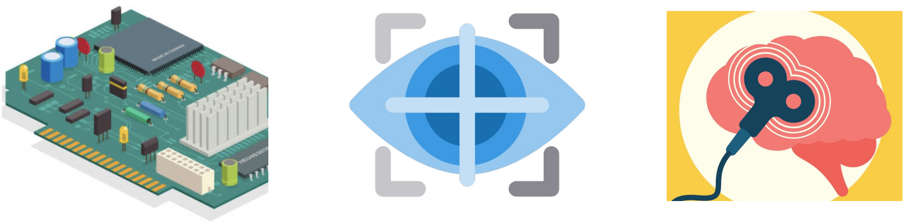


Understanding *understanding*: How do we reason about computational logic?

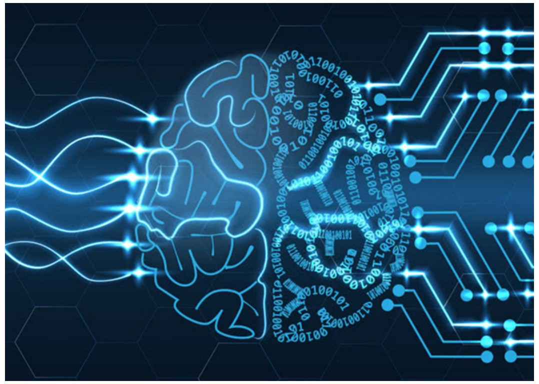
Hammad Ahmad
(he/him)
University of Michigan, Ann Arbor

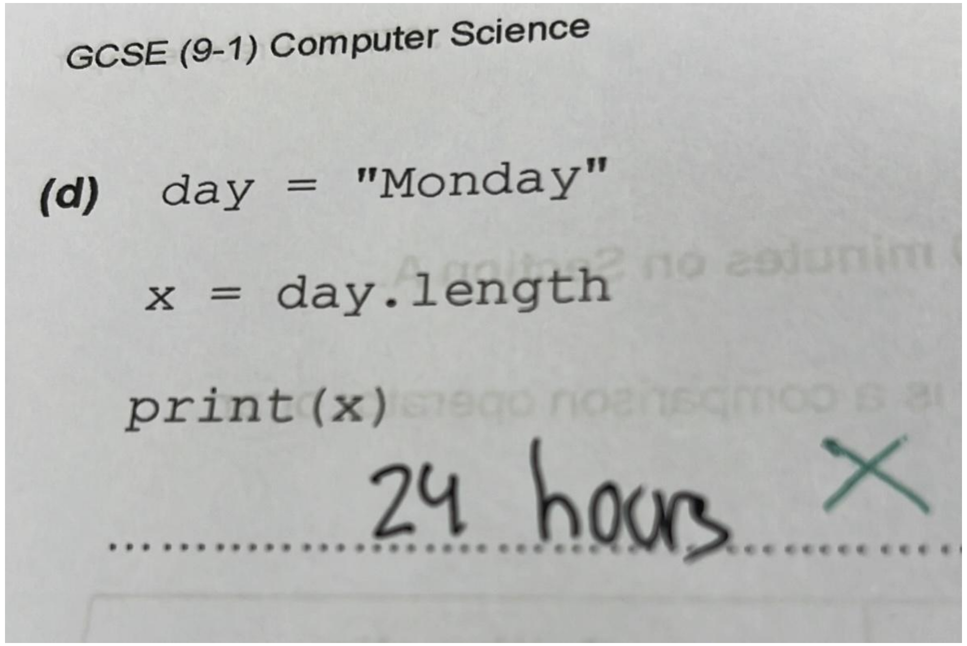
“Understanding *understanding*”

Cognition: Mental processes involved in comprehension and gaining knowledge

“Computational Logic”

Computers do not think like humans do!



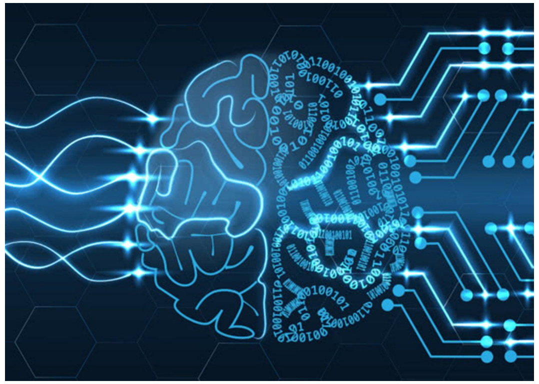


GCSE (9-1) Computer Science

(d)
```
day = "Monday"
x = day.length
print(x)
```

24 hours

“Computational Logic”

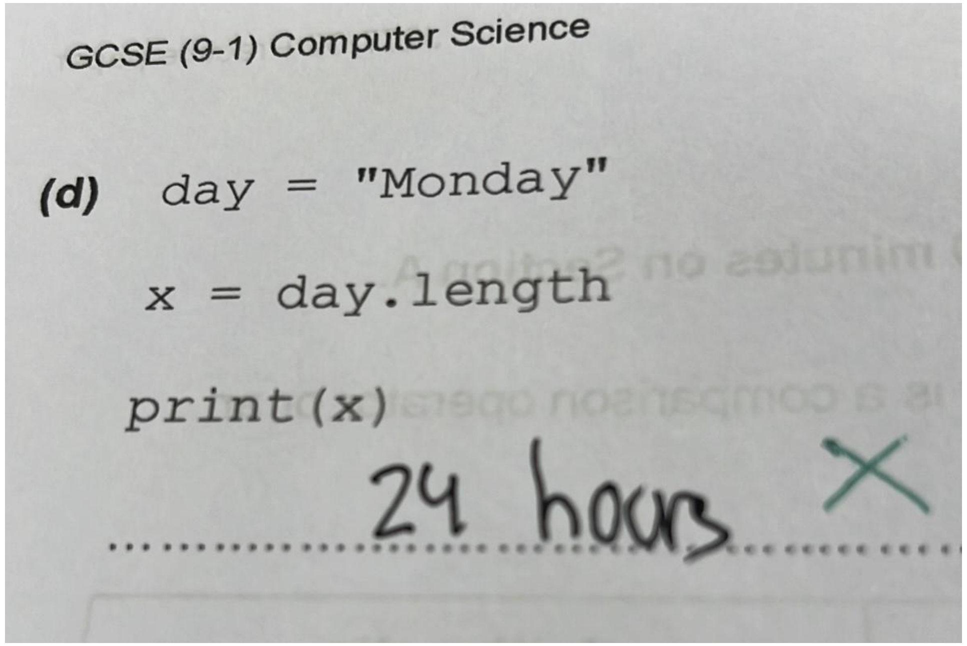


Computers do not think like humans do!

Future industry professionals and academics **need to be trained for computational logic** reasoning

Logical reasoning in CS forms a **core component** of undergraduate CS curricula

Introductory CS courses structured around **cultivating creative thinking and problem solving** using logical reasoning



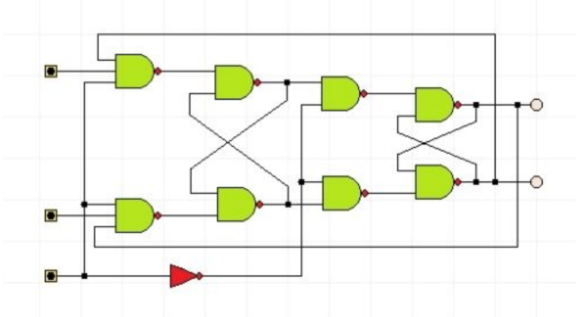
GCSE (9-1) Computer Science

(d)
```
day = "Monday"
x = day.length
print(x)
```
24 hours ✗

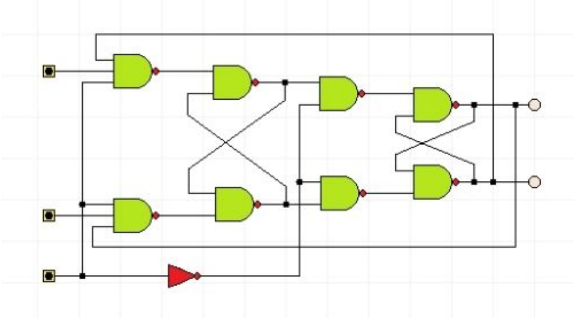
Defining “Logic”

Digital logic

(e.g., hardware designs; EECS 215, 270)

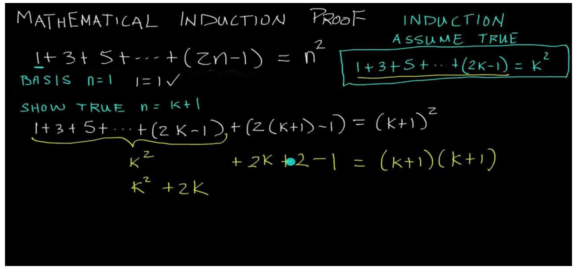


Defining “Logic”



Digital logic

(e.g., hardware designs; EECS 215, 270)



MATHEMATICAL INDUCTION PROOF

$1+3+5+\cdots+(2n-1)=n^2$

BASIS $n=1$ $1=1$ ✓

INDUCTION

ASSUME TRUE

$\boxed{1+3+5+\cdots+(2K-1)=K^2}$

SHOW TRUE $n=K+1$

$\underbrace{1+3+5+\cdots+(2K-1)}_{K^2}+(2(K+1)-1)=(K+1)^2$

$K^2+2K+2-1=(K+1)(K+1)$

K^2+2K

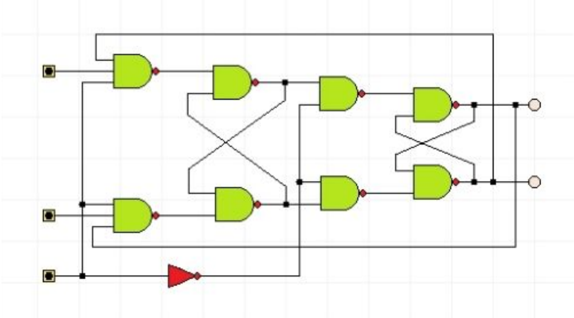
Mathematical logic

(e.g., proofs about algorithms; EECS 203, 376)

Defining "Logic"

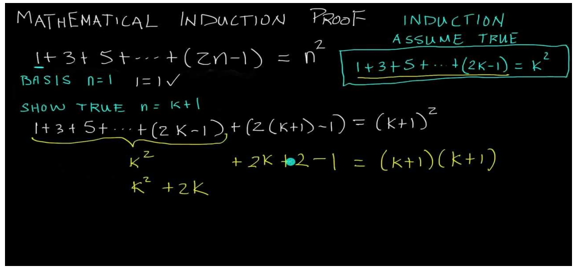
Digital logic

(e.g., hardware designs; EECS 215, 270)



Mathematical logic

(e.g., proofs about algorithms; EECS 203, 376)



MATHEMATICAL INDUCTION PROOF

$1+3+5+\cdots+(2n-1) = n^2$

BASIS $n=1$ $1=1$ ✓

INDUCTION ASSUME TRUE

$\boxed{1+3+5+\cdots+(2K-1) = K^2}$

SHOW TRUE $n = K+1$

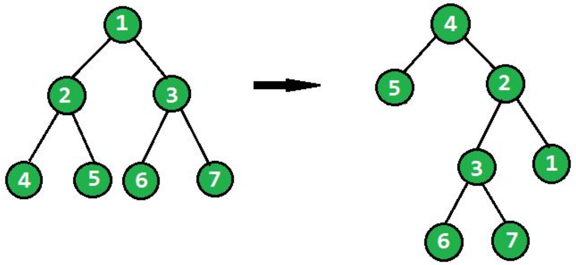
$\underbrace{1+3+5+\cdots+(2K-1)}_{K^2} + (2(K+1)-1) = (K+1)^2$

$+2K+2-1 = (K+1)(K+1)$

K^2+2K

Programming logic

(e.g., manipulating data structures; EECS 183, 281)



Why should we care about cognition?

A CS1 Spatial Skills Intervention and the Impact on Introductory Programming Abilities

Ryan Bockmon Stephen Cooper William Koperski

Denton, Texas Cincinnati, Ohio Charlotte

Does spatial skills instruction improve STEM outcomes? The answer is ‘yes’

Sheryl Sorby[a,*], Norma Veurink[b], Scott Streiner[c]

Development of a cognition-priming model describing learning in a STEM classroom

Richa

Cognitive Load Theory in the Context of Teaching and Learning Computer Programming: A Systematic Literature Review

João Henrique Berssanette and Antonio Carlos de Francisco

Insights into numerical cognition: considering eye-fixations in number processing and arithmetic

J. Mock[1] · S. Huber[1] · E. Klein[1,2] · K. Moeller[1,3,4]

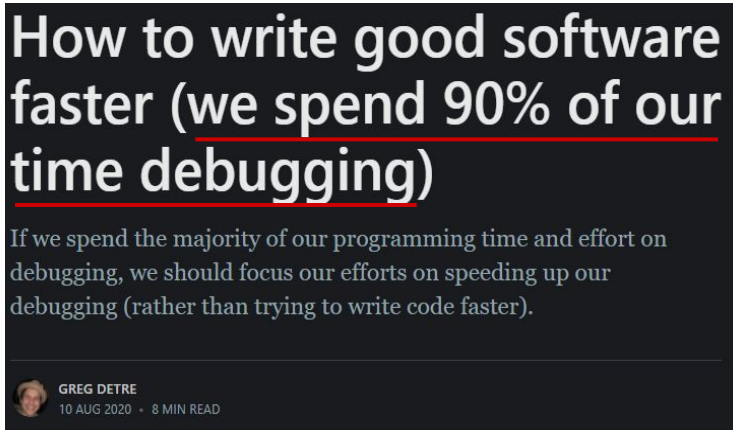
From anecdote to evidence: the relationship between personality and need for cognition of developers

Daniel Russo[1] · Andres R. Masegosa[1] · Klaas-Jan Stol[2]

Understanding software developers’ cognition in agile requirements engineering

Jingdong Jia[a,*], Xiaoying Yang[a], Rong Zhang[b], Xi Liu[a]

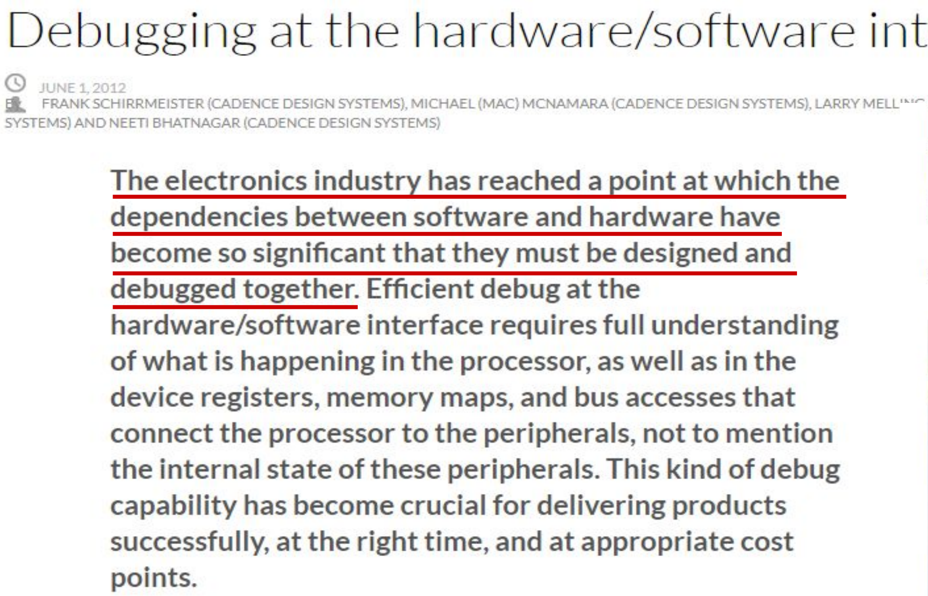
Why should we care about how computers think?



How to write good software faster (we spend 90% of our time debugging)

If we spend the majority of our programming time and effort on debugging, we should focus our efforts on speeding up our debugging (rather than trying to write code faster).

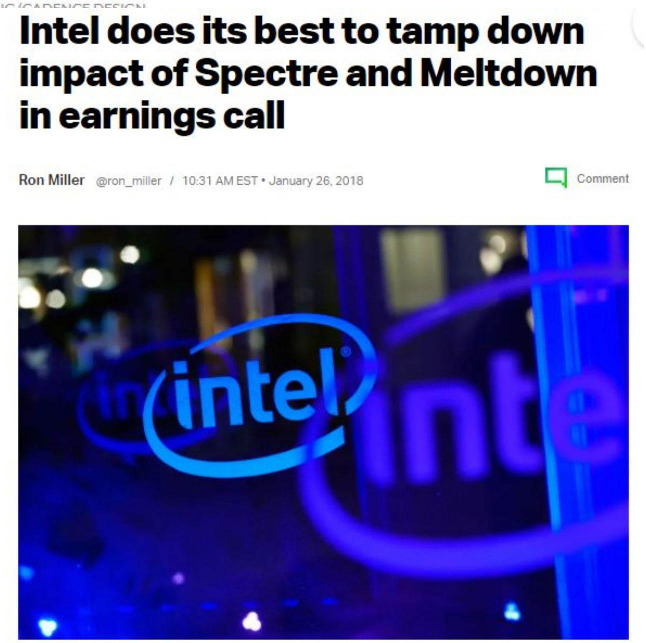
GREG DETRE
10 AUG 2020 · 8 MIN READ



Debugging at the hardware/software interface

JUNE 1, 2012
FRANK SCHIRRMEISTER (CADENCE DESIGN SYSTEMS), MICHAEL (MAC) MCNAMARA (CADENCE DESIGN SYSTEMS), LARRY MELL
SYSTEMS) AND NEETI BHATNAGAR (CADENCE DESIGN SYSTEMS)

The electronics industry has reached a point at which the dependencies between software and hardware have become so significant that they must be designed and debugged together. Efficient debug at the hardware/software interface requires full understanding of what is happening in the processor, as well as in the device registers, memory maps, and bus accesses that connect the processor to the peripherals, not to mention the internal state of these peripherals. This kind of debug capability has become crucial for delivering products successfully, at the right time, and at appropriate cost points.



Intel does its best to tamp down impact of Spectre and Meltdown in earnings call

Ron Miller @ron_miller / 10:31 AM EST • January 26, 2018

Comment

Amazon's one hour of downtime on Prime Day may have cost it up to $100 million in lost sales

Sean Wolfe Jul 19, 2018, 10:53 AM

YOUTUBE · Published December 14

Google lost $1.7M in ad revenue during YouTube outage, expert says

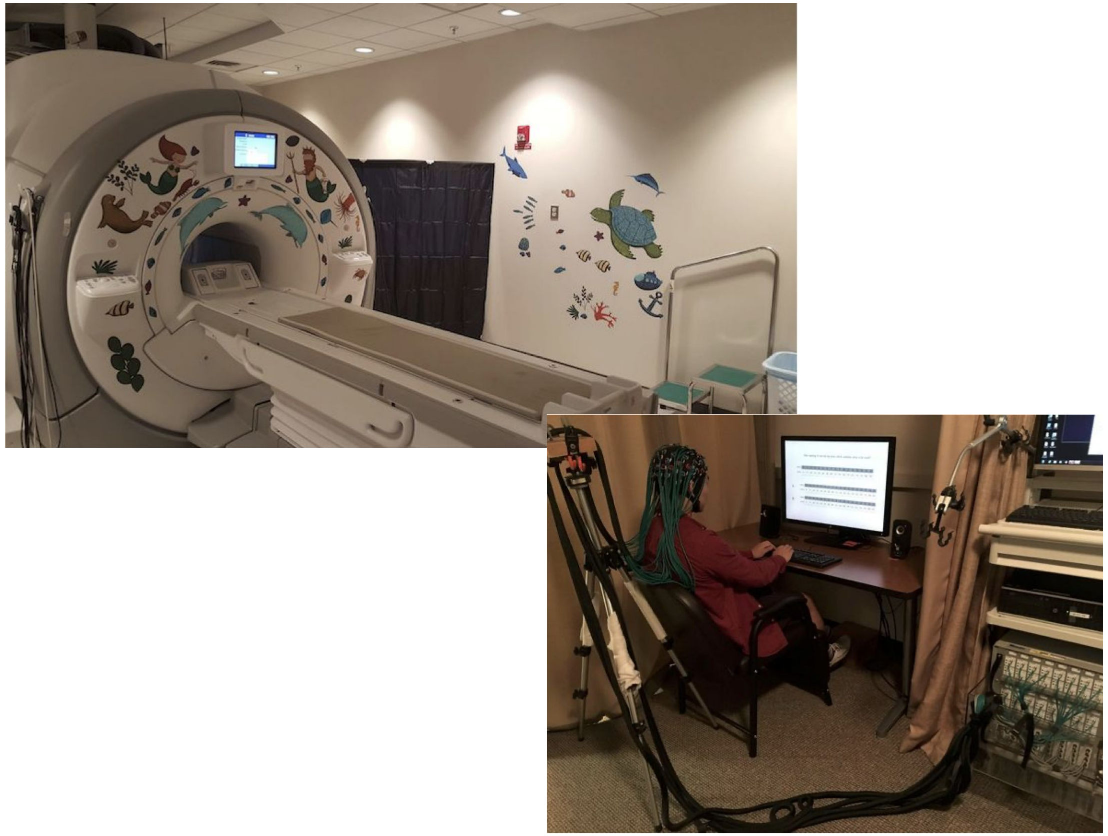
YouTube and other Google services, such as Gmail, suffered outage Monday morning

We want to better understand how programmers *reason about computers*.

Desired Properties in Our Study

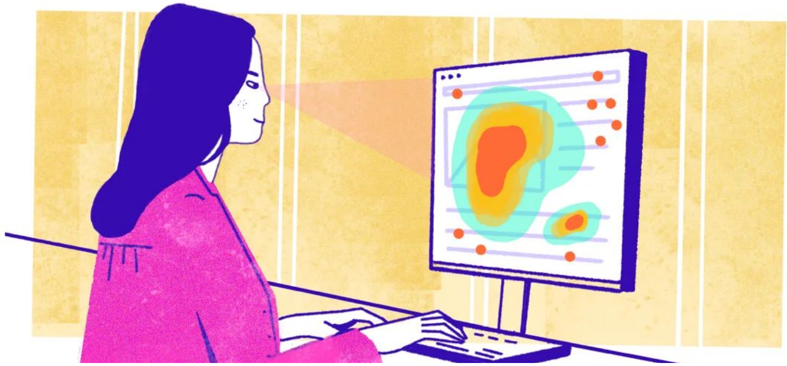
(1) Non-intrusive Methodology

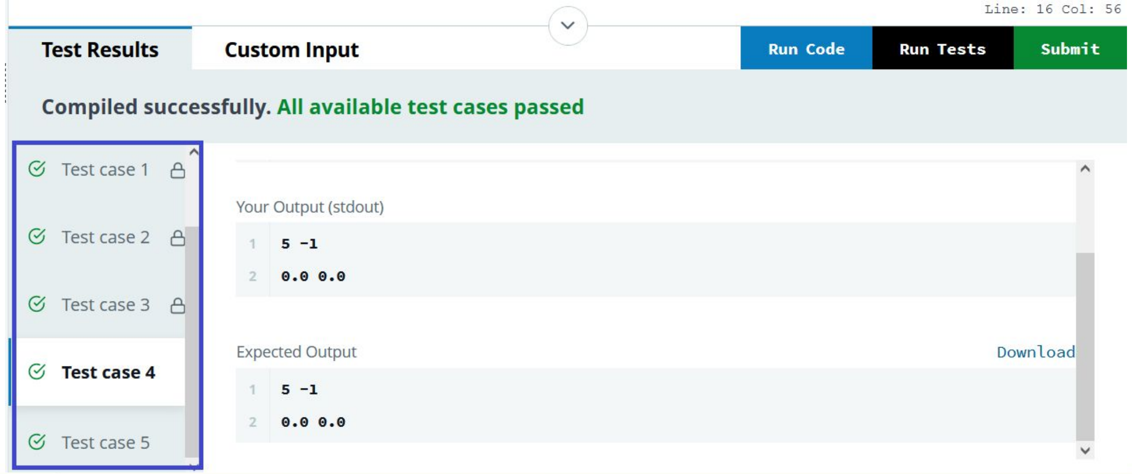
instead of



Desired Properties in Our Study

(2) Objective Measures





Compiled successfully. All available test cases passed

- ✓ Test case 1
- ✓ Test case 2
- ✓ Test case 3
- ✓ **Test case 4**
- ✓ Test case 5

Your Output (stdout)

```
5 -1
0.0 0.0
```

Expected Output

```
5 -1
0.0 0.0
```

instead of

○ Extremely Satisfied

2. To what extent do you agree with the foll

○ Disagree Completely
○ Strongly Disagree
○ Somewhat Disagree
○ Somewhat Agree
○ Strongly Agree
☑ Agree Completely

best about working for T

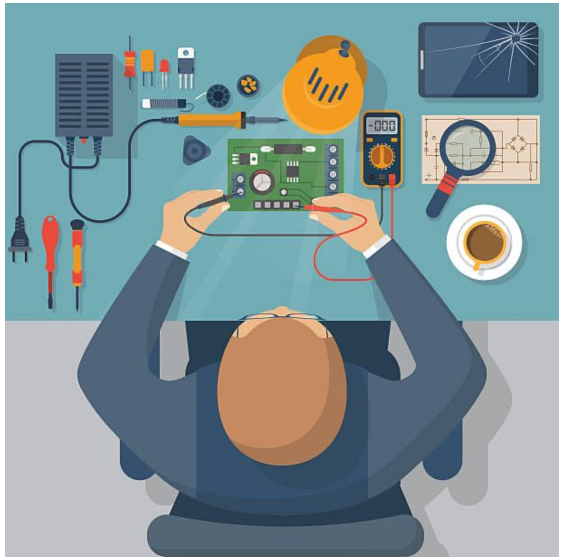
Desired Properties in Our Study

(3) Context-specific Models

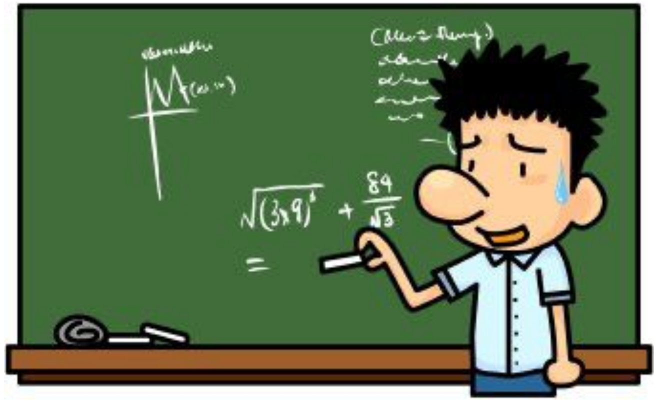
vs.

Desired Properties in Our Study

(3) Context-specific Models



vs.



vs.

Thesis Statement

It is possible to use ***objective measures***

Thesis Statement

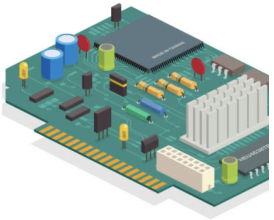
*It is possible to use **objective measures** to obtain **mathematical models** of the **cognitive processes** underlying computational logic reasoning tasks*

Thesis Statement

*It is possible to use **objective measures** to obtain **mathematical models** of the **cognitive processes** underlying computational logic reasoning tasks, and these models can highlight prospective cognitive interventions for student training.*

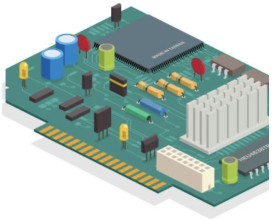
Three Research Components

Three Research Components



Using automated program repair for hardware as a debugging assistant for designers

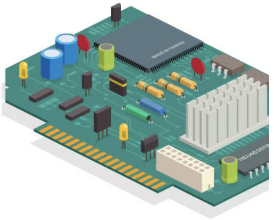
Three Research Components



Using automated program repair for hardware as a debugging assistant for designers

Using eye-tracking to understand cognition for computer science formalisms

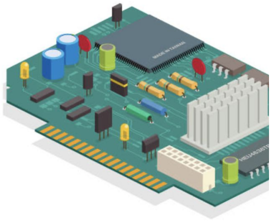
Three Research Components



Using automated program repair for hardware as a debugging assistant for designers

Using eye-tracking to understand cognition for computer science formalisms

Using neurostimulation to investigate the relationship between spatial reasoning and programming

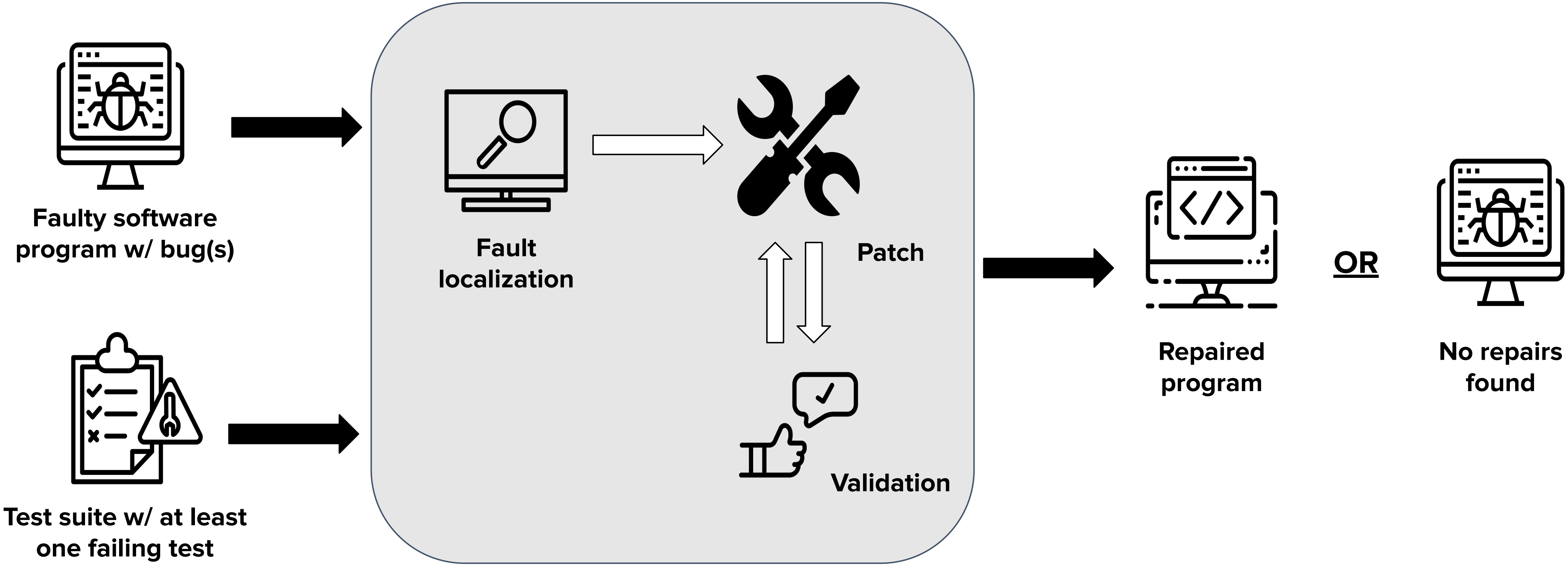


Automated Program Repair for Hardware as a Debugging Assistant

Can we build a state-of-the-art automated repair tool for hardware designs (i.e., digital logic), and use it as a debugging assistant for designers?

Have you ever spent a *long* time finding and fixing a small bug in a program?

Automated Program Repair (APR)



Faulty software program w/ bug(s)

Test suite w/ at least one failing test

Fault localization

Patch

Validation

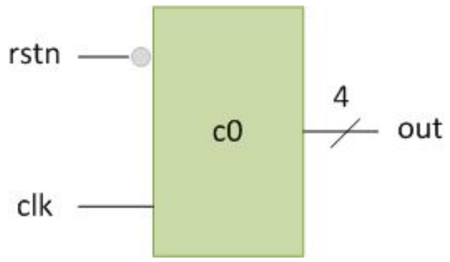
Repaired program

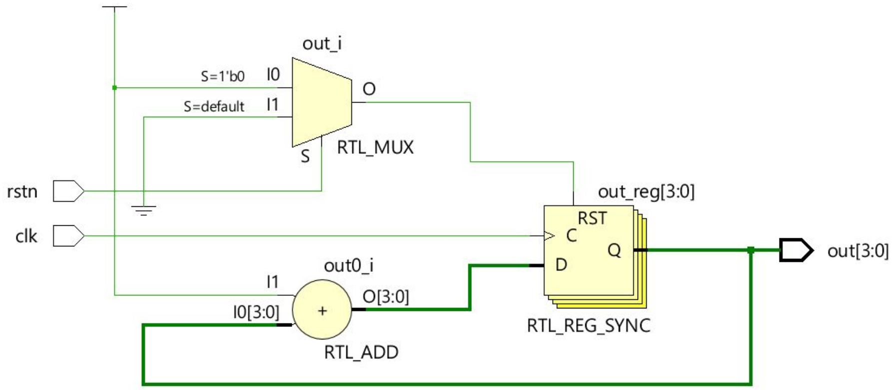
OR

No repairs found

Hardware Designs

Digital specifications for electronic devices, computer systems, or integrated circuits

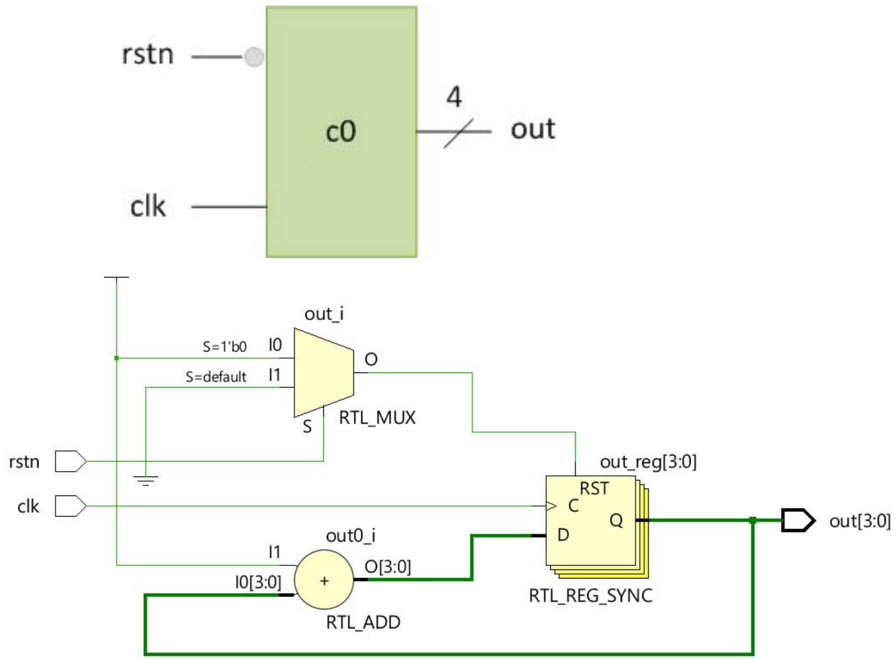




Hardware Designs

Digital specifications for electronic devices, computer systems, or integrated circuits

Typically written using **hardware description languages** (HDLs) like Verilog and VHDL



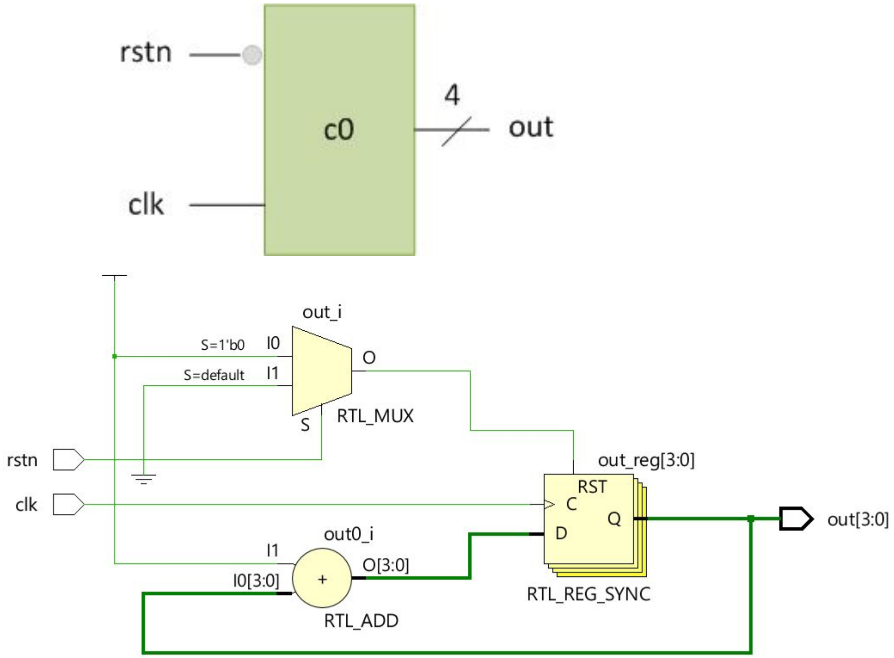
```
module counter ( input clk,
               input rstn,
               output reg[3:0]
           out);
always @ (posedge clk) begin
     if (! rstn) out <= 0;
     else out <= out + 1;
end endmodule
```

Hardware Designs

Digital specifications for electronic devices, computer systems, or integrated circuits

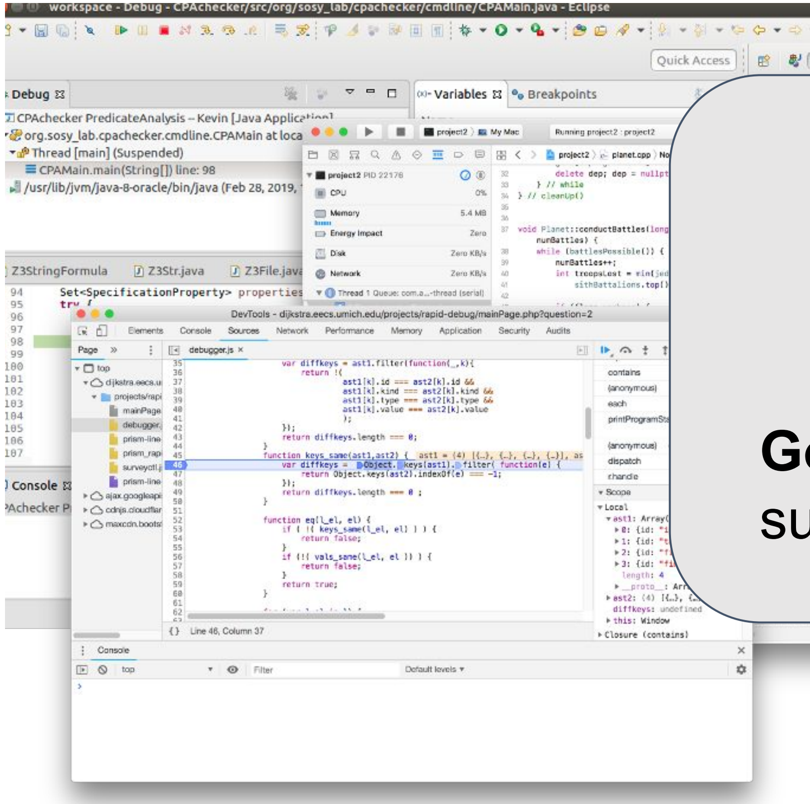
Typically written using **hardware description languages** (HDLs) like Verilog and VHDL

Correspond to the **"stage 0" of the hardware design process**

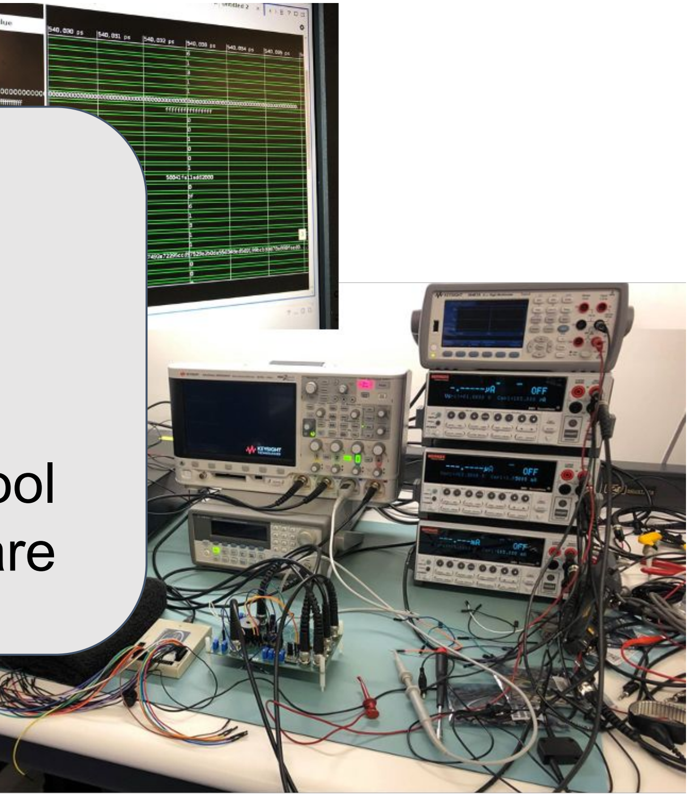


```
module counter ( input clk,
                 input rstn,
                 output reg[3:0]
              out);
always @ (posedge clk) begin
     if (! rstn) out <= 0;
     else out <= out + 1;
end endmodule
```

A Tale of Two Debugging Worlds



Goal: Bridge the gap between tool support for software and hardware



What software developers expect

What hardware designers use

Software vs. Hardware

A key difference: serial execution vs. **parallelism**

```
animals = [“cat”, “dog”, “cat”]
cat_counter = 0
for animal in animals:
    if animal == “cat”:
        cat_counter += 1
print(cat_counter)
```

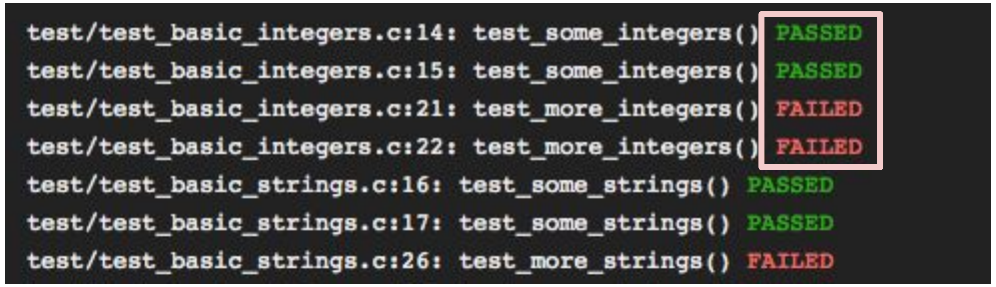
Serial Python code

```
module counter ( input clk,
                 input rstn,
                 output reg[3:0] out);
always @ (posedge clk) begin
    if (! rstn) out <= 0;
    else out <= out + 1;
end endmodule
```

Parallel Verilog code

Software vs. Hardware

Another key difference: test suites vs. **testbenches**



```
test/test_basic_integers.c:14: test_some_integers() PASSED
test/test_basic_integers.c:15: test_some_integers() PASSED
test/test_basic_integers.c:21: test_more_integers() FAILED
test/test_basic_integers.c:22: test_more_integers() FAILED
test/test_basic_strings.c:16: test_some_strings() PASSED
test/test_basic_strings.c:17: test_some_strings() PASSED
test/test_basic_strings.c:26: test_more_strings() FAILED
```

```
Compiler version N-2017.12-SP2-1_Full64; Runtime version N-2017.12-SP2-1_Full64;  Jan 11 11:37 2021
                time,    clk,     reset,   enable, count_out,      overflow_out
                   0,    0,       0,       0,         x,              x
                   5,    1,       0,       0,         x,              x
                            ...
                 250,    0,       0,       1,         5,              1
                 255,    1,       0,       1,         5,              1
                 256,    1,       0,       1,         6,              1
$finish called from file "first_counter_tb_t3.v", line 70.
$finish at simulation time                  258
```

Software APR to Hardware?

Problem: Existing techniques from software APR cannot be directly applied to hardware designs!

How do we repurpose software APR for hardware designs?

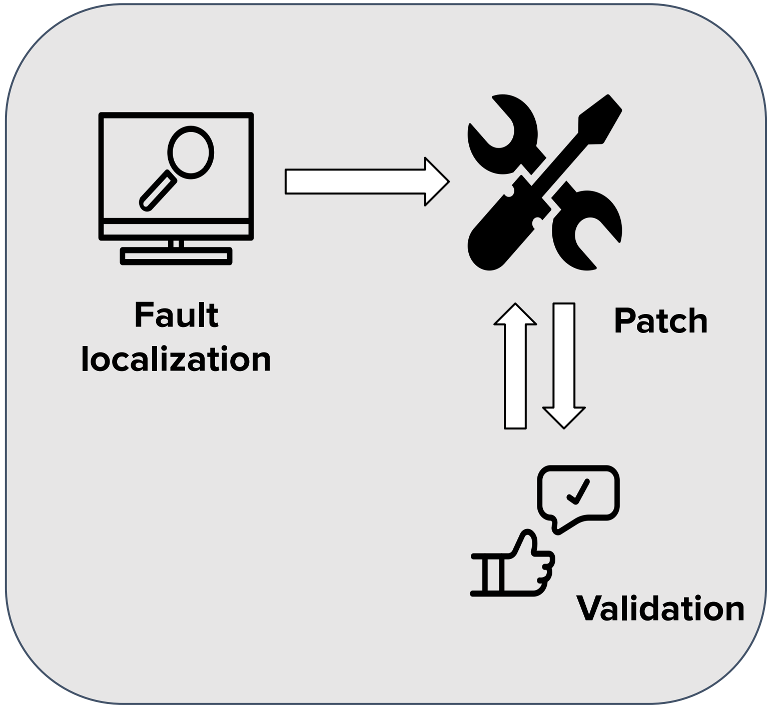
Introducing: *CirFix*

CirFix: A hardware-design focused automated repair algorithm

- First-of-its kind APR tool for hardware designs
- Novel **fault localization approach** suitable for hardware designs

Fault localization

Introducing: *CirFix*


Fault localization

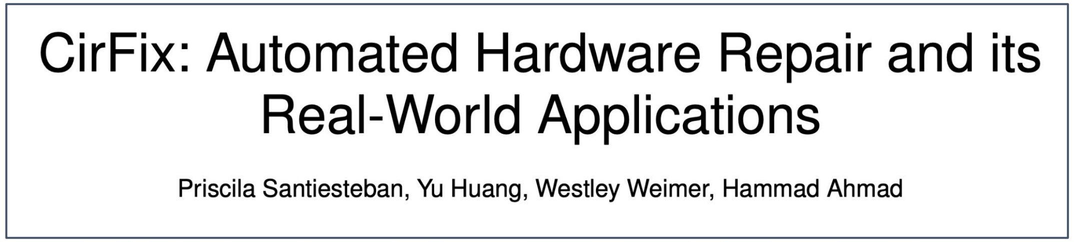
Patch

Validation

CirFix: A hardware-design focused automated repair algorithm

- First-of-its kind APR tool for hardware designs
- Novel **fault localization approach** suitable for hardware designs
- Novel **approach to guide the search for repairs** using the existing hardware design process

- Results published in *ASPLOS*'22 and *TSE*'23


CirFix: Automated Hardware Repair and its Real-World Applications

Priscila Santiesteban, Yu Huang, Westley Weimer, Hammad Ahmad

CirFix: Automatically Repairing Defects in Hardware Design Code

Hammad Ahmad
hammada@umich.edu
University of Michigan, Ann Arbor
Ann Arbor, Michigan, USA

Yu Huang
yhhy@umich.edu
University of Michigan, Ann Arbor
Ann Arbor, Michigan, USA

Westley Weimer
weimerw@umich.edu
University of Michigan, Ann Arbor
Ann Arbor, Michigan, USA

CirFix: Empirical Evaluation

“How many hardware defects can CirFix actually repair?”

- No public benchmarks available for Verilog defects (largely due to IP constraints)
- Constructed a **benchmark suite** of 32 different hardware defects to evaluate CirFix
    - 6 classroom-level designs and 5 larger, open-source designs
    - 19 “easy” defects and 13 “hard” defects
- Benchmark suite publicly available for future researchers to evaluate hardware repair approaches

CirFix: Empirical Evaluation

“How many hardware defects can CirFix actually repair?”

- Ran five resource-constrained, independent CirFix trials for each defect, stopping when a repair was found
- CirFix produced *high-quality* (i.e., correct upon manual inspection) repairs for 16/32 (**50%**) defects
- Repair rate comparable to strong results from software-based APR (e.g., GenProg

CirFix is effective at automatically repairing defects in hardware designs!

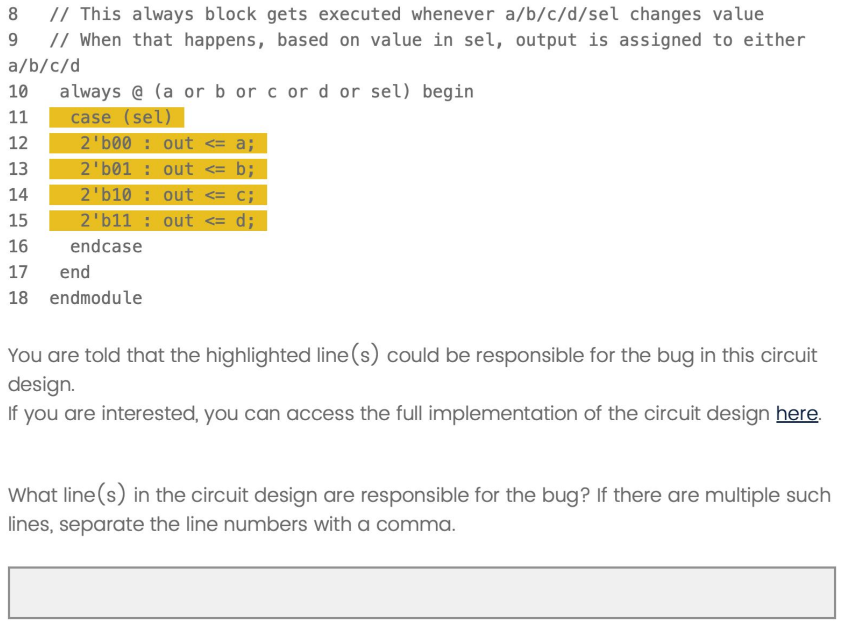
CirFix: Human Study Design

“How useful do developers find CirFix?”

- IRB-approved experimental protocol (HUM00199335)
- **41 participants** in the study (predominantly Michigan students)

- Participants asked to identify and fix defects from the CirFix benchmark, with or without **debugging hints**
  - Debugging hint: highlighting **lines of code** implicated by CirFix

- Participants also asked to rate the **accuracy** and **helpfulness** of presented hints
- Designer performance assessed by evaluating F-scores (F_1) and time taken to complete each debugging task



```
8   // This always block gets executed whenever a/b/c/d/sel changes value
9   // When that happens, based on value in sel, output is assigned to either
a/b/c/d
10   always @ (a or b or c or d or sel) begin
11    case (sel)
12     2'b00 : out <= a;
13     2'b01 : out <= b;
14     2'b10 : out <= c;
15     2'b11 : out <= d;
16    endcase
17   end
18  endmodule
```

You are told that the highlighted line(s) could be responsible for the bug in this circuit design.
If you are interested, you can access the full implementation of the circuit design here.

What line(s) in the circuit design are responsible for the bug? If there are multiple such lines, separate the line numbers with a comma.

CirFix: Human Study Results

“How useful do developers find CirFix?”

- **No statistically significant difference in time taken** to localize faults with debugging hints (*p* = 0.41, Student t-test)

- Trend for **participant debugging accuracy better** with debugging hints (F_1 = 0.67) vs. no hints (F_1 = 0.29)
  - Trend does not rise to statistical significance (*p* = 0.12)

- Debuggin **l and accurate** than tho **gnificant**)
  - Help

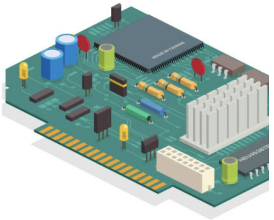
CirFix could be beneficial as a debugging assistant in a classroom context!

CirFix: Wrapping it Up

Can we build a state-of-the-art automated repair tool for hardware designs (i.e., digital logic), and use it as a debugging assistant for designers?

CirFix: Wrapping it Up

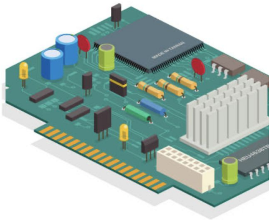
Can we build a state-of-the-art automated repair tool for hardware designs (i.e., digital logic), and use it as a debugging assistant for designers?



- CirFix can automatically repair hardware designs, achieving a repair rate comparable to that of software APR

CirFix: Wrapping it Up

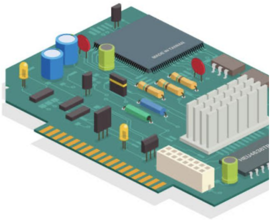
Can we build a state-of-the-art automated repair tool for hardware designs (i.e., digital logic), and use it as a debugging assistant for designers?



- CirFix can automatically repair hardware designs, achieving a repair rate comparable to that of software APR

- Programmers using CirFix as a debugging assistant
  - Rate the tool as significantly helpful for classroom-level designs
  - Show trends of improved debugging accuracy

Three Research Components



~~Using automated program repair for hardware as a debugging assistant for designers~~

Using eye-tracking to understand cognition for computer science formalisms

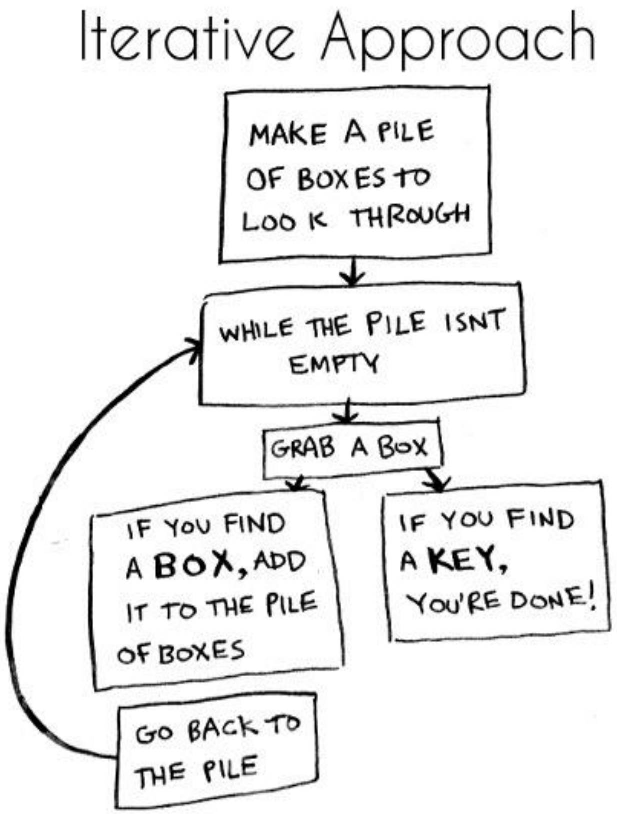
Using neurostimulation to investigate the relationship between spatial reasoning and programming

Eye-Tracking for Computer Science Formalisms

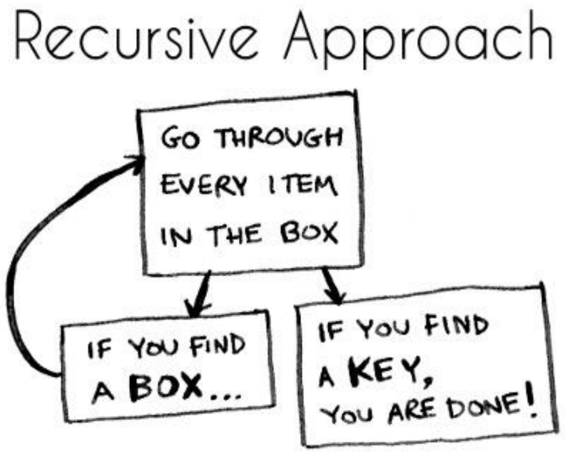
Can we use eye-tracking to investigate how students read and understand computer science formalisms (i.e., mathematical logic)?

Common Student Sentiment:

"I find **iterative reasoning** easier than **recursive reasoning** for algorithmic problem solving."



vs.

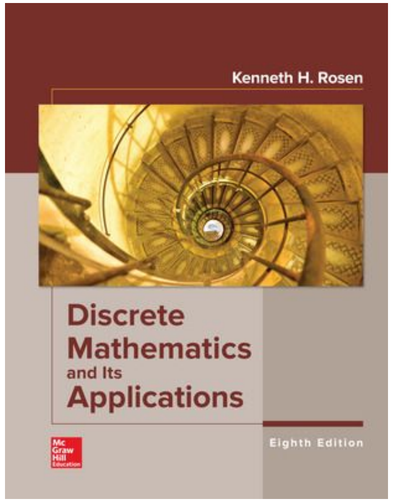


Formalism Comprehension

Students sometimes have a hard time with logical algorithmic reasoning (i.e., **mathematical logic**)

Many CS programs require majors to take several courses focusing on **formal reasoning** (e.g., discrete math, theory, algorithm analysis)

At Michigan: **EECS 203, 376, MATH 416**

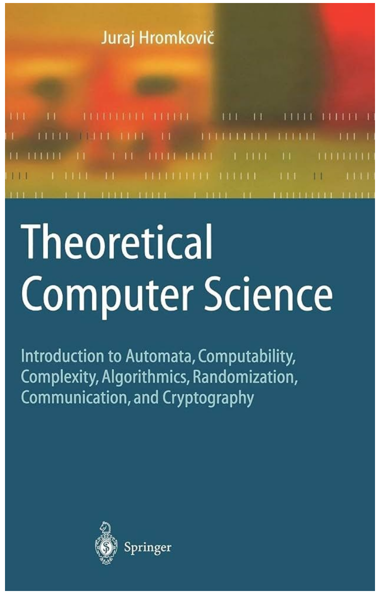


Kenneth H. Rosen

Discrete Mathematics and Its Applications

Eighth Edition

Mc Graw Hill Education



Juraj Hromkovič

Theoretical Computer Science

Introduction to Automata, Computability, Complexity, Algorithmics, Randomization, Communication, and Cryptography

Springer

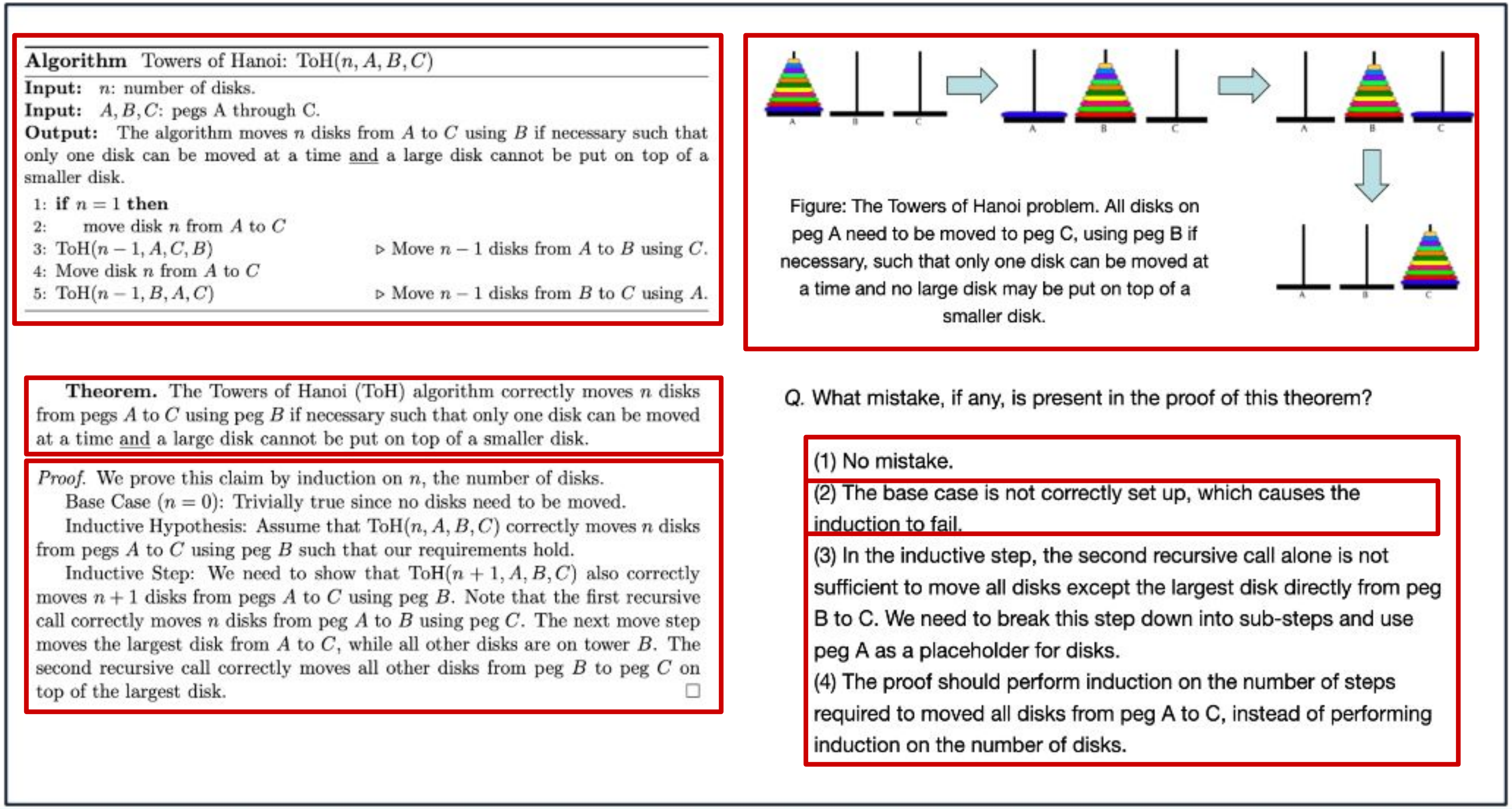
Formalism Comprehension

Formal reasoning is widely used to improve software quality and reliability!

and many, *many* more…

Are students learning and retaining effective strategies for reasoning about computer science formalisms?

“Formalism” Defined



Algorithm Towers of Hanoi: ToH(n, A, B, C)

Input: n: number of disks.

Input: A, B, C: pegs A through C.

Output: The algorithm moves n disks from A to C using B if necessary such that only one disk can be moved at a time and a large disk cannot be put on top of a smaller disk.

```
if $n = 1$ then
    move disk $n$ from $A$ to $C$
ToH($n - 1, A, C, B$)        ▷ Move $n - 1$ disks from $A$ to $B$ using $C$.
Move disk $n$ from $A$ to $C$
ToH($n - 1, B, A, C$)        ▷ Move $n - 1$ disks from $B$ to $C$ using $A$.
```

Figure: The Towers of Hanoi problem. All disks on peg A need to be moved to peg C, using peg B if necessary, such that only one disk can be moved at a time and no large disk may be put on top of a smaller disk.

Theorem. The Towers of Hanoi (ToH) algorithm correctly moves n disks from pegs A to C using peg B if necessary such that only one disk can be moved at a time and a large disk cannot be put on top of a smaller disk.

Proof. We prove this claim by induction on n, the number of disks.

Base Case ($n = 0$): Trivially true since no disks need to be moved.

Inductive Hypothesis: Assume that ToH(n, A, B, C) correctly moves n disks from pegs A to C using peg B such that our requirements hold.

Inductive Step: We need to show that ToH($n + 1, A, B, C$) also correctly moves $n + 1$ disks from pegs A to C using peg B. Note that the first recursive call correctly moves n disks from peg A to B using peg C. The next move step moves the largest disk from A to C, while all other disks are on tower B. The second recursive call correctly moves all other disks from peg B to peg C on top of the largest disk. ☐

Q. What mistake, if any, is present in the proof of this theorem?

(1) No mistake.

(2) The base case is not correctly set up, which causes the induction to fail.

(3) In the inductive step, the second recursive call alone is not sufficient to move all disks except the largest disk directly from peg B to C. We need to break this step down into sub-steps and use peg A as a placeholder for disks.

(4) The proof should perform induction on the number of steps required to moved all disks from peg A to C, instead of performing induction on the number of disks.

Enter: Eye-Tracking

- **Cheap** and **non-invasive** measure of **problem solving strategies**

- Approximates dynamics of **visual attention** (e.g., where we focus, and for how long)
- Serves as a proxy for **cognitive load** (i.e., strain on working memory) and **task difficulty**

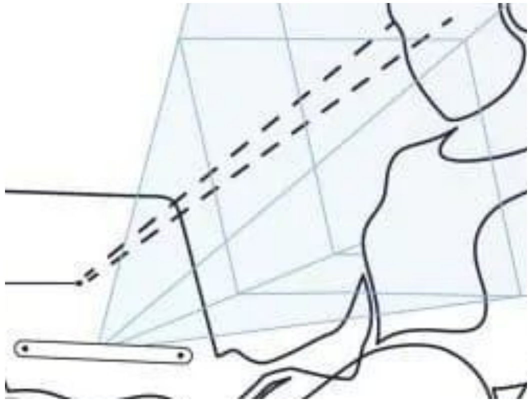
How Does Eye-Tracking Work?

An **eye-tracker** consists of cameras and projectors

How Does Eye-Tracking Work?

An **eye-tracker** consists of cameras and projectors

The projectors create a pattern of near-infrared light on the eyes

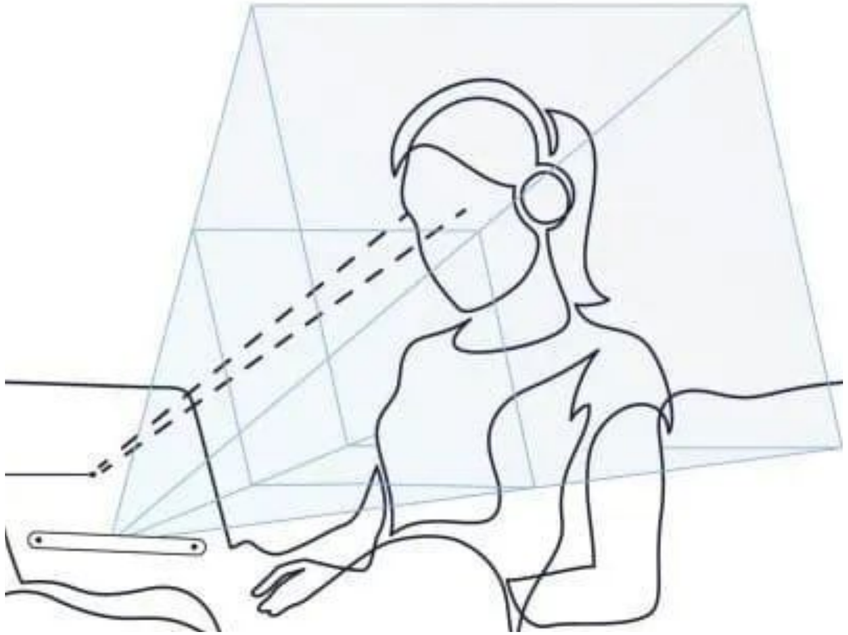


How Does Eye-Tracking Work?

An **eye-tracker** consists of cameras and projectors

The projectors create a pattern of near-infrared light on the eyes

The cameras take high-resolution images of the eyes and the pattern



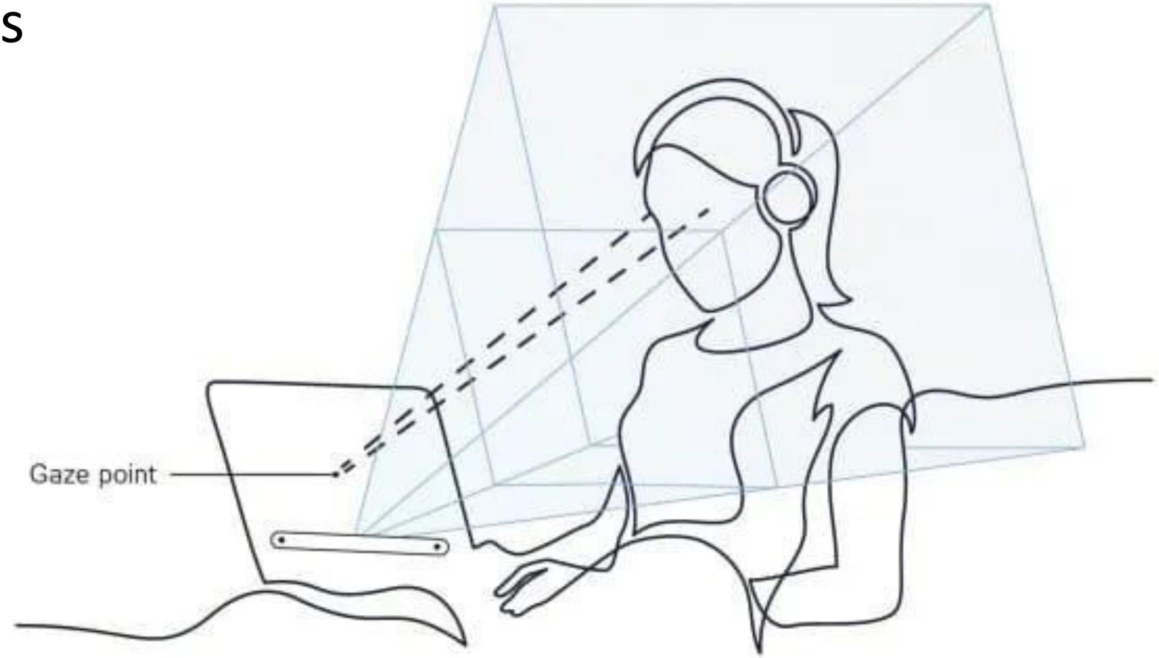
How Does Eye-Tracking Work?

An **eye-tracker** consists of cameras and projectors

The projectors create a pattern of near-infrared light on the eyes

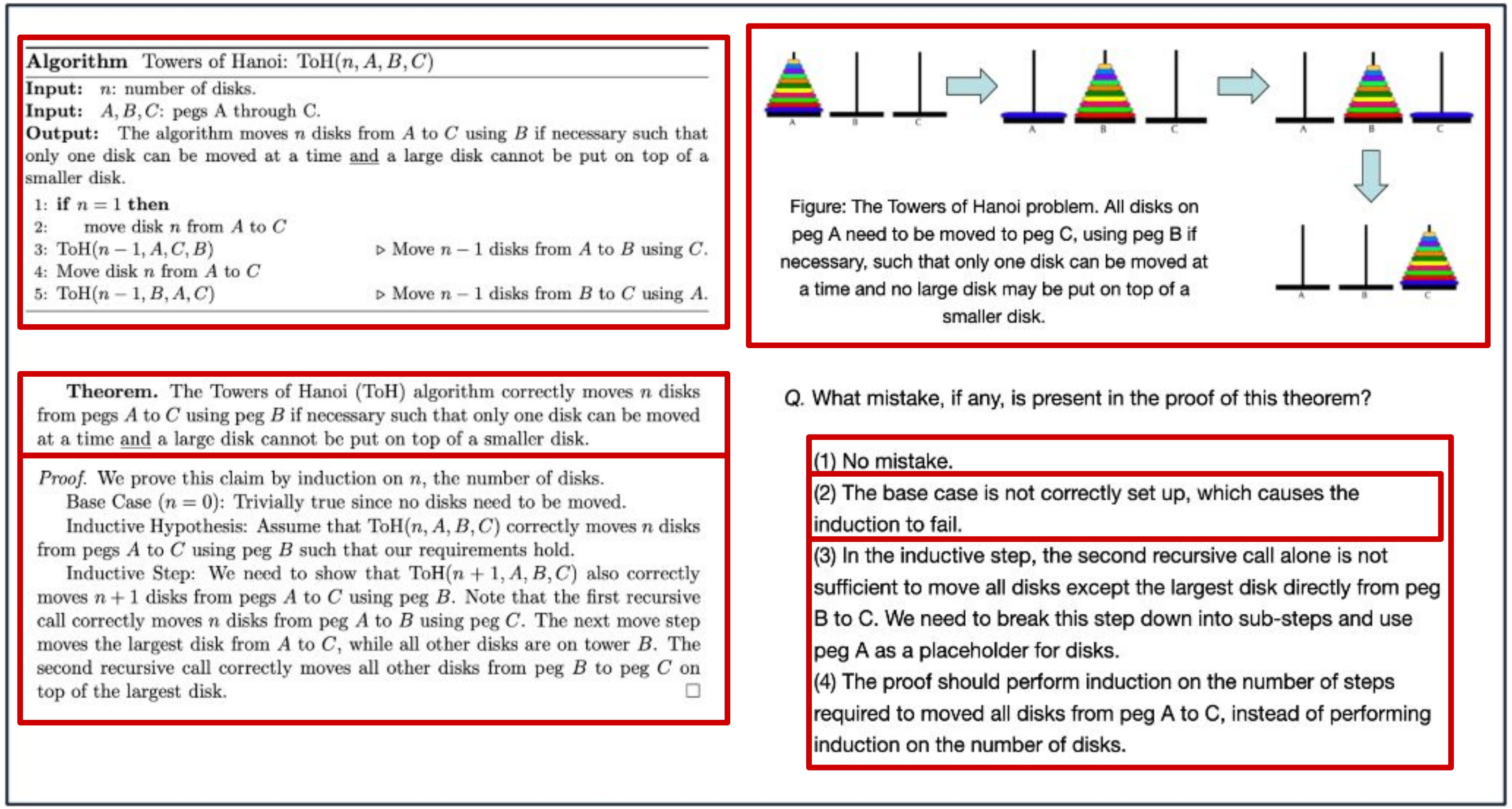
The cameras take high-resolution images of the eyes and the pattern

Machine learning, image processing, and mathematical algorithms are used to determine the eyes' position and "gaze point"



Formalism Comprehension: Some Eye-Tracking Terminology

Areas of Interest (AOIs)



Algorithm Towers of Hanoi: ToH(n, A, B, C)

Input: n: number of disks.
Input: A, B, C: pegs A through C.
Output: The algorithm moves n disks from A to C using B if necessary such that only one disk can be moved at a time and a large disk cannot be put on top of a smaller disk.

```
if $n = 1$ then
    move disk $n$ from $A$ to $C$
ToH$(n-1, A, C, B)$        ▷ Move $n-1$ disks from $A$ to $B$ using $C$.
Move disk $n$ from $A$ to $C$
ToH$(n-1, B, A, C)$        ▷ Move $n-1$ disks from $B$ to $C$ using $A$.
```

Figure: The Towers of Hanoi problem. All disks on peg A need to be moved to peg C, using peg B if necessary, such that only one disk can be moved at a time and no large disk may be put on top of a smaller disk.

Theorem. The Towers of Hanoi (ToH) algorithm correctly moves n disks from pegs A to C using peg B if necessary such that only one disk can be moved at a time and a large disk cannot be put on top of a smaller disk.

Proof. We prove this claim by induction on n, the number of disks.

Base Case $(n = 0)$: Trivially true since no disks need to be moved.

Inductive Hypothesis: Assume that ToH(n, A, B, C) correctly moves n disks from pegs A to C using peg B such that our requirements hold.

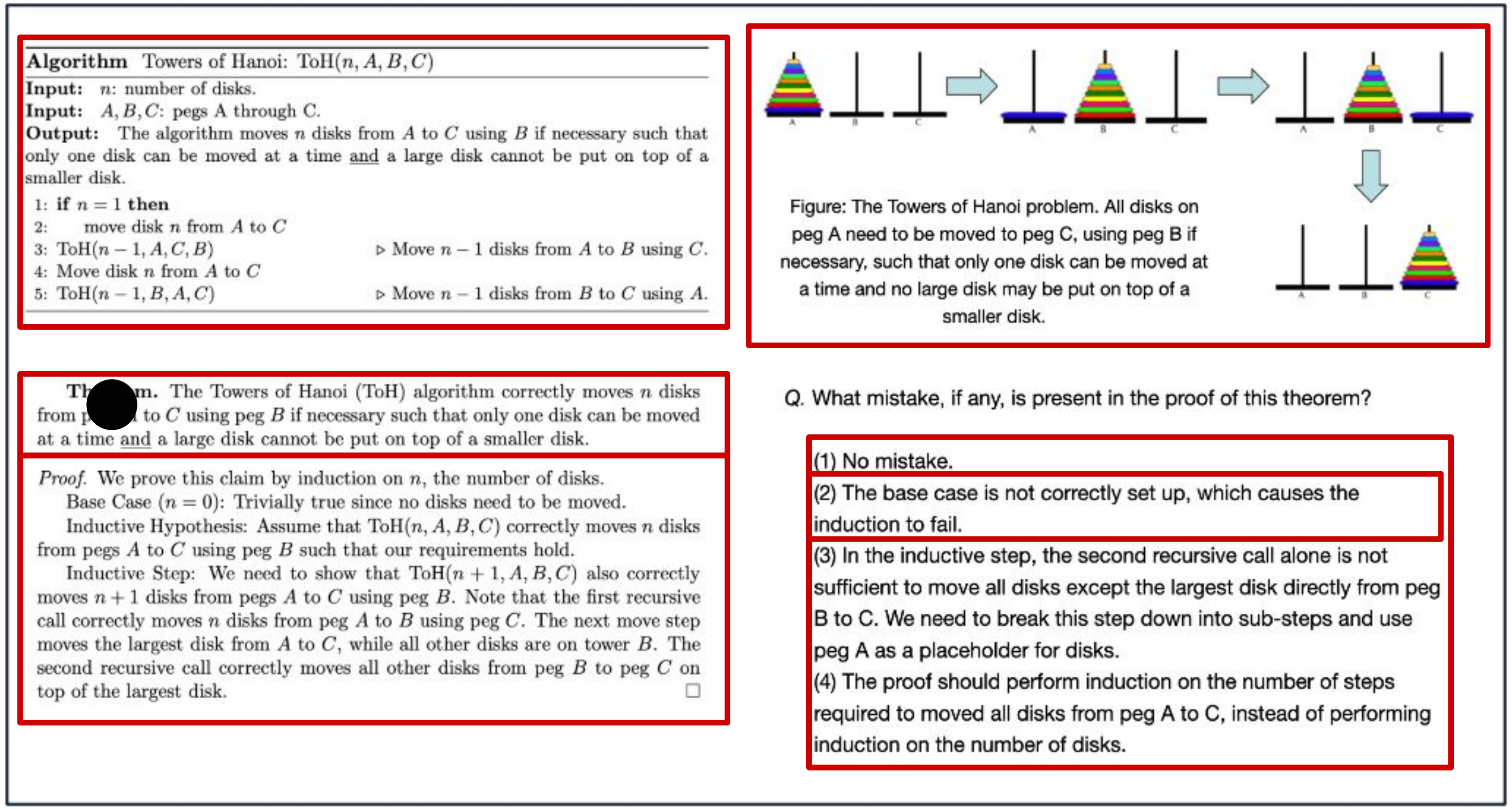
Inductive Step: We need to show that ToH$(n+1, A, B, C)$ also correctly moves $n+1$ disks from pegs A to C using peg B. Note that the first recursive call correctly moves n disks from peg A to B using peg C. The next move step moves the largest disk from A to C, while all other disks are on tower B. The second recursive call correctly moves all other disks from peg B to peg C on top of the largest disk. ☐

Q. What mistake, if any, is present in the proof of this theorem?

(1) No mistake.
(2) The base case is not correctly set up, which causes the induction to fail.
(3) In the inductive step, the second recursive call alone is not sufficient to move all disks except the largest disk directly from peg B to C. We need to break this step down into sub-steps and use peg A as a placeholder for disks.
(4) The proof should perform induction on the number of steps required to moved all disks from peg A to C, instead of performing induction on the number of disks.

Formalism Comprehension: Some Eye-Tracking Terminology

Fixation



Algorithm Towers of Hanoi: ToH(n, A, B, C)

Input: n: number of disks.

Input: A, B, C: pegs A through C.

Output: The algorithm moves n disks from A to C using B if necessary such that only one disk can be moved at a time and a large disk cannot be put on top of a smaller disk.

```
if $n = 1$ then
    move disk $n$ from $A$ to $C$
ToH$(n-1, A, C, B)$          ▷ Move $n-1$ disks from $A$ to $B$ using $C$.
Move disk $n$ from $A$ to $C$
ToH$(n-1, B, A, C)$          ▷ Move $n-1$ disks from $B$ to $C$ using $A$.
```

Figure: The Towers of Hanoi problem. All disks on peg A need to be moved to peg C, using peg B if necessary, such that only one disk can be moved at a time and no large disk may be put on top of a smaller disk.

Theorem. The Towers of Hanoi (ToH) algorithm correctly moves n disks from peg A to C using peg B if necessary such that only one disk can be moved at a time and a large disk cannot be put on top of a smaller disk.

Proof. We prove this claim by induction on n, the number of disks.

Base Case $(n = 0)$: Trivially true since no disks need to be moved.

Inductive Hypothesis: Assume that ToH(n, A, B, C) correctly moves n disks from pegs A to C using peg B such that our requirements hold.

Inductive Step: We need to show that ToH$(n+1, A, B, C)$ also correctly moves $n+1$ disks from pegs A to C using peg B. Note that the first recursive call correctly moves n disks from peg A to B using peg C. The next move step moves the largest disk from A to C, while all other disks are on tower B. The second recursive call correctly moves all other disks from peg B to peg C on top of the largest disk. ☐

Q. What mistake, if any, is present in the proof of this theorem?

(1) No mistake.

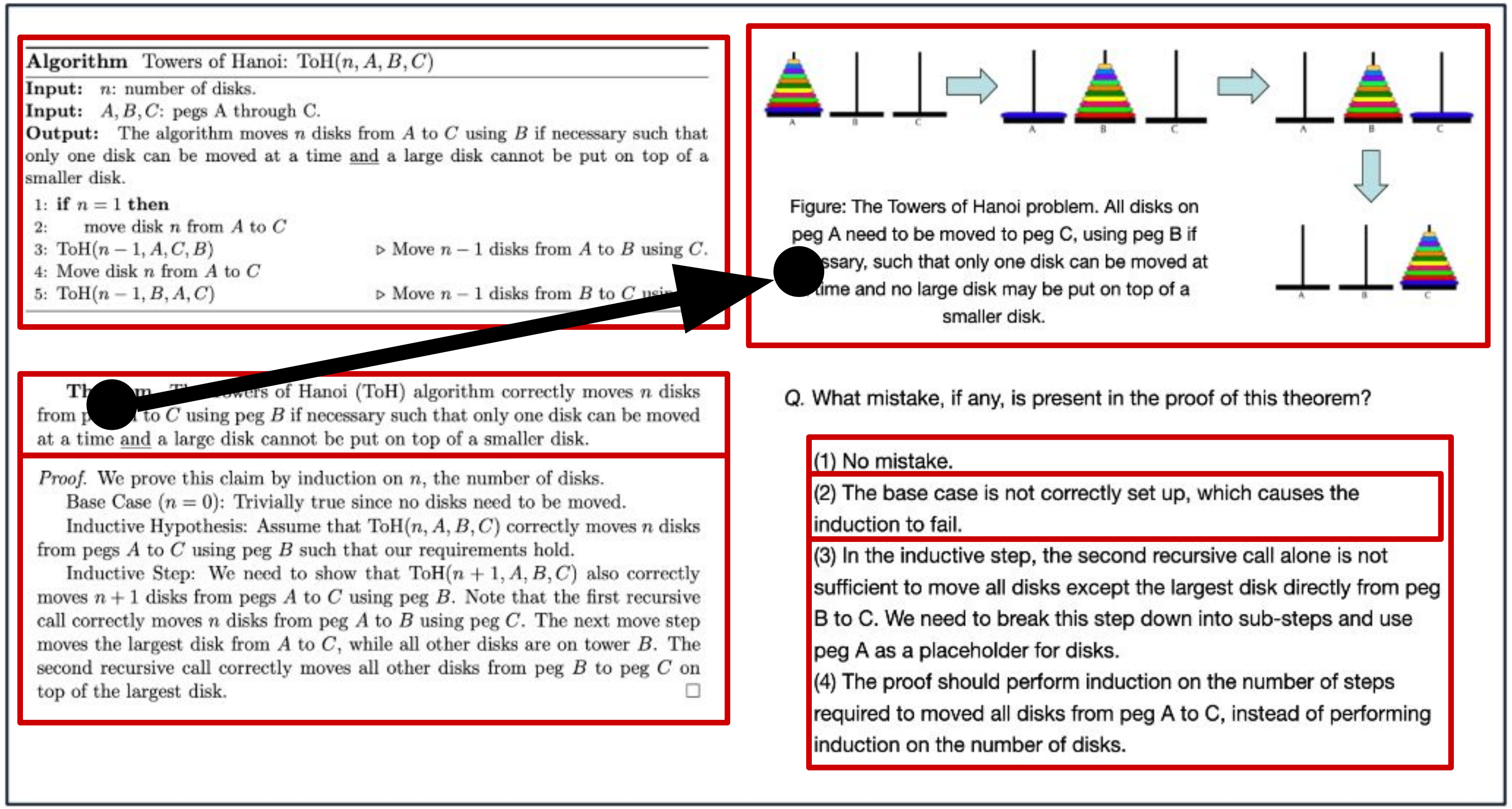
(2) The base case is not correctly set up, which causes the induction to fail.

(3) In the inductive step, the second recursive call alone is not sufficient to move all disks except the largest disk directly from peg B to C. We need to break this step down into sub-steps and use peg A as a placeholder for disks.

(4) The proof should perform induction on the number of steps required to moved all disks from peg A to C, instead of performing induction on the number of disks.

Formalism Comprehension: Some Eye-Tracking Terminology

Attention Switching



Algorithm Towers of Hanoi: ToH(n, A, B, C)

Input: n: number of disks.

Input: A, B, C: pegs A through C.

Output: The algorithm moves n disks from A to C using B if necessary such that only one disk can be moved at a time and a large disk cannot be put on top of a smaller disk.

```
if $n = 1$ then
    move disk $n$ from $A$ to $C$
ToH($n-1, A, C, B$)        ▷ Move $n-1$ disks from $A$ to $B$ using $C$.
Move disk $n$ from $A$ to $C$
ToH($n-1, B, A, C$)        ▷ Move $n-1$ disks from $B$ to $C$ usi
```

Figure: The Towers of Hanoi problem. All disks on peg A need to be moved to peg C, using peg B if ssary, such that only one disk can be moved at ime and no large disk may be put on top of a smaller disk.

Th **m.** Th wers of Hanoi (ToH) algorithm correctly moves n disks from p to C using peg B if necessary such that only one disk can be moved at a time and a large disk cannot be put on top of a smaller disk.

Proof. We prove this claim by induction on n, the number of disks.

Base Case $(n = 0)$: Trivially true since no disks need to be moved.

Inductive Hypothesis: Assume that ToH(n, A, B, C) correctly moves n disks from pegs A to C using peg B such that our requirements hold.

Inductive Step: We need to show that ToH$(n+1, A, B, C)$ also correctly moves $n+1$ disks from pegs A to C using peg B. Note that the first recursive call correctly moves n disks from peg A to B using peg C. The next move step moves the largest disk from A to C, while all other disks are on tower B. The second recursive call correctly moves all other disks from peg B to peg C on top of the largest disk. ☐

Q. What mistake, if any, is present in the proof of this theorem?

(1) No mistake.

(2) The base case is not correctly set up, which causes the induction to fail.

(3) In the inductive step, the second recursive call alone is not sufficient to move all disks except the largest disk directly from peg B to C. We need to break this step down into sub-steps and use peg A as a placeholder for disks.

(4) The proof should perform induction on the number of steps required to moved all disks from peg A to C, instead of performing induction on the number of disks.

Formalism Comprehension: Human Study Design

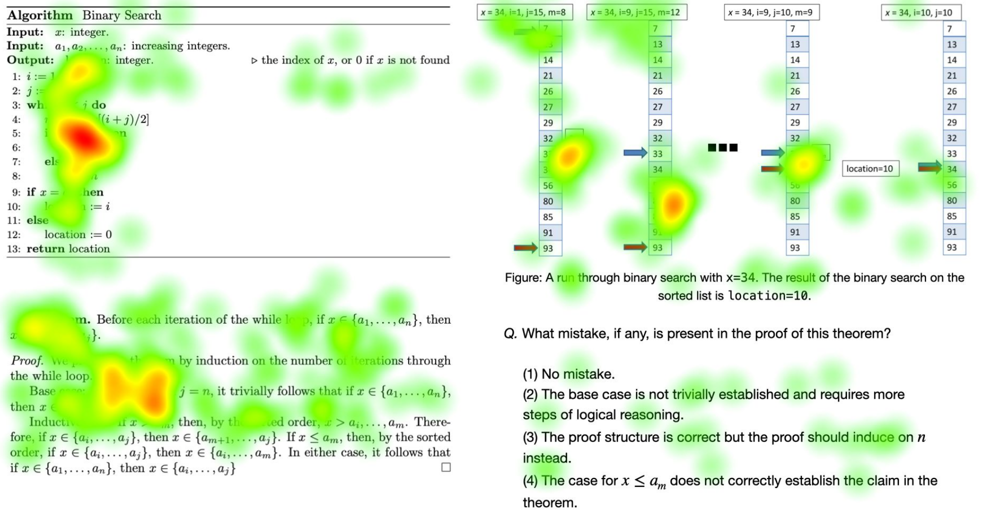


Figure: A run through binary search with x=34. The result of the binary search on the sorted list is `location=10`.

Q. What mistake, if any, is present in the proof of this theorem?

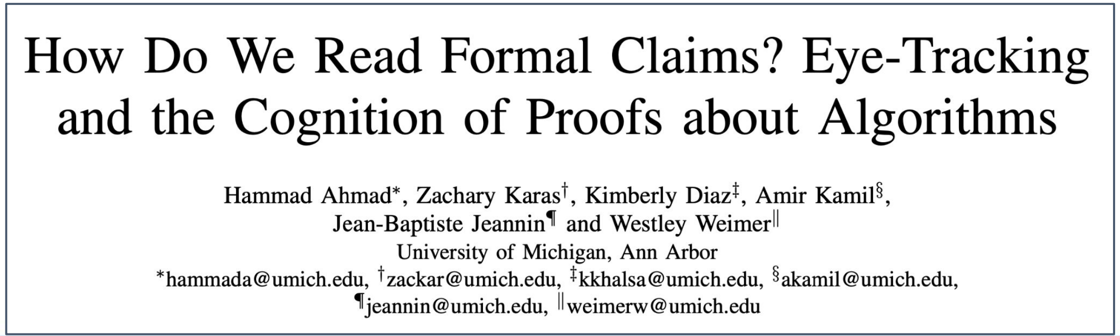
(1) No mistake.
(2) The base case is not trivially established and requires more steps of logical reasoning.
(3) The proof structure is correct but the proof should induce on n instead.
(4) The case for $x \leq a_m$ does not correctly establish the claim in the theorem.

“How do students find mistakes in proofs?”

- IRB-approved experimental protocol (HUM00204278)
- **34 participants** in the study (predominantly Michigan students)

- Participants shown a series of **algorithmic proofs** from a textbook, each with an associated figure and possible mistake
- Participants asked to identify the presence of mistakes in each proof
- **Eye-tracking** used to assess comprehension strategy

- Results published in *ICSE*’23



How Do We Read Formal Claims? Eye-Tracking and the Cognition of Proofs about Algorithms

Hammad Ahmad*, Zachary Karas†, Kimberly Diaz‡, Amir Kamil§,
Jean-Baptiste Jeannin¶ and Westley Weimer‖
University of Michigan, Ann Arbor
*hammada@umich.edu, †zackar@umich.edu, ‡kkhalsa@umich.edu, §akamil@umich.edu,
¶jeannin@umich.edu, ‖weimerw@umich.edu

Formalism Comprehension: Human Study Results

“Is more preparation correlated with better efficacy at finding mistakes in proofs?”

- **No statistically significant difference in response times and accuracies** between more and less prepared participants
  - “*More prepared*”: Have taken more than 4 courses covering CS formalisms and pass a pre-screening test (16/34 participants)
  - No correlation between formalism course count and response accuracy (Pearson’s $r = 0.036$, $p = 0.84$)

Taking more classes prepares students to read the proof and answer choices more thoroughly, but that may not be enough!

Formalism Comprehension: Human Study Results

“Are students able to assess their performances for proof reading tasks?”

- **No evidence of correlations** between
    - Response accuracy and self-reported expertise with formalisms (Kendall’s τ test, $\tau = 0.21$, $p = 0.18$)
    - Response accuracy and self-perceived task difficulty ($\tau = 0.14$, $p = 0.35$)
    - Response accuracy and self-perceived proof readability ($\tau = -0.14$, $p = 0.32$)

Student self-reports of their experience or familiarity with formalism comprehension tasks may not be reliable!

Formalism Comprehension: Human Study Results

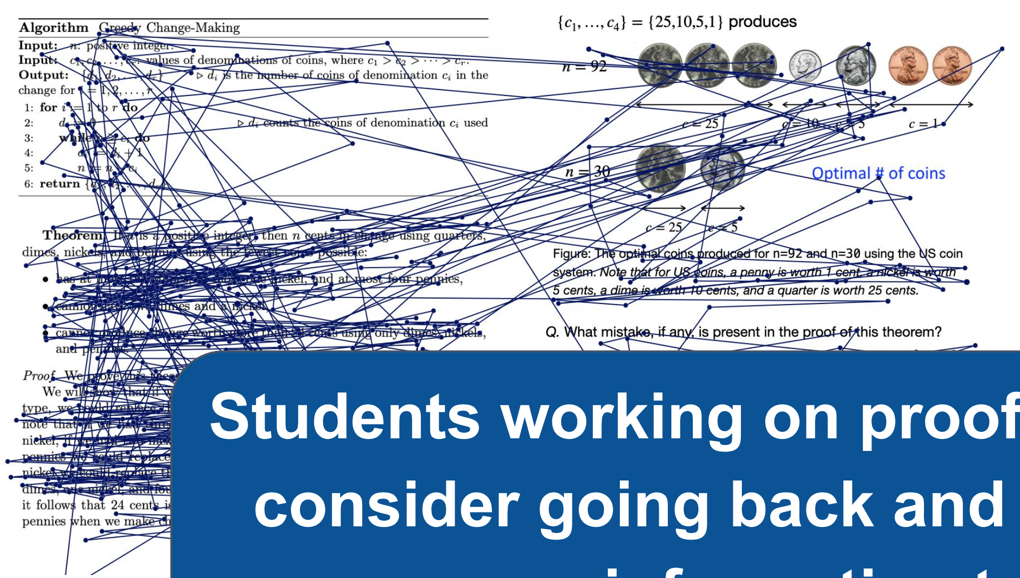
“What sets apart higher-performing participants from lower-performing ones?”

- Ability to **spot mistakes in proofs for recursive algorithms** (p = 0.006, **statistically significant**)
- Ability to **spot mistakes in inductive proofs** (p = 0.01, **statistically significant**)
- Iterative algorithms, direct proofs, and proofs by contradiction do not pose as many challenges in a mistake-finding context

Students struggling with proof comprehension may benefit from practicing inductive reasoning and recursion!

Formalism Comprehension: Human Study Results

- Higher-performing participants **display more attention switching behavior**, i.e., frequently go back and forth between presented information ($p = 0.002$, **statistically significant**)



Algorithm Greedy Change-Making

```
Input: $n$: positive integer.
Input: $c_1, c_2, \ldots, c_r$: values of denominations of coins, where $c_1 > c_2 > \cdots > c_r$.
Output: $\{d_1, d_2, \ldots, d_r\}$    ▷ $d_i$ is the number of coins of denomination $c_i$ in the change for $i = 1, 2, \ldots, r$
for $i := 1$ to $r$ do
    $d_i := 0$    ▷ $d_i$ counts the coins of denomination $c_i$ used
    while $n \geq c_i$ do
        $d_i := d_i + 1$
        $n := n - c_i$
return $\{d_1, d_2, \ldots, d_r\}$
```

Theorem. If n is a positive integer, then n cents in change using quarters, dimes, nickels, and pennies using the fewest coins possible:

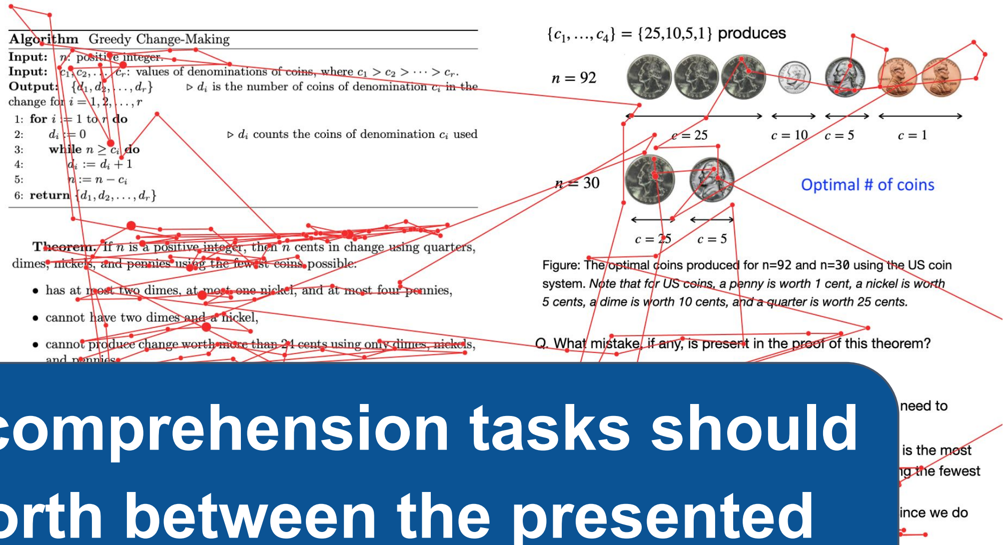
- has at most two dimes, at most one nickel, and at most four pennies,
- cannot have two dimes and a nickel,
- cannot produce change worth more than 24 cents using only dimes, nickels, and pennies.

Proof. We prove this
We will show that if w
type, we could replace th
note that if we had
nickel, if we had
pennies we could replace
nickel we could replace th
dimes, one nickel, and fou
it follows that 24 cents is
pennies when we make ch

$\{c_1, \ldots, c_4\} = \{25{,}10{,}5{,}1\}$ produces

Figure: The optimal coins produced for n=92 and n=30 using the US coin system. *Note that for US coins, a penny is worth 1 cent, a nickel is worth 5 cents, a dime is worth 10 cents, and a quarter is worth 25 cents.*

Q. What mistake, if any, is present in the proof of this theorem?



need to
is the most
g the fewest
ince we do

Students working on proof comprehension tasks should consider going back and forth between the presented information to let it assimilate!

Formalism Comprehension: Wrapping it Up

Can we use eye-tracking to investigate how students read and understand computer science formalisms (i.e., mathematical logic)?

Formalism Comprehension: Wrapping it Up

Can we use eye-tracking to investigate how students read and understand computer science formalisms (i.e., mathematical logic)?

- Incoming preparation and student self-reports are not accurate predictors of success with formalism comprehension

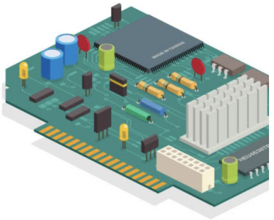
Formalism Comprehension: Wrapping it Up

Can we use eye-tracking to investigate how students read and understand computer science formalisms (i.e., mathematical logic)?

- Incoming preparation and student self-reports are not accurate predictors of success with formalism comprehension

- Higher-performing students
  - Are more effective at inductive and recursive reasoning
  - Display more attention-switching behaviors

Three Research Components



~~Using automated program repair for hardware as a debugging assistant for designers~~

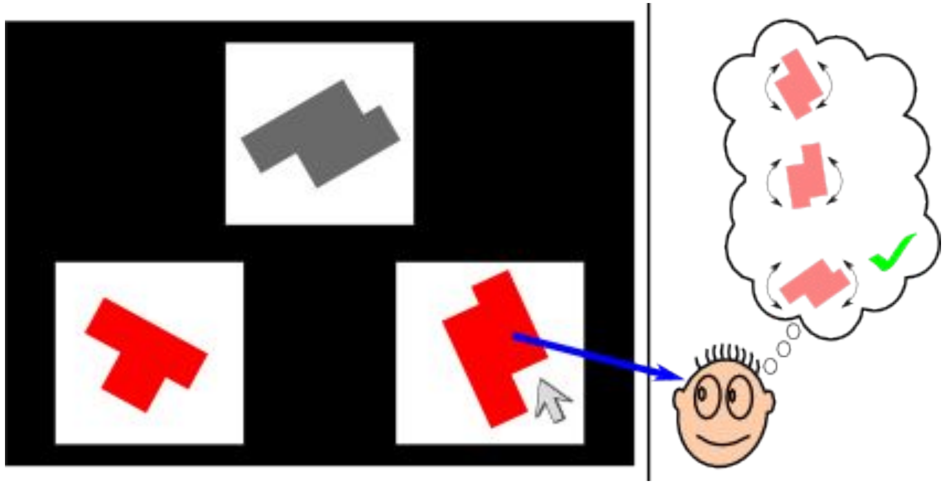
~~Using eye-tracking to understand cognition for computer science formalisms~~

Using neurostimulation to investigate the relationship between spatial reasoning and programming

Neurostimulation and Programming

Can we use neurostimulation to investigate brain activity for coding tasks (i.e., programming logic)?

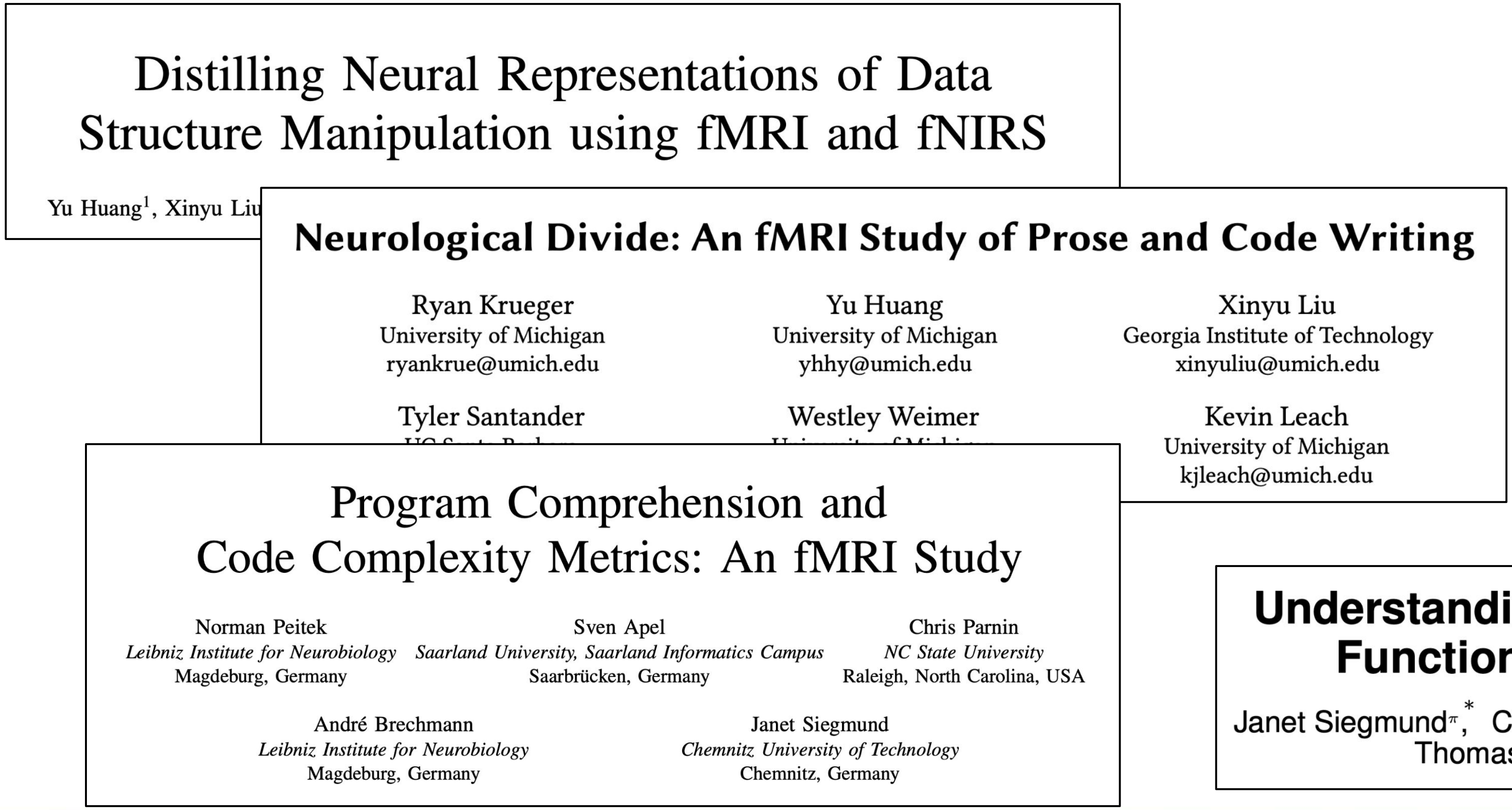
How is our brain activity for *programming* related that for *mentally rotating and manipulating objects*?



≟

Programming and Spatial Reasoning

Brain activity for **spatial reasoning** correlates with that for **programming** tasks



Distilling Neural Representations of Data Structure Manipulation using fMRI and fNIRS

Yu Huang[1], Xinyu Liu

Neurological Divide: An fMRI Study of Prose and Code Writing

Ryan Krueger
University of Michigan
ryankrue@umich.edu

Yu Huang
University of Michigan
yhhy@umich.edu

Xinyu Liu
Georgia Institute of Technology
xinyuliu@umich.edu

Tyler Santander

Westley Weimer

Kevin Leach
University of Michigan
kjleach@umich.edu

Program Comprehension and Code Complexity Metrics: An fMRI Study

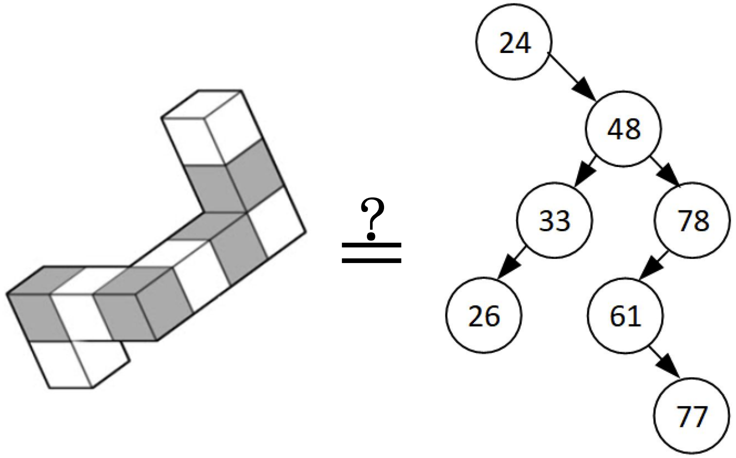
Norman Peitek
Leibniz Institute for Neurobiology
Magdeburg, Germany

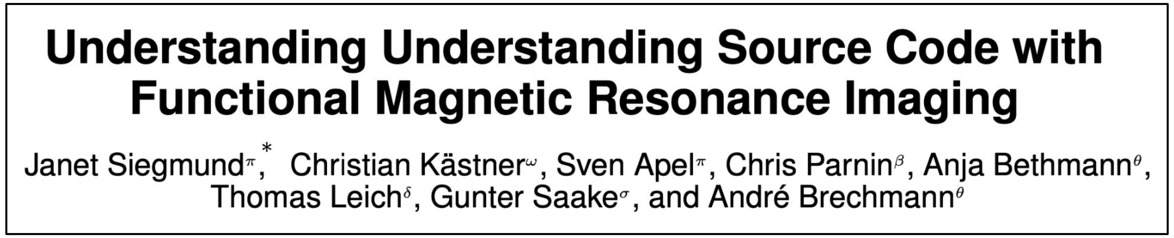
Sven Apel
Saarland University, Saarland Informatics Campus
Saarbrücken, Germany

Chris Parnin
NC State University
Raleigh, North Carolina, USA

André Brechmann
Leibniz Institute for Neurobiology
Magdeburg, Germany

Janet Siegmund
Chemnitz University of Technology
Chemnitz, Germany



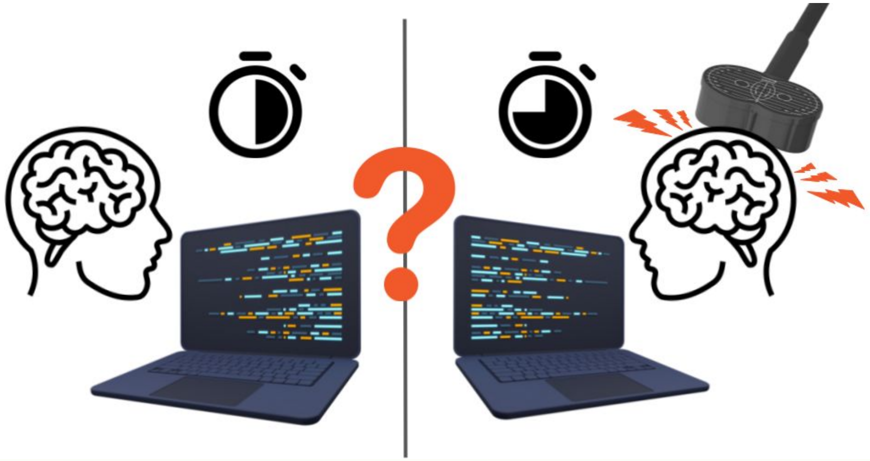


Understanding Understanding Source Code with Functional Magnetic Resonance Imaging

Janet Siegmund[π,*], Christian Kästner[ω], Sven Apel[π], Chris Parnin[β], Anja Bethmann[θ],
Thomas Leich[δ], Gunter Saake[σ], and André Brechmann[θ]

Programming and Spatial Reasoning

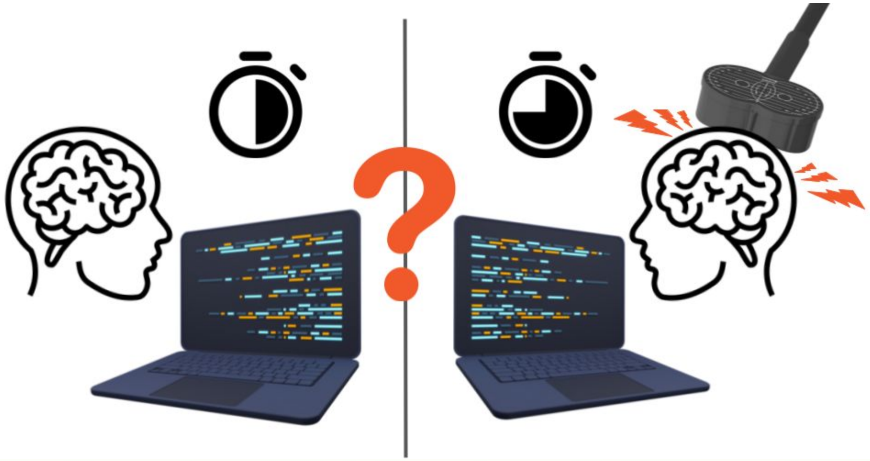
Is brain activity for spatial reasoning causally related to that for programming tasks?



Programming and Spatial Reasoning

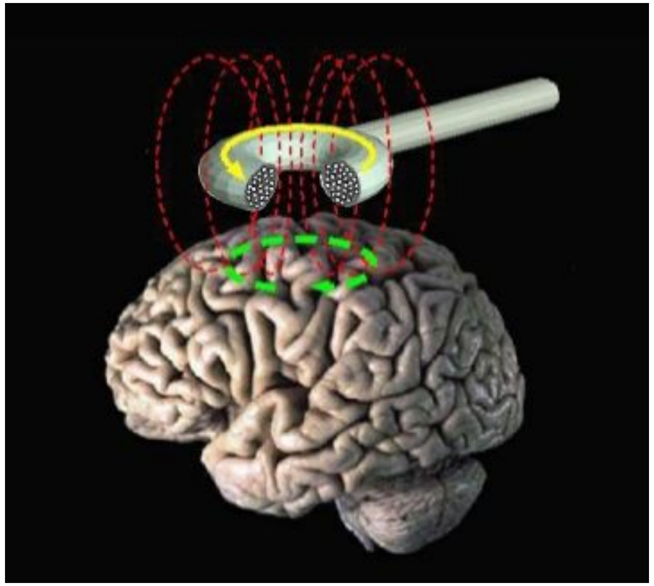
Is brain activity for spatial reasoning causally related to that for programming tasks?

Should we be training people to mentally rotate 3D objects to get better at programming?



Enter: Transcranial Magnetic Stimulation

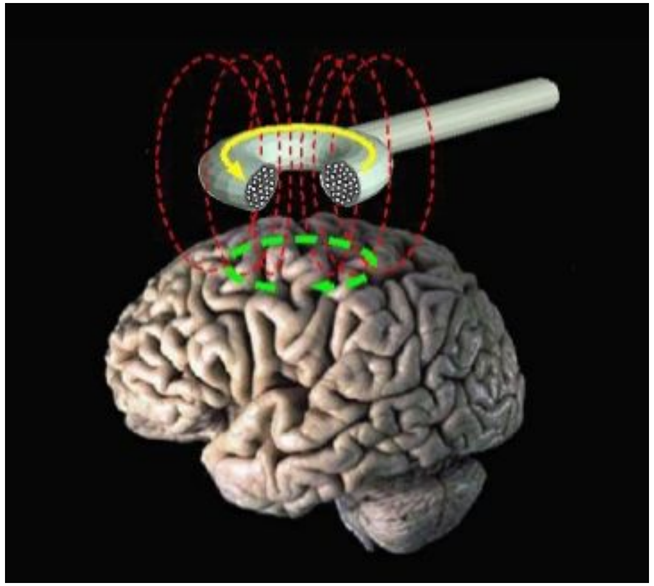
- **Safe** and **non-invasive**
- **Clinically used** as a treatment for depression, smoking cessation, OCD, etc.
- **Well-established** research tool



Enter: Transcranial Magnetic Stimulation

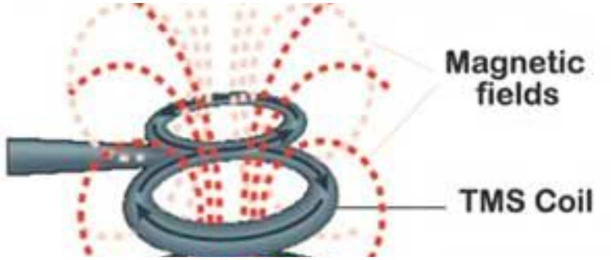
- **Safe** and **non-invasive**
- **Clinically used** as a treatment for depression, smoking cessation, OCD, etc.
- **Well-established** research tool

- Time-efficient way to investigate **causal relationships** in brain activity (e.g., compared to longitudinal studies over the course of weeks, months, or even years!)



How does TMS work?

TMS **pulses** produce a **magnetic field** around the TMS coil

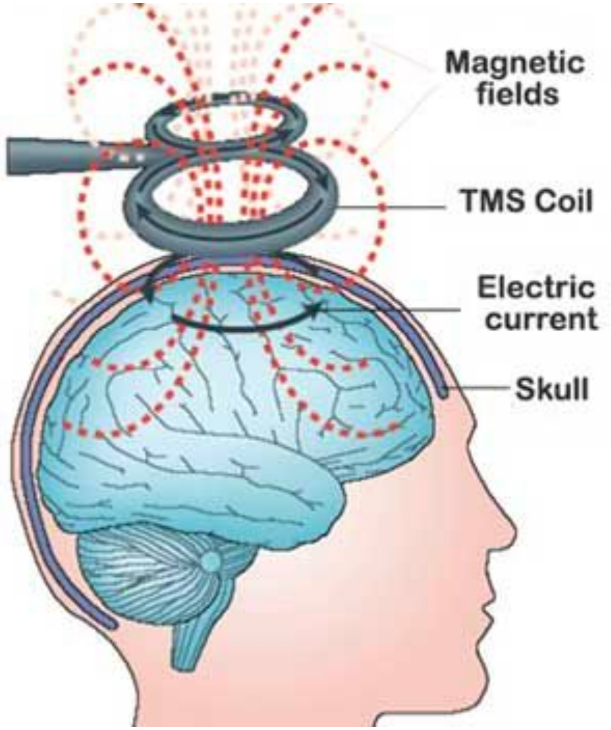


How does TMS work?

TMS **pulses** produce a **magnetic field** around the TMS coil

The magnetic field **induces a current** in the neurons of the brain region of interest

The induced current **excites** or **inhibits** brain activity in the region

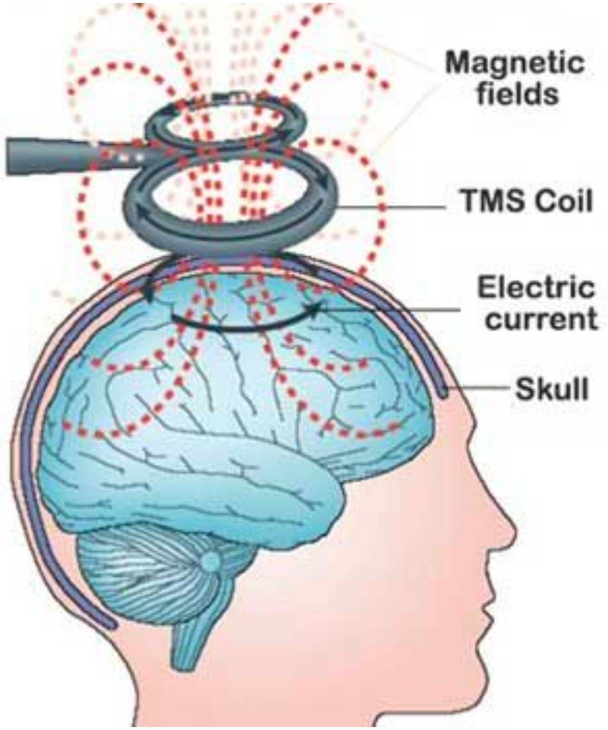


How does TMS work?

TMS **pulses** produce a **magnetic field** around the TMS coil

The magnetic field **induces a current** in the neurons of the brain region of interest

The induced current **excites** or **inhibits** brain activity in the region



By altering activity in certain brain regions, we can investigate the causal involvement of the regions for certain tasks!

TMS for Programming: Human Study Design

- IRBMED-approved experimental protocol (HUM00216195)
- **16 participants** in the study (Michigan students and industry developers)

- Participant brain scans collected through **functional magnetic resonance imaging** (fMRI)

TMS for Programming: Human Study Design

- IRBMED-approved experimental protocol (HUM00216195)
- **16 participants** in the study (Michigan students and industry developers)

- Participant brain scans collected through **functional magnetic resonance imaging** (fMRI)

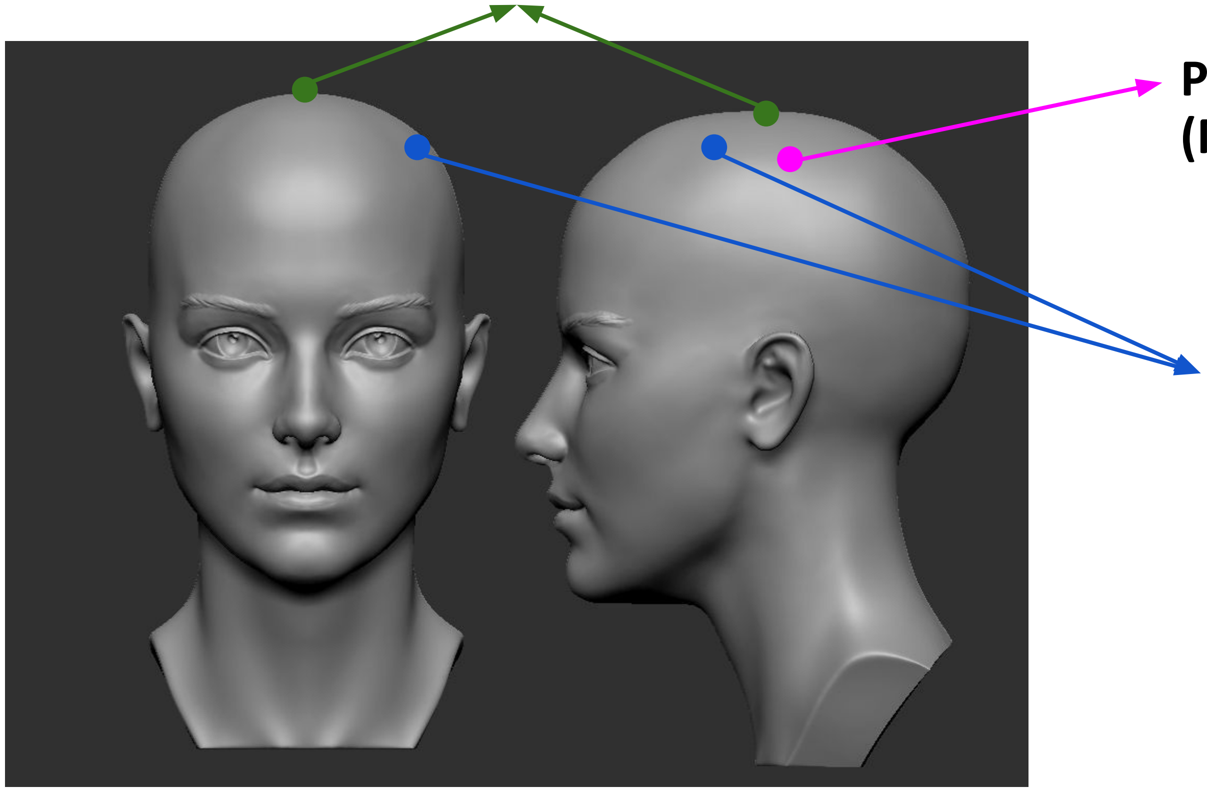
- Established **anatomical landmark-based localization approaches** used to identify brain regions of interest

TMS for Programming: Human Study Design

- Participants attend 2-4 TMS sessions (up to three treatment sessions, one control session; each on a different day)
  - Treatment: **supplementary motor area** (SMA) or **primary motor cortex** (M1), both responsible for motor actions and associated with spatial reasoning
  - Control: **cranial vertex** region, not associated with spatial reasoning

- 40 seconds of **neurostimulation** followed by 30 minutes of tasks on a regular computer
  - 3 pulses of stimulation at 50 Hz, repeated every 200ms, for a total of 600 pulses

TMS for Programming: Brain Regions



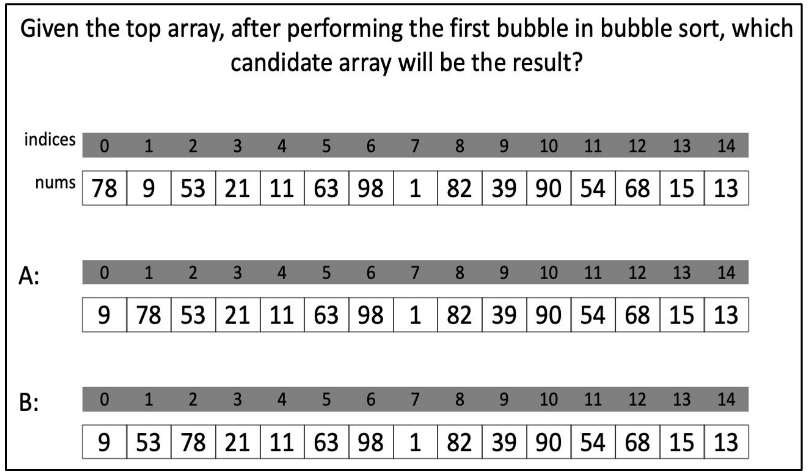
Cranial Vertex

Primary Motor Cortex (M1)

Supplementary Motor Area (SMA)

TMS for Programming: Human Study Design

- “Tasks”:
    - Data structure manipulation (e.g., sorting arrays, rotating trees)



Given the top array, after performing the first bubble in bubble sort, which candidate array will be the result?

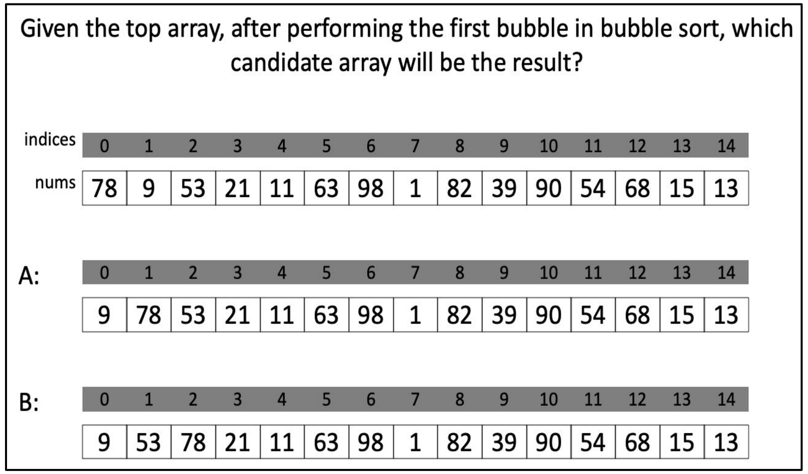
indices	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
nums	78	9	53	21	11	63	98	1	82	39	90	54	68	15	13

A:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	9	78	53	21	11	63	98	1	82	39	90	54	68	15	13

B:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	9	53	78	21	11	63	98	1	82	39	90	54	68	15	13

TMS for Programming:
Human Study Design

- “Tasks”:
  - Data structure manipulation (e.g., sorting arrays, rotating trees)
  - Mental rotation of 3D objects



Given the top array, after performing the first bubble in bubble sort, which candidate array will be the result?

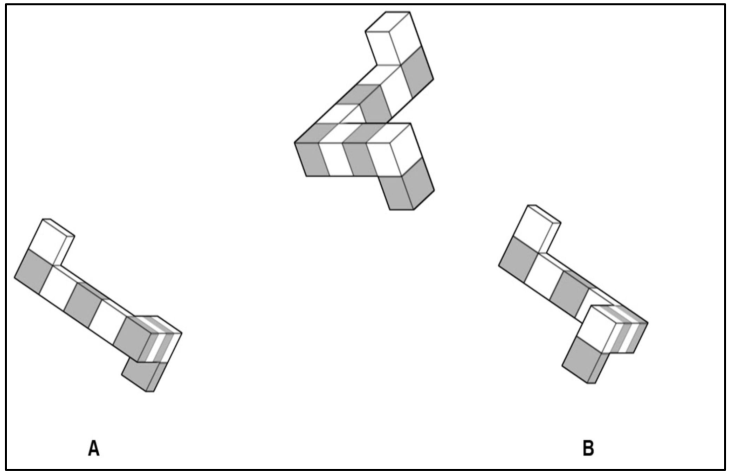
indices	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
nums	78	9	53	21	11	63	98	1	82	39	90	54	68	15	13

A:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
9	78	53	21	11	63	98	1	82	39	90	54	68	15	13

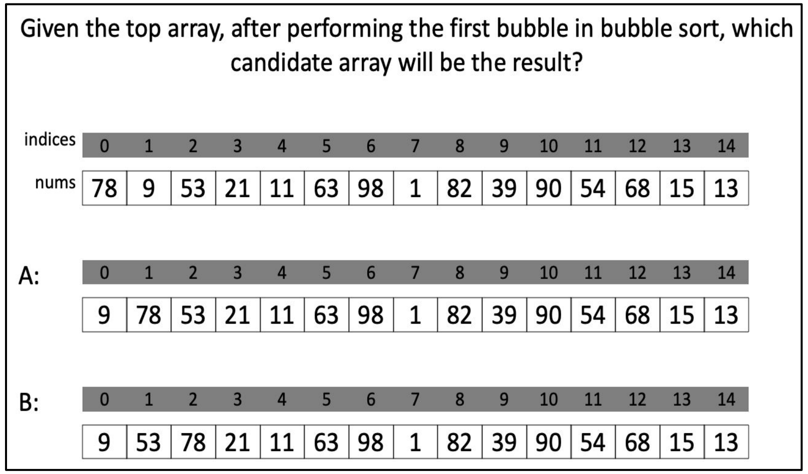
B:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
9	53	78	21	11	63	98	1	82	39	90	54	68	15	13



TMS for Programming: Human Study Design

- “Tasks”:
    - Data structure manipulation (e.g., sorting arrays, rotating trees)
    - Mental rotation of 3D objects
    - Code comprehension (e.g., tracing through code)



Given the top array, after performing the first bubble in bubble sort, which candidate array will be the result?

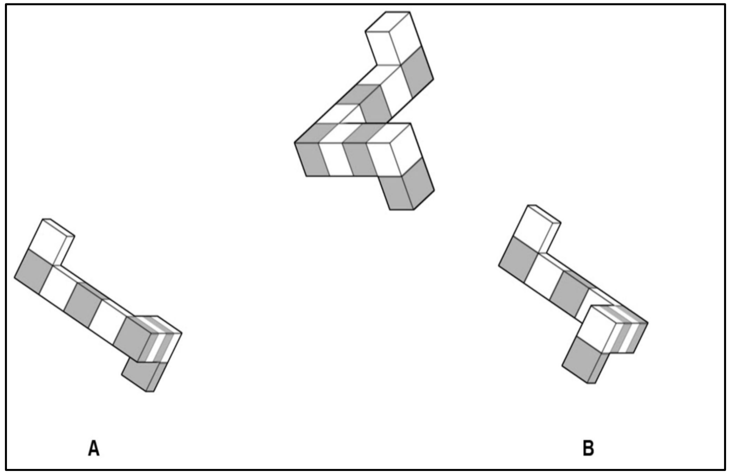
indices	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
nums	78	9	53	21	11	63	98	1	82	39	90	54	68	15	13

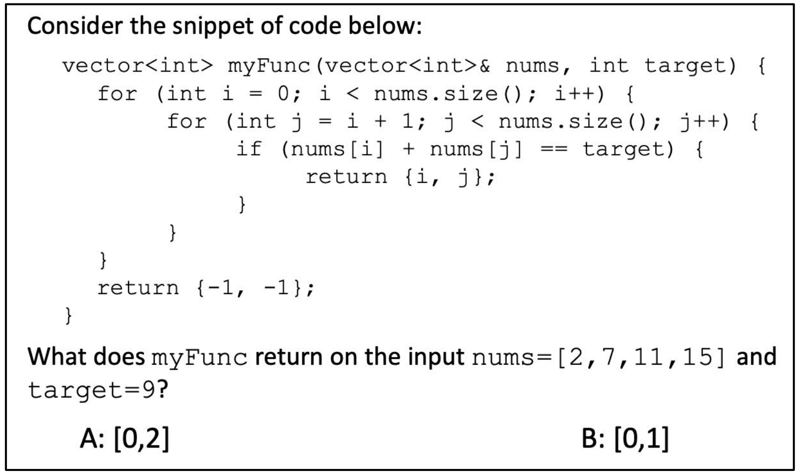
A:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
9	78	53	21	11	63	98	1	82	39	90	54	68	15	13

B:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
9	53	78	21	11	63	98	1	82	39	90	54	68	15	13





Consider the snippet of code below:

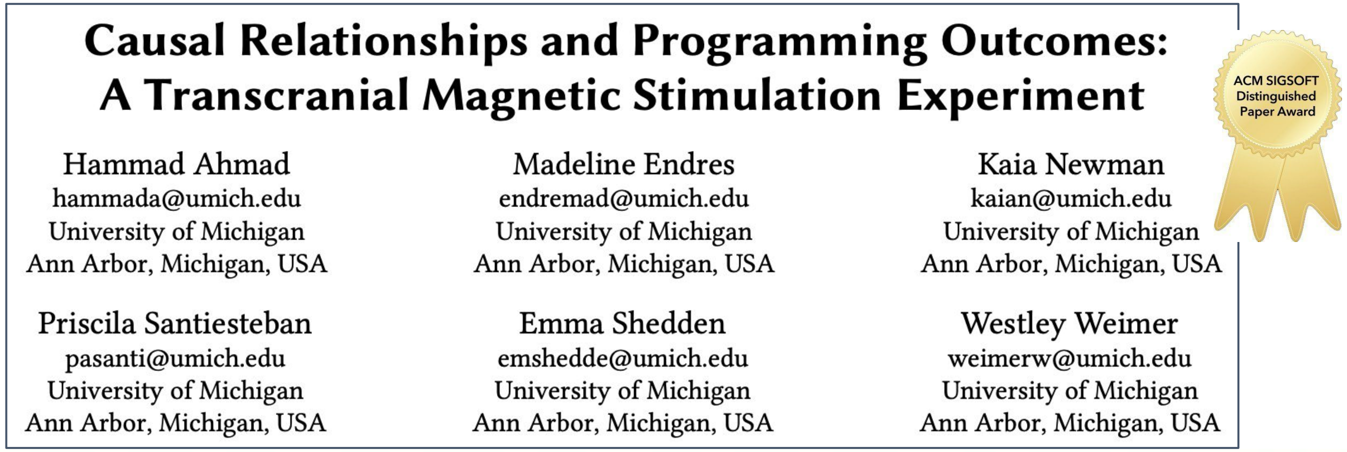
```
vector<int> myFunc(vector<int>& nums, int target) {
   for (int i = 0; i < nums.size(); i++) {
         for (int j = i + 1; j < nums.size(); j++) {
               if (nums[i] + nums[j] == target) {
                     return {i, j};
               }
         }
   }
   return {-1, -1};
}
```

What does `myFunc` return on the input `nums=[2,7,11,15]` and `target=9`?

A: [0,2]　　B: [0,1]

TMS for Programming: Human Study Design

- “Tasks”:
    - Data structure manipulation (e.g., sorting arrays, rotating trees)
    - Mental rotation of 3D objects
    - Code comprehension (e.g., tracing through code)
- Results published in *ICSE*’24 (with an ACM Distinguished Paper Award)



Causal Relationships and Programming Outcomes: A Transcranial Magnetic Stimulation Experiment

Hammad Ahmad
hammada@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

Madeline Endres
endremad@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

Kaia Newman
kaian@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

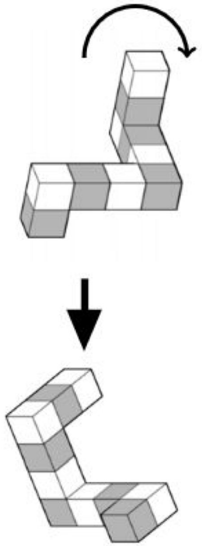
Priscila Santiesteban
pasanti@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

Emma Shedden
emshedde@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

Westley Weimer
weimerw@umich.edu
University of Michigan
Ann Arbor, Michigan, USA

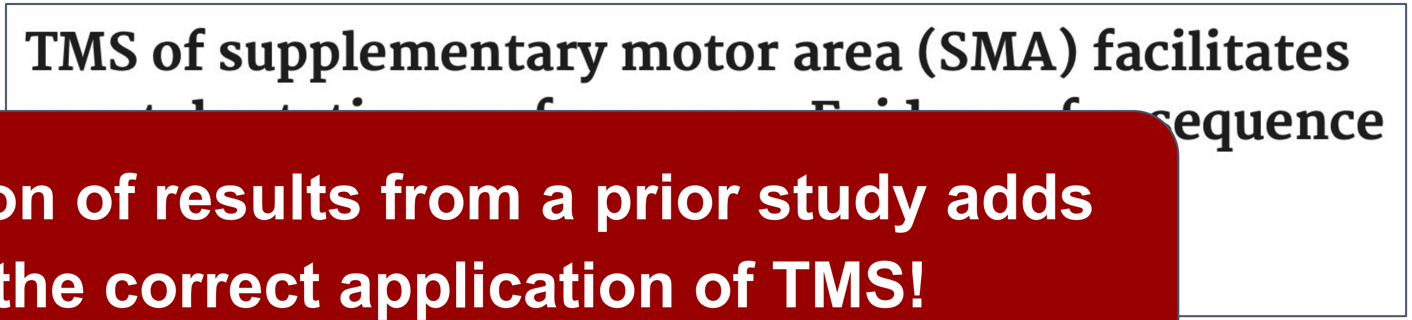
ACM SIGSOFT Distinguished Paper Award

TMS for Programming: Human Study Results



“Does TMS of the SMA influence spatial reasoning performance?”

- Stimulating the SMA affects the time taken to perform mental rotation tasks (15.3% increase, *p* $\leq$ 0.02, **statistically significant**)
    - Partial replication of results from Cona et al.

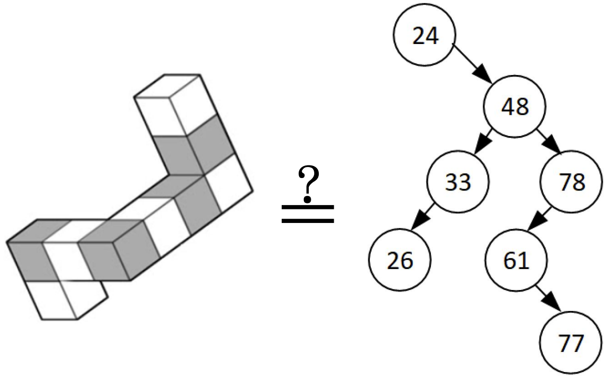


TMS of supplementary motor area (SMA) facilitates

sequence

Our partial replication of results from a prior study adds confidence in the correct application of TMS!

TMS for Programming: Human Study Results



“Do we use the same areas of our brains for spatial reasoning and programming?”

- **No evidence of a direct causal relationship** between programming outcomes and brain activity in SMA and M1 (**!!!**)
  - Disrupting brain activity for spatial reasoning does not affect response accuracy or time for programming when compared to the baseline
  - Results disagree with multiple previously-published correlations

Our previous understanding of the brain’s involvement in programming may not be correct!

TMS for Programming: Human Study Results

- **TMS can affect response times** for programming tasks
  - Multi-level regression analysis reveals a **2.2% variance** in response time attributed to TMS, **statistically significant**

TMS for Programming: Human Study Results

- **TMS can affect response times** for programming tasks
  - Multi-level regression analysis reveals a **2.2% variance** in response time attributed to TMS, **statistically significant**

Factor Affecting Response Times	**Effect Size (Normalized)**
“How hard is the question?”	1.00

TMS for Programming: Human Study Results

- **TMS can affect response times** for programming tasks
    - Multi-level regression analysis reveals a **2.2% variance** in response time attributed to TMS, **statistically significant**

Factor Affecting Response Times	**Effect Size (Normalized)**
“How hard is the question?”	1.00
“Participant expertise”	0.18

TMS for Programming: Human Study Results

- **TMS can affect response times** for programming tasks
  - Multi-level regression analysis reveals a **2.2% variance** in response time attributed to TMS, **statistically significant**

Factor Affecting Response Times	**Effect Size (Normalized)**
“How hard is the question?”	1.00
“Participant expertise”	0.18
“TMS”	**0.05**

TMS for Programming: Human Study Results

- **TMS can affect response times** for programming tasks
    - Multi-level regression analysis reveals a **2.2% variance** in response time attributed to TMS, **statistically significant**

Factor Affecting Response Times	**Effect Size (Normalized)**
TMS	0.05

Neurostimulation can be used to alter computing outcomes, warranting further exploration of the technique to investigate causality!

TMS for Programming: Wrapping it Up

Can we use neurostimulation to investigate brain activity for coding tasks (i.e., programming logic)?

TMS for Programming: Wrapping it Up

Can we use neurostimulation to investigate brain activity for coding tasks (i.e., programming logic)?

- No evidence of a causal relationship between activity in SMA / M1 and reasoning about programming
    - Our results disagree with multiple previously published correlations, challenging our understanding of the brain’s involvement in programming

TMS for Programming: Wrapping it Up

Can we use neurostimulation to investigate brain activity for coding tasks (i.e., programming logic)?

- No evidence of a causal relationship between activity in SMA / M1 and reasoning about programming
    - Our results disagree with multiple previously published correlations, challenging our understanding of the brain’s involvement in programming

- Neurostimulation can alter programming outcomes

Publications (supporting this thesis)

1. **Causal Relationships and Programming Outcomes: A Transcranial Magnetic Stimulation Experiment.** Hammad Ahmad, Madeline Endres, Kaia Newman, Priscila Santiesteban, Emma Shedden, Westley Weimer. *ICSE (2024)*. **[ACM Distinguished Paper Award]**
2. **CirFix: Automated Hardware Repair and its Real-Word Applications.** Priscila Santiesteban, Yu Huang, Westley Weimer, Hammad Ahmad. *TSE (2023).*
3. **How Do We Read Formal Claims? Eye-Tracking and the Cognition of Proofs about Algorithms.** Hammad Ahmad, Zachary Karas, Kimberly Diaz, Amir Kamil, Jean-Baptiste Jeannin, Westley Weimer. *ICSE (2023)*.
4. **LOGI: An Empirical Model of Heat-Induced Disk Drive Data Loss and its Implications for Data Recovery.** Hammad Ahmad, Colton Holoday, Ian Bertram, Kevin Angstadt, Zohreh Sharafi, Westley Weimer. *PROMISE (2022)*.
5. **Sift: Using Refinement-Guided Automation to Verify Complex Distributed Systems.** Haojun Ma, Hammad Ahmad, Aman Goel, Eli Goldweber, Jean-Baptiste Jeannin, Manos Kapritsos, Baris Kasikci. *ATC (2022)*.
6. **Digging into Semantics: Where do search-based software repair methods search?** Hammad Ahmad, Padraic Cashin, Stephanie Forrest, Westley Weimer. *PPSN (2022)*.
7. **CirFix: Automatically Repairing Defects in Hardware Design Code.** Hammad Ahmad, Yu Huang, Westley Weimer. *ASPLOS (2022)*.
8. **Applying Automated Program Repair to Dataflow Programming Languages.** Yu Huang, Hammad Ahmad, Stephanie Forrest, Westley Weimer. *GI Workshop @ ICSE (2021)*.
9. **A Program Logic to Verify Signal Temporal Logic Specifications of Hybrid Systems.** Hammad Ahmad, Jean-Baptiste Jeannin. *HSCC (2021)*.
10. **A Comparison of Semantic-Based Initialization Methods for Genetic Programming**. Hammad Ahmad, Thomas Helmuth. *Student Workshop @ GECCO (2018).*

Acknowledgements

My advisor: Westley Weimer

Acknowledgements

My co-advisor: Jean-Baptiste Jeannin

Acknowledgements

My committee members

Prof. Westley Weimer

Lec. IV Amir Kamil

Prof. Stephanie Forrest

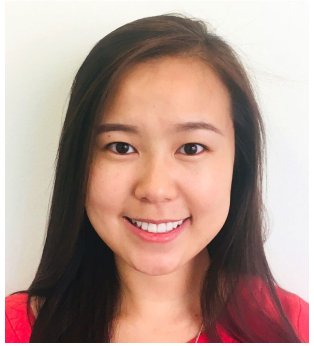
Asst. Prof. Taraz Lee

Acknowledgements

My collaborators and mentors over the years


Dr. Sara Sprenkle
(W&L)


Dr. Kevin Angstadt
(St. Lawrence)


Dr. Yu Huang
(Vanderbilt)


Dr. Kimberly Diaz
(UMich)


Dr. Manos Kapritsos
(UMich)


Dr. Baris Kasikci
(UWashington)


Dr. Zohreh Sharafi
(Polytechnique Montréal)


Dr. David Paoletti
(UMich)


Prof. Marcus Darden
(UMich)

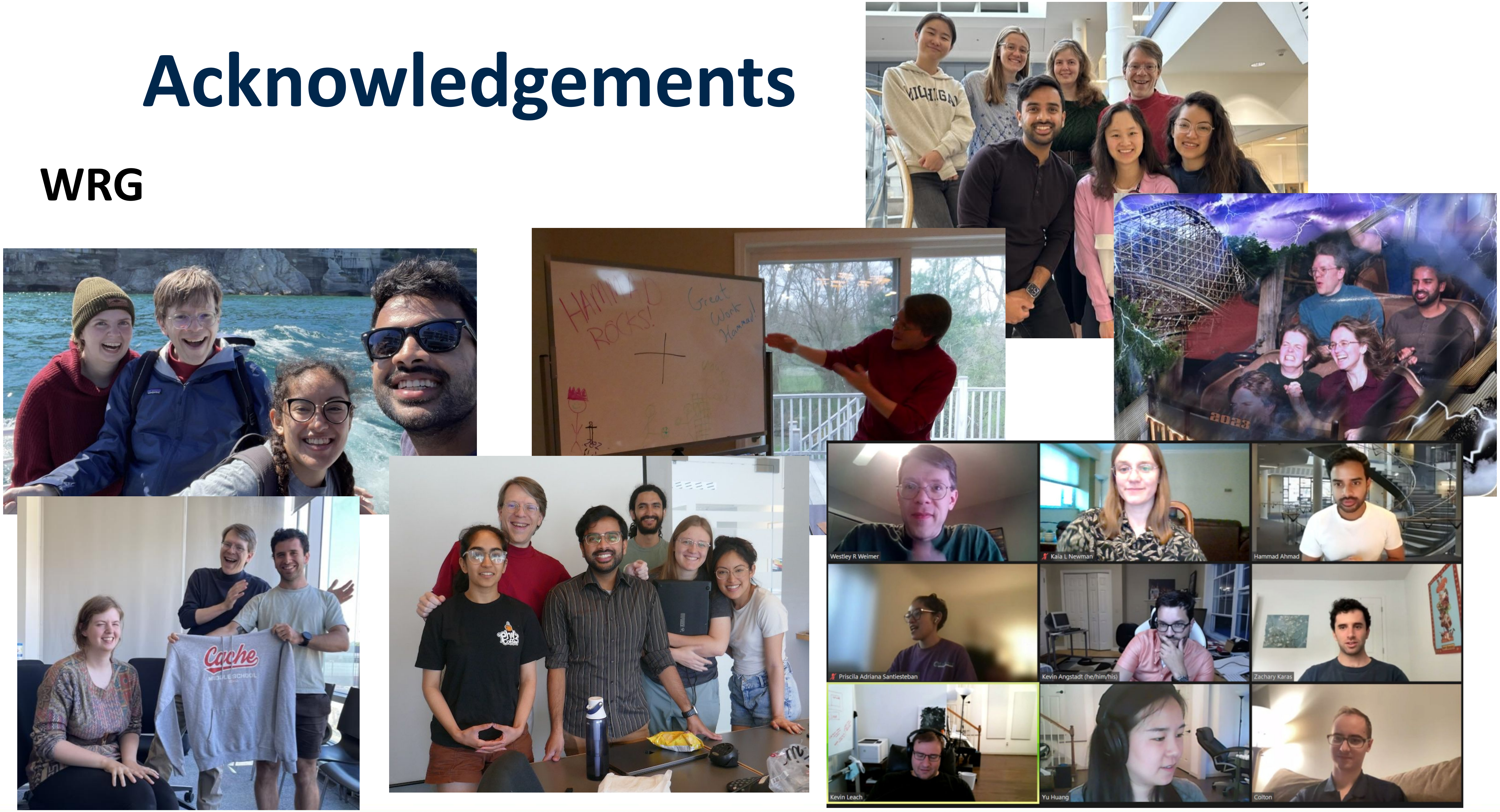

Dr. Héctor Garcia-Ramirez
(UMich)


Dr. James Brissenden
(UMich)

and others…

Acknowledgements

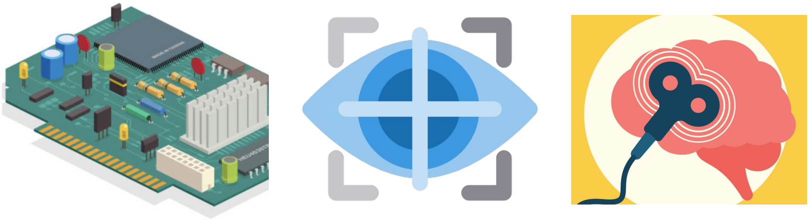
WRG



Acknowledgements

My friends and family

Putting It All Together…



- Humans and computers think in different ways

- We can use **functional**, **physiological**, and **medical** methods to better understand how humans reason about computational logic
    - **Functional**: “Can you find the bug?”
    - **Physiological**: “Where are you looking as you search for the bug?”
    - **Medical**: “What goes on in your brain as you search for the bug?”

- Knowing the **cognitive basis of logical reasoning** can help us enhance tool support for developers and explore more effective methods to teach CS

- De-identified datasets publicly available at: https://websites.umich.edu/~hammada/research/