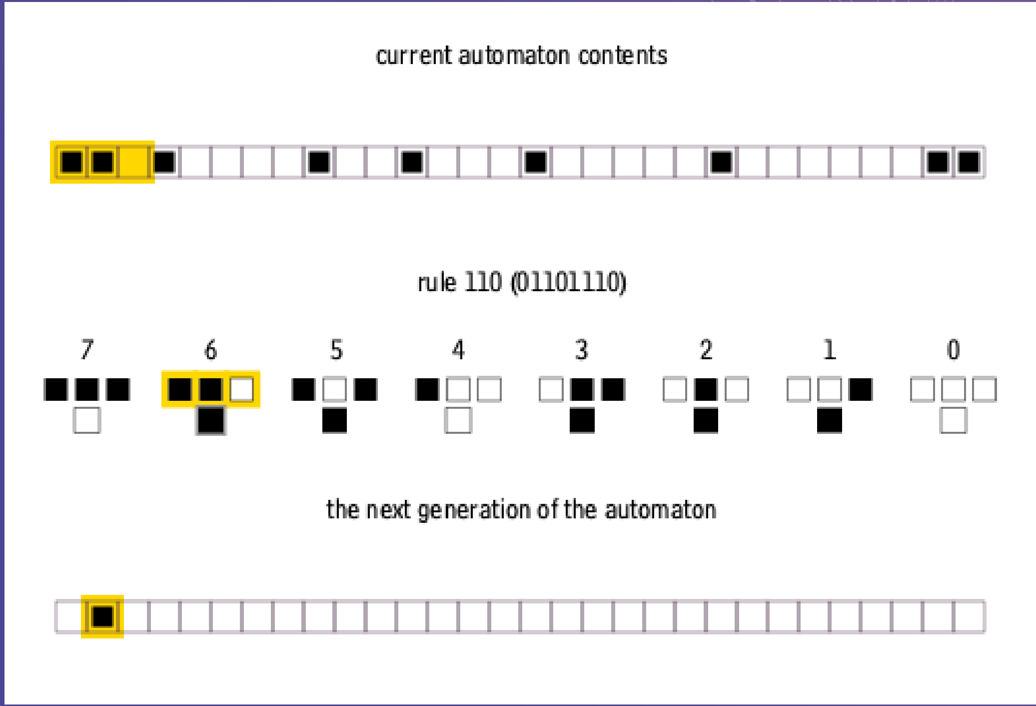


Different structures for CGoL

How is Rule 110 TC?

Undecidable whether one state can reach another in general

Class 4 cellular automaton according to Wolfram, meaning that an initial display may have chaotic or oscillating behavior for an indefinite amount of time



Rule 110 animated