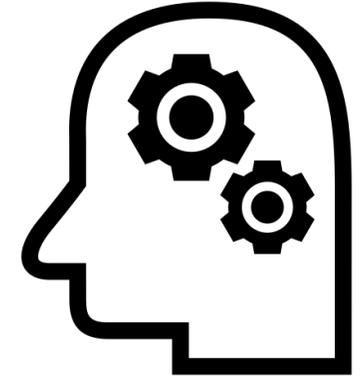


Understanding *understanding*: How do we reason about computational logic?

Hammad Ahmad
Ph.D. Dissertation Proposal

“Understanding *understanding*”

Cognition: Mental processes involved in comprehension and gaining knowledge



Cognition and Pedagogy



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

Ryan Bockmon
University of Nebraska - Lincoln
Lincoln, Nebraska
rbockmon@cse.unl.edu

Stephen Cooper
University of Nebraska - Lincoln
Lincoln, Nebraska
scooper22@unl.edu

William Koperski
University of Nebraska - Lincoln
Lincoln, Nebraska
wjakoperski@huskers.unl.edu

Jonathan Gratch

Sheryl Sorby

Mohsen Dorodchi

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

Sheryl Sorby^{a,*}, Norma Veurink^b, Scott Streiner^c

To Read or to Rotate? Comparing the Effects of Technical Reading Training and Spatial Skills Training on Novice Programming Ability

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

Priti Shah
priti@umich.edu
University of Michigan, Psychology
USA

Madison Fansher
mfansher@umich.edu
University of Michigan, Psychology
USA

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

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

J. Mock¹ · S. Huber¹ · E. Klein^{1,2} · K. Moeller^{1,3,4}

Investigating Visual Perception in Teaching and Learning with Advanced Eye-Tracking Methodologies: Rewards and Challenges of an Innovative Research Paradigm

Matthias Nückles¹ 

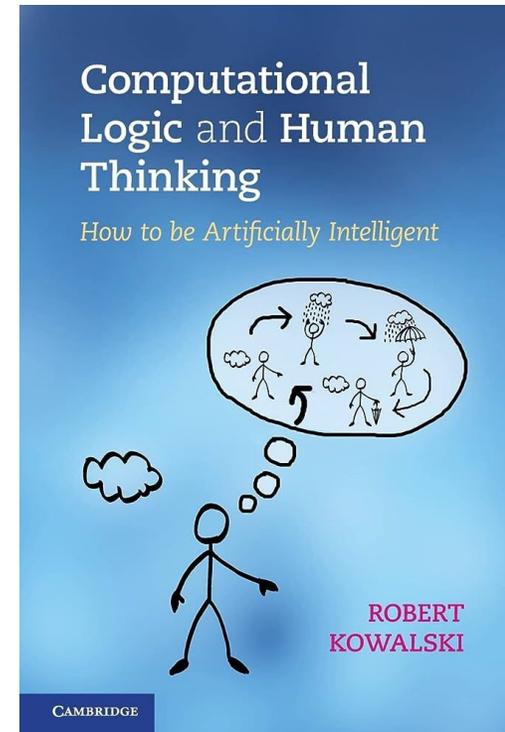
“Computational Logic”

Computers do not think like humans do!

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

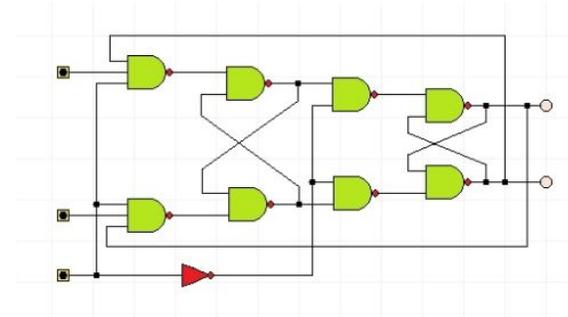
Logical reasoning in CS forms a **core component** of undergraduate CS curricula

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



Defining “Logic”

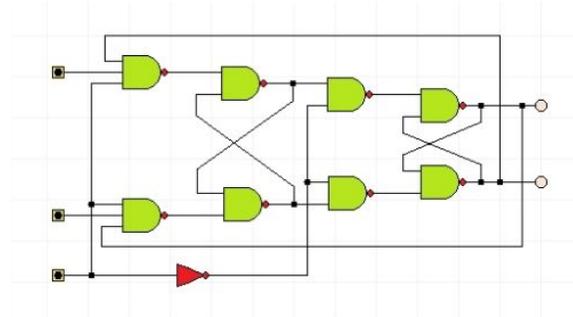
Digital logic
(e.g., hardware designs)



Defining “Logic”

Digital logic

(e.g., hardware designs)



Mathematical logic

(e.g., proofs about algorithms)

MATHEMATICAL INDUCTION PROOF

INDUCTION ASSUME TRUE

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

BASIS $n=1$ $1=1$ ✓

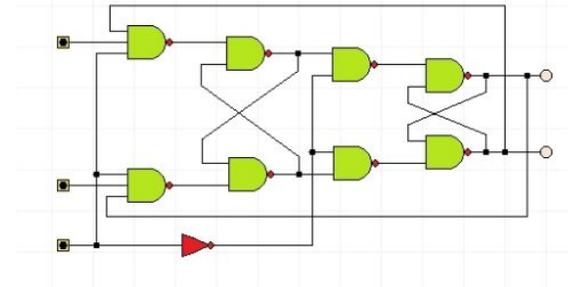
SHOW TRUE $n=k+1$

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

Defining “Logic”

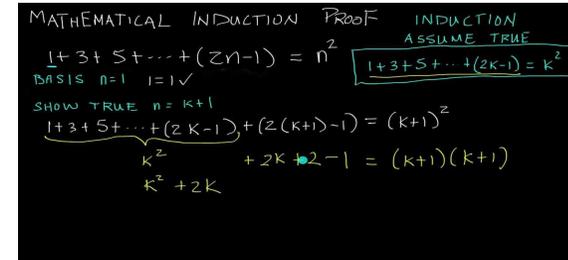
Digital logic

(e.g., hardware designs)



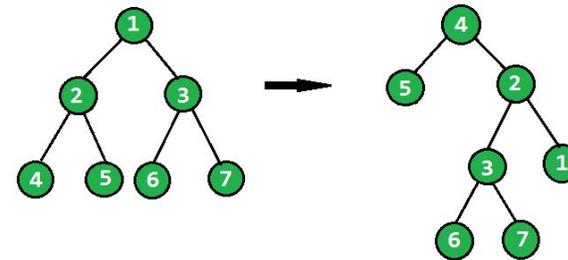
Mathematical logic

(e.g., proofs about algorithms)



Programming logic

(e.g., manipulating data structures)



Desired Properties of a Solution

(1) Non-intrusive Methodology



instead of



Desired Properties of a Solution

(2) Objective Measures



instead of



Line: 16 Col: 56

Test Results Custom Input Run Code Run Tests Submit

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)

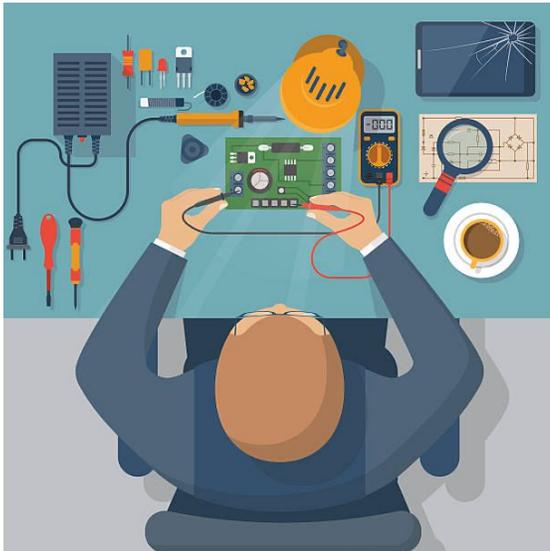
```
1 5 -1
2 0.0 0.0
```

Expected Output [Download](#)

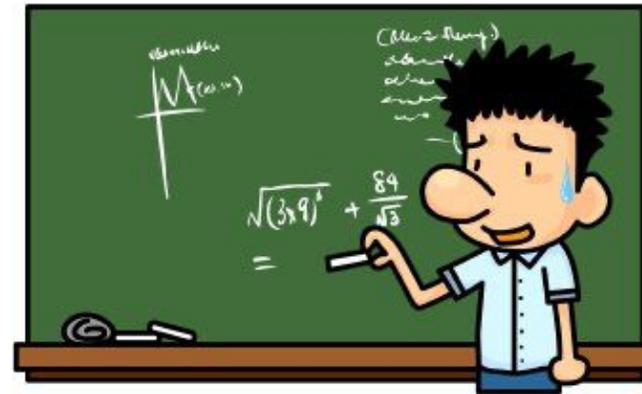
```
1 5 -1
2 0.0 0.0
```

Desired Properties of a Solution

(3) Context-specific Models



VS.

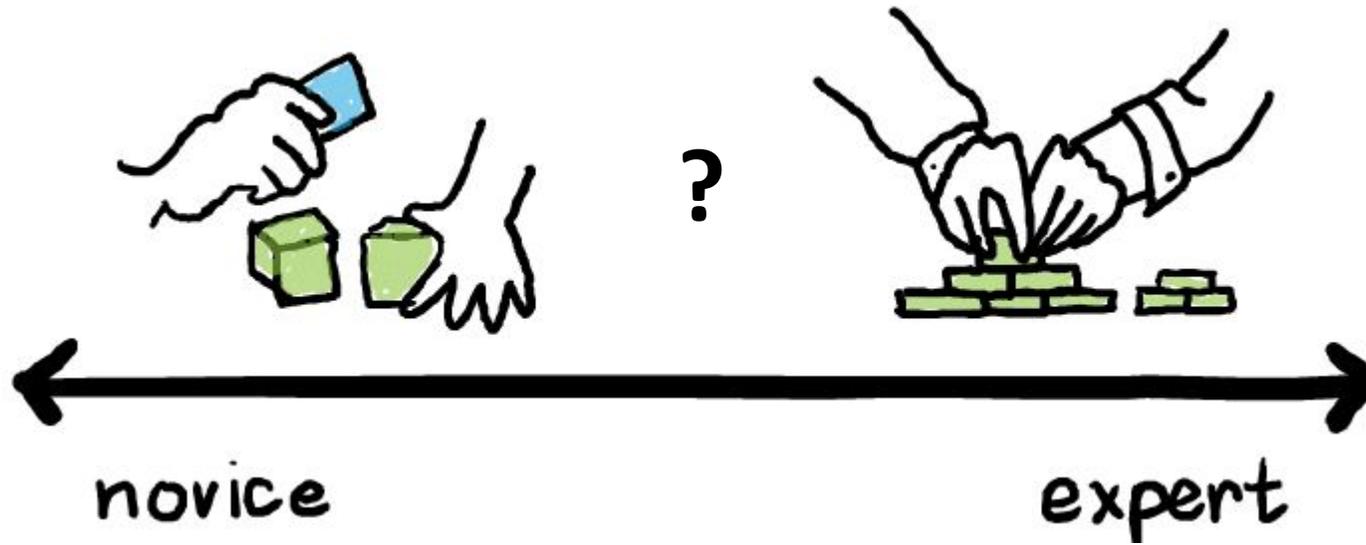


VS.



Desired Properties of a Solution

(4) Incoming Preparation or Expertise



Insights

We can implement **objective, non-intrusive** measures in a CS context to obtain **correlations**.

We can use **medical devices** in a CS context to investigate **causality**.

We can employ **advanced statistical rigor** in CS to account for **student background and context**.

Insights: Explained

We can implement **objective, non-intrusive** measures in a CS context to obtain **correlations**.

Insights: Explained

We can implement **novel debugging algorithms** and **eye-tracking** in a CS context to obtain **correlations**.

Insights: Explained

We can implement **novel debugging algorithms** and **eye-tracking** in a CS context to obtain **correlations**.

We can use **medical devices** in a CS context to investigate **causality**.

Insights: Explained

We can implement **novel debugging algorithms** and **eye-tracking** in a CS context to obtain **correlations**.

We can use **functional magnetic resonance imaging (fMRI)** and **transcranial magnetic stimulation (TMS)** in a CS context to investigate **causality**.

Insights: Explained

We can implement **novel debugging algorithms** and **eye-tracking** in a CS context to obtain **correlations**.

We can use **functional magnetic resonance imaging (fMRI)** and **transcranial magnetic stimulation (TMS)** in a CS context to investigate **causality**.

We can employ **advanced statistical rigor** in CS to account for **student background and context**.

Insights: Explained

We can implement **novel debugging algorithms** and **eye-tracking** in a CS context to obtain **correlations**.

We can use **functional magnetic resonance imaging (fMRI)** and **transcranial magnetic stimulation (TMS)** in a CS context to investigate **causality**.

We can employ **advanced statistical rigor** in CS to account for **student incoming preparation effects on task outcomes**.

Thesis Statement

*It is possible to use **objective measures** to obtain context-specific **mathematical models** of the **cognitive processes** underlying logical reasoning, and these models can accurately **explain student behavior**.*

Thesis Statement: Explained

*It is possible to use **functional**, **physiological**, and **medical** measures to obtain context-specific **mathematical models** of the **cognitive processes** underlying logical reasoning, and these models can accurately **explain student behavior**.*

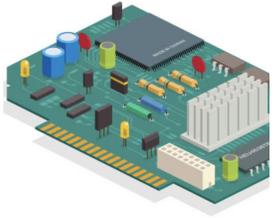
If we can construct an accurate model for the cognitive processes associated with computational reasoning tasks, **educators may be able to use that understanding to investigate how to better teach logical reasoning to students.**

Proposal Overview

Three components:

Proposal Overview

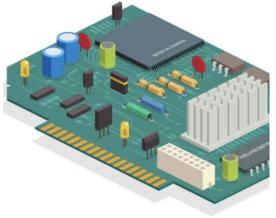
Three components:



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

Proposal Overview

Three components:



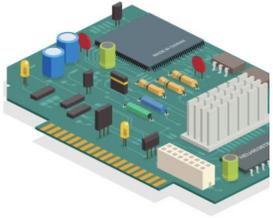
Using automated program repair for hardware as a debugging assistant for designers



Using eye-tracking to understand cognition for computer science formalisms

Proposal Overview

Three components:



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



Using eye-tracking to understand cognition for computer science formalisms



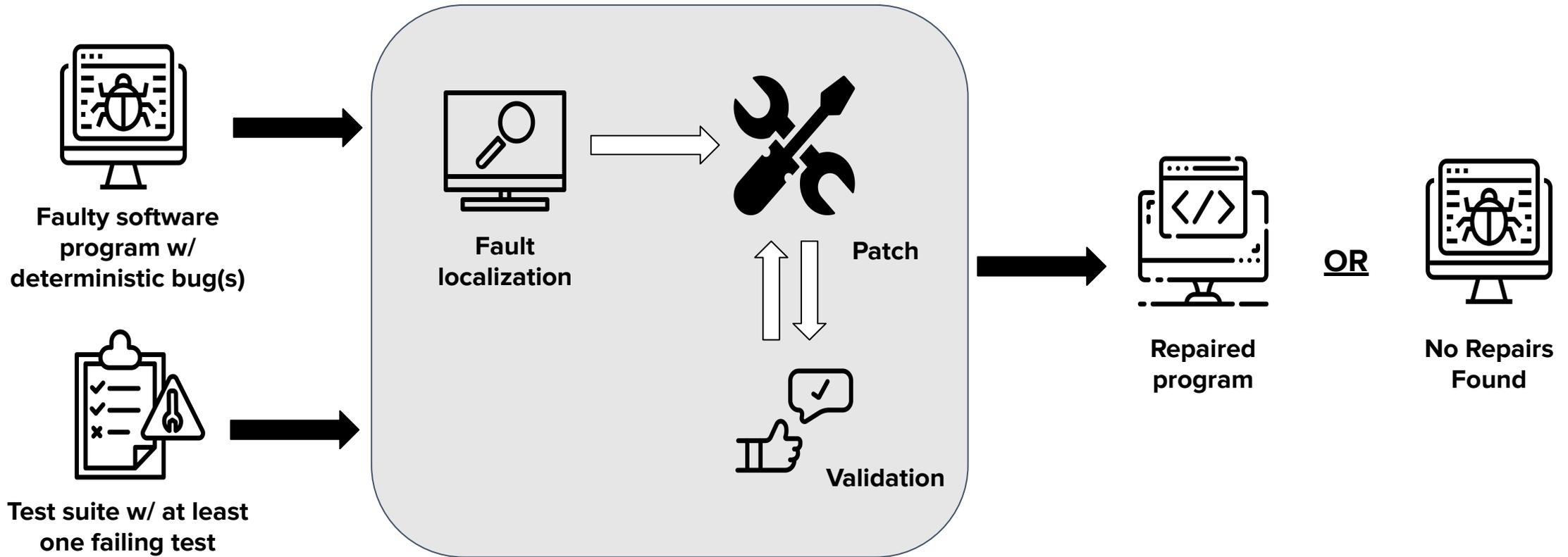
Using TMS to codify the relationship between spatial reasoning and programming



Automated Program Repair for Hardware as a Debugging Assistant

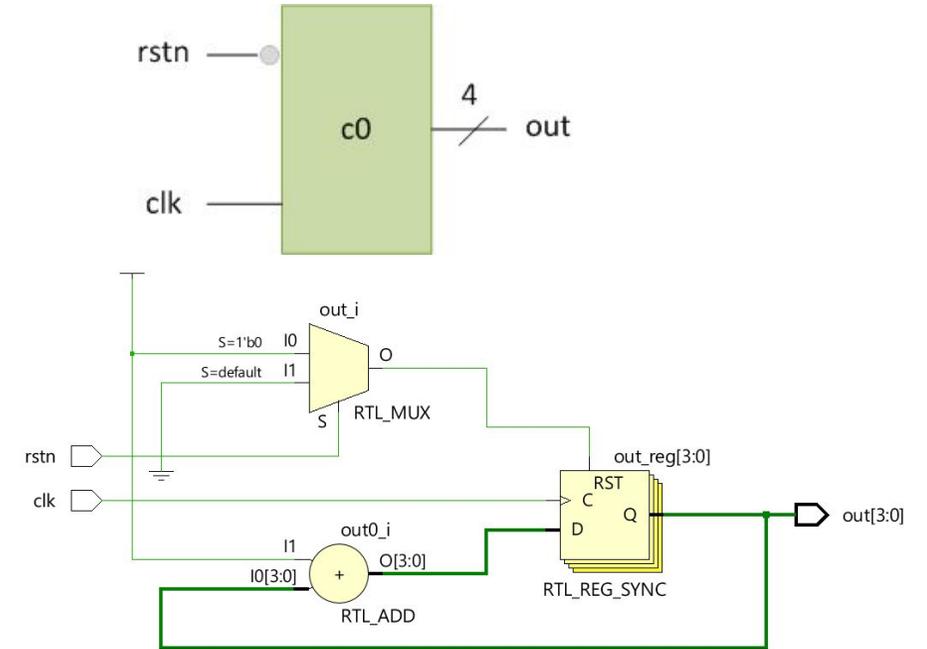
Can we build a state-of-the-art automated repair tool for hardware designs (i.e., **digital logic**), and use it as a debugging assistant for designers?

Automated Program Repair (APR)



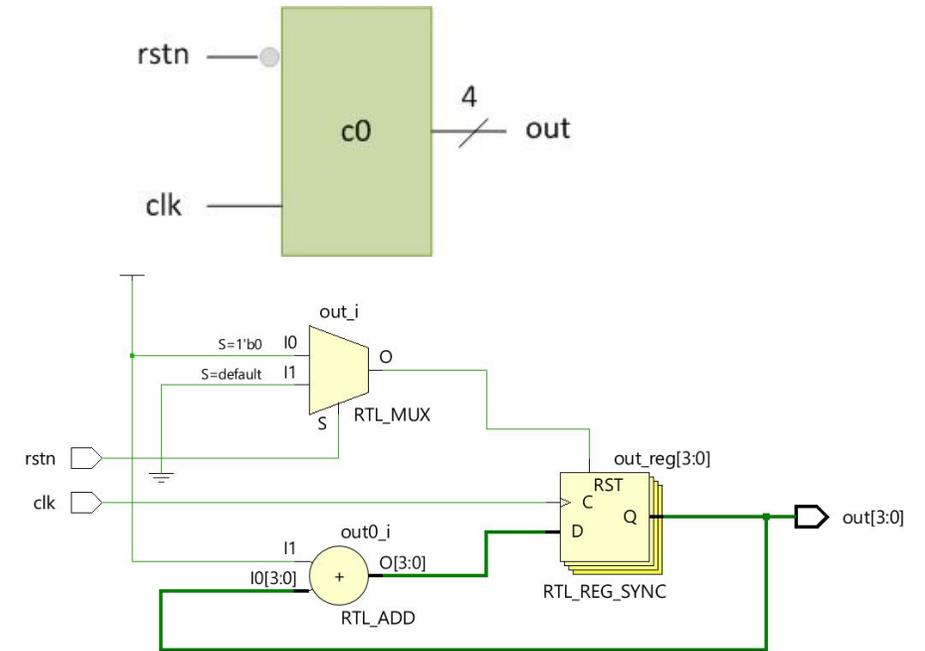
Hardware Designs

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



Hardware Designs

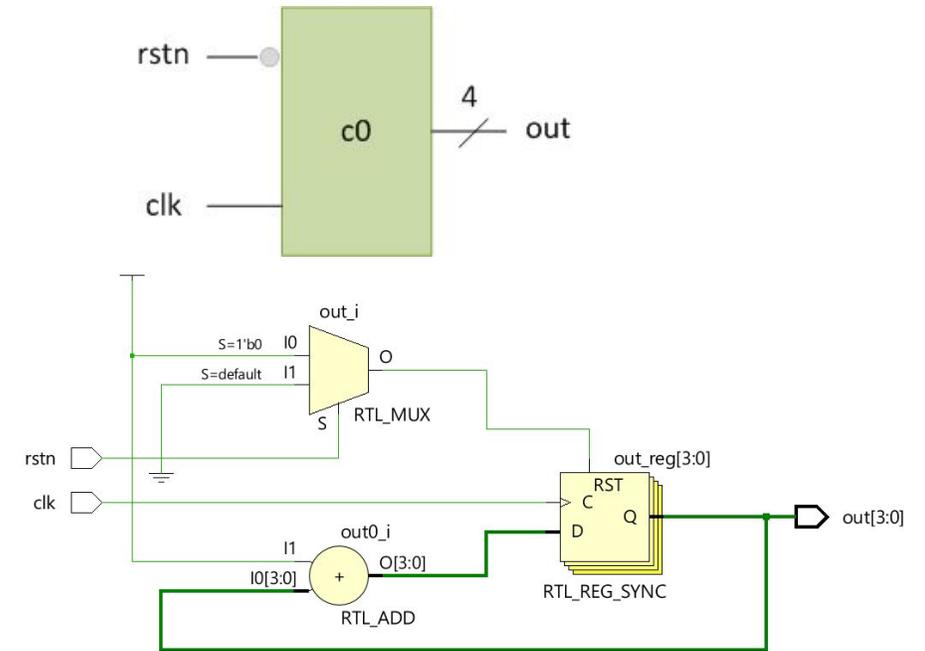
- **Digital specifications** for electronic devices, computer systems, or integrated circuits
- Typically written using **hardware description languages (HDLs)** like Verilog and VHDL



```
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
```

Software vs. Hardware

One 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

APR for Hardware?

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

How do we bridge the gap between software APR and hardware designs?

Introducing: *CirFix*

CirFix: A hardware-design focused automated repair algorithm based on genetic programming

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

- Preliminary results in ASPLOS'22 and TSE'23

More details in the proposal text!

RQ1: Experiments and Metrics

RQ1: What fraction of defects can CirFix actually repair?

Problem: No publicly-available benchmarks for hardware defects that are indicative of real industrial defects and corresponds to a wide range of project sizes (largely due to IP constraints)!

How do we evaluate CirFix?

RQ1: Experiments and Metrics

Problem: No publicly-available benchmarks for hardware defects that are indicative of real industrial defects and corresponds to a wide range of project sizes (largely due to IP constraints)!

Constructed a **benchmark suite** of 32 different hardware defects to evaluate CirFix

RQ1: Experiments and Metrics

Problem: No publicly-available benchmarks for hardware defects that are indicative of real industrial defects and corresponds to a wide range of project sizes (largely due to IP constraints)!

Constructed a **benchmark suite** of 32 different hardware defects to evaluate CirFix

- Corresponds to 6 introductory-level circuit designs and 5 off-the-shelf (larger, industrial) designs
- Includes 19 “easy” defects and 13 “hard” defects

We make our benchmark suite **publicly available** for future researchers to evaluate hardware repair approaches!

RQ1: Preliminary Results

RQ1: What fraction of defects can CirFix actually repair?

- Ran five resource-constrained, independent CirFix trials for each defect, stopping when a **plausible repair** (i.e., *a repair passing all tests*) was found
- CirFix found 21/32 (**65.6%**) plausible repairs, with 16/32 (**50%**) deemed to be correct (i.e., *high quality*) upon manual inspection
- Repair rate comparable to strong results from software-based APR (e.g., GenProg at **52.5%**, Angelix at **34.1%**)

RQs 2-3: Experiments and Metrics

RQ2: Does the CirFix fault localization improve designers' objective performances?

RQ3: In what contexts do designers find CirFix helpful?

Problem: Need to have real designers use CirFix as a debugging assistant to evaluate its efficacy!

How do we meaningfully evaluate real designers using CirFix?

RQs 2-3: Controlled Human Study

- Conducted under IRB **HUM00199335**
- **n = 41 participants** in the study
- Participants asked to identify and fix defects from the CirFix benchmark, each accompanied with **no debugging hints**, **partial debugging hints**, or full **debugging hints**
 - Partial hints: highlighting **variables** implicated by CirFix
 - Full hints: highlighting **lines of code** implicated by CirFix
- Participants also asked to rate the **accuracy** and **helpfulness** of debugging hints (where applicable)
- Designer performance assessed by evaluating F-scores (F_1) and time taken to complete each debugging task

RQs 2-3: Controlled Human Study

```
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
```

Example stimulus

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.

RQs 2: Results

RQ2: Does the CirFix fault localization improve designers' objective performances?

- **No statistically significant difference in time taken** to localize faults with annotations ($p = 0.41$, Student t-test)
- **F-score for participants higher** for full hints vs. partial hints vs. no hints ($F_1 = 0.67$, $F_1 = 0.33$, $F_1 = 0.29$)
 - Trend was not statistically significant ($p = 0.12$)
- **Difference in F-scores for experts vs. novices** when given debugging hints ($F_1 = 0.37$, $F_1 = 0.17$)
 - Statistically significant with **large effect size** ($p = 0.04$, $d = 0.54$)

RQs 2: Results

RQ2: Does the CirFix fault localization improve designers' objective performances?

- **F-score for participants higher** for full hints vs. partial hints vs. no hints ($F_1 = 0.67$, $F_1 = 0.33$, $F_1 = 0.29$)
 - Trend was not statistically significant ($p = 0.12$)
- **Difference in F-scores for experts vs. novices** when given debugging hints ($F_1 = 0.37$, $F_1 = 0.17$)
 - Statistically significant with **large effect size** ($p = 0.04$, $d = 0.54$)

⇒ **CirFix can be useful as a hardware debugger!**

RQs 3: Results

RQ3: In what contexts do designers find CirFix helpful?

- Full debugging hints on intro-level designs rated as **significantly more helpful** than those on larger, off-the-shelf designs ($p = 0.01$, $d = 0.7$; $p = 0.002$, $d = 1.05$; **large effect size**)

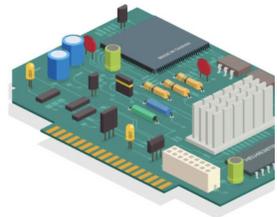
⇒ **CirFix can be more beneficial as a debugging assistant in a pedagogical context!**

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

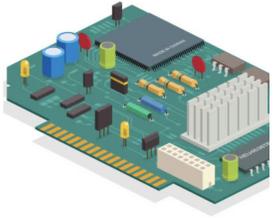
Can we build a state-of-the-art automated repair tool for hardware designs (i.e., **digital logic**), and use it as a debugging assistant for designers?



Yes, we can!

Proposal Overview

Three components:



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



Using eye-tracking to understand cognition for computer science formalisms



Using TMS to codify the relationship between spatial reasoning and programming



Eye-Tracking for Computer Science Formalisms

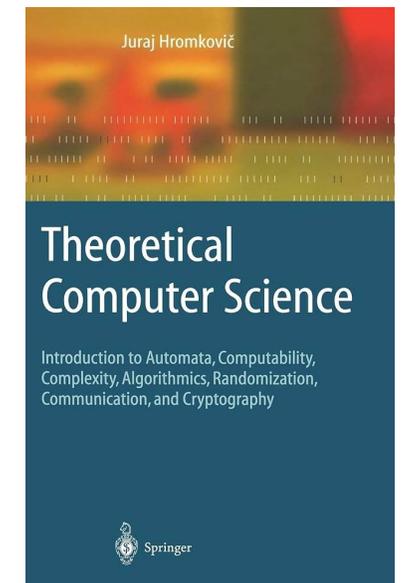
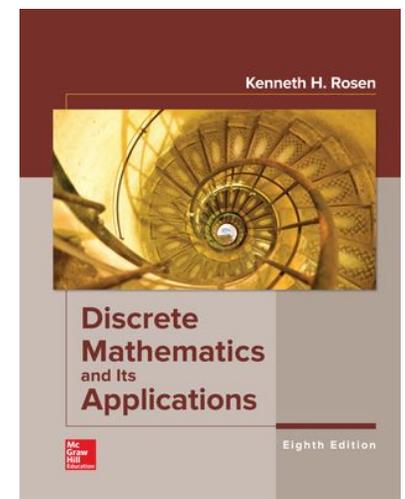
Can we use objective measures to investigate how students read and understand computer science formalisms (i.e., **mathematical logic**)?

Formalism Comprehension

Educators put a lot of emphasis on training students for 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)

Yet, undergraduate CS theory courses tend to have **poor student outcomes** and **satisfaction**



Formalism Comprehension

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

Formalism Comprehension

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

Sadly, not as much as we would like. :-)

Enter: Eye-Tracking

- **Objective measure** for participant problem solving strategies
- **Cheap and non-invasive**
- Approximates dynamics of **visual attention** (e.g., where participants focus, and for how long)
- Serves as a proxy for **cognitive load** (i.e., strain on working memory) and **task difficulty**



Formalism Comprehension: Experiments and Metrics

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.

- 1: **if** $n = 1$ **then**
- 2: move disk n from A to C
- 3: $ToH(n - 1, A, C, B)$ ▷ Move $n - 1$ disks from A to B using C .
- 4: Move disk n from A to C
- 5: $ToH(n - 1, B, A, C)$ ▷ Move $n - 1$ disks from B to C using A .

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. □

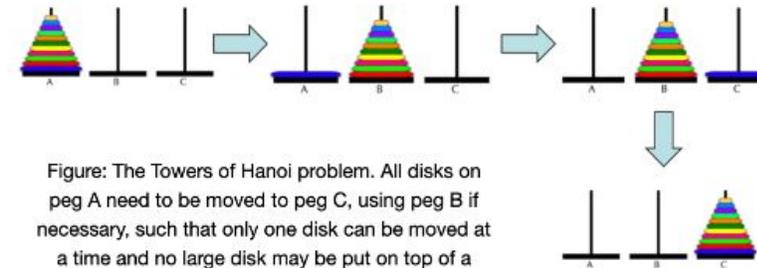


Figure: The Towers of Hanoi problem. All disks on peg A need to be moved to peg C , using peg B if necessary, such that only one disk can be moved at a time and no large disk may be put on top of a smaller disk.

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: Experiments and Metrics

Pre-defined 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.

- 1: **if** $n = 1$ **then**
- 2: move disk n from A to C
- 3: $ToH(n - 1, A, C, B)$ ▷ Move $n - 1$ disks from A to B using C .
- 4: Move disk n from A to C
- 5: $ToH(n - 1, B, A, C)$ ▷ Move $n - 1$ disks from B to C using A .

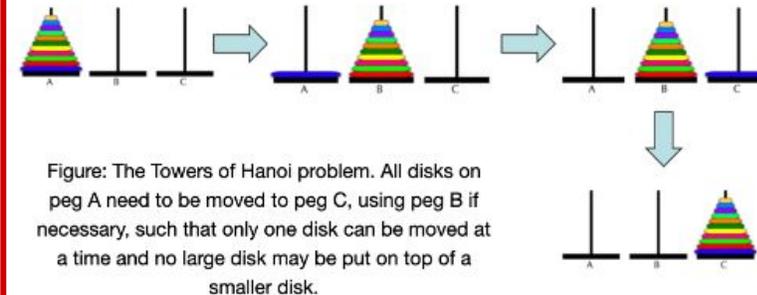
Theorem. The Towers of Hanoi (ToH) algorithm correctly moves n disks from pegs A to C using peg B if necessary such that only one disk can be moved at a time and a large disk cannot be put on top of a smaller disk.

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: Experiments and Metrics

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.

- 1: **if** $n = 1$ **then**
- 2: move disk n from A to C
- 3: $ToH(n - 1, A, C, B)$ ▷ Move $n - 1$ disks from A to B using C .
- 4: Move disk n from A to C
- 5: $ToH(n - 1, B, A, C)$ ▷ Move $n - 1$ disks from B to C using A .

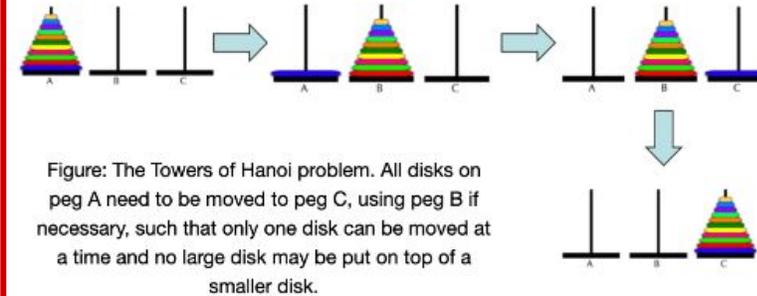


Figure: The Towers of Hanoi problem. All disks on peg A need to be moved to peg C , using peg B if necessary, such that only one disk can be moved at a time and no large disk may be put on top of a smaller disk.

Fixation

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.

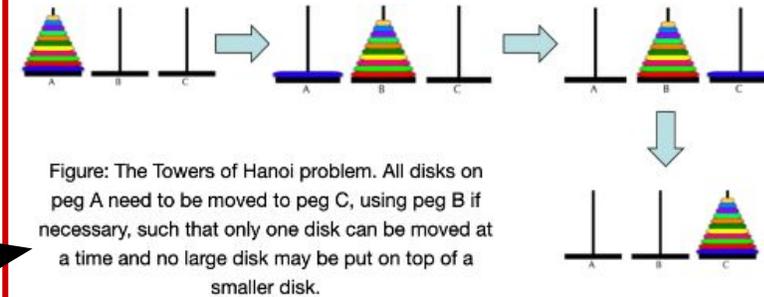
(4) The proof should perform induction on the number of steps required to moved all disks from peg A to C , instead of performing induction on the number of disks.

Formalism Comprehension: Experiments and Metrics

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.

- 1: **if** $n = 1$ **then**
- 2: move disk n from A to C
- 3: $ToH(n - 1, A, C, B)$ ▷ Move $n - 1$ disks from A to B using C .
- 4: Move disk n from A to C
- 5: $ToH(n - 1, B, A, C)$ ▷ Move $n - 1$ disks from B to C using A .



Saccade

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: Experiments and Metrics

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.

- 1: **if** $n = 1$ **then**
- 2: move disk n from A to C
- 3: $ToH(n - 1, A, C, B)$ ▷ Move $n - 1$ disks from A to B using C.
- 4: Move disk n from A to C
- 5: $ToH(n - 1, B, A, C)$ ▷ 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.

Attention
Switching

Formalism Comprehension: Controlled Human Study

- Conducted under IRB **HUM00204278**
- **n = 34 participants** in the study
- Participants shown a series of **algorithmic proofs** from an undergraduate textbook, each with an associated figure and possible logical / arithmetic mistake
- Participants asked to identify the presence of mistakes in each proof
- **Individual performance** assessed by evaluating response accuracy and time, **response strategy** assessed by evaluating gaze data
- Preliminary results in ICSE'23

RQ1: Experiments and Metrics

RQ1: What is the effect of incoming preparation on student outcomes for formalism comprehension?

Problem: Incoming preparation is hard to measure!

How do we determine if someone is more- or less-prepared for formal reasoning?

RQ1: Experiments and Metrics

Problem: Incoming preparation is hard to measure!

Use both **coursework count** and **performance** as a proxy for preparation

- **Coursework count:** The number of CS theory courses covering formalisms a participant has completed with passing grades or is currently enrolled in
- **Performance:** Whether or not a participant correctly identifies the mistake in a pre-screening proof from an undergraduate textbook

Participants who have course count $>$ the median value and pass the pre-screening classified as *more-prepared* (**16/34** participants)

RQ1: Preliminary Results

RQ1: What is the effect of incoming preparation on student outcomes for formalism comprehension?

- **No statistically significant difference in response times and accuracies** between more- and less-prepared participants ($p = 0.93$, $p = 0.96$; two-tailed Mann-Whitney U-test)
- **No correlation between # theory courses and response accuracy** (Pearson's $r = 0.036$, $p = 0.84$)

⇒ **Students with more incoming preparation perform no better on formalism comprehension tasks, on average, than students with lower incoming preparation!**

RQ1: Preliminary Results

RQ1: What is the effect of incoming preparation on student outcomes for formalism comprehension?

- More-prepared students **fixate longer** on the proof text ($p = 0.005$), correct answer ($p = 0.038$), and distractor choices ($p = 0.03$)

⇒ **Students with more incoming preparation may be better trained to read the proof and answer choices more thoroughly, yet do not achieve better outcomes than students with less preparation!**

RQ2: Preliminary Results

RQ2: How do student performance self-reports align with more empirical task outcomes?

- **No correlation** 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 ($\tau = -0.14$, $p = 0.32$)

⇒ **Students may not be accurate at self-reporting their experience or familiarity with formalism comprehension tasks!**

RQ3: Preliminary Results

RQ3: What distinguishes higher-performing students from lower performing ones?

Participants with above median response accuracy classified as *higher-performing* (**15/34** participants)

- Higher-performing participants **more likely to spot mistakes in inductive proofs** (χ^2 test, $p = 0.01$)
- Higher-performing participants **more likely to spot mistakes in proofs for recursive algorithms** ($p = 0.006$)

⇒ **Lower-performing students may benefit from more practice with inductive proofs and recursive algorithms.**

RQ3: Preliminary Results

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

Algorithm Greedy Change-Making

Input: n , positive integer.
Input: c_1, \dots, c_r : values of denominations of coins, where $c_1 > c_2 > \dots > c_r$.
Output: $\{d_1, d_2, \dots, d_r\}$ d_i is the number of coins of denomination c_i in the change for n .

```

1: for  $i = 1$  to  $r$  do
2:    $d_i := \lfloor n / c_i \rfloor$ 
3:    $n := n - d_i c_i$ 
4:    $d_i := d_i + 1$ 
5:    $n := n - c_i$ 
6: return  $\{d_1, d_2, \dots, d_r\}$ 
  
```

Theorem: If n is a positive integer, then n cents can be paid 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 produce change worth more than 24 cents using only dimes, nickels, and pennies,
- cannot produce change worth more than 24 cents using only dimes, nickels, and pennies.

Proof: We prove this theorem by contradiction. We will show that if we had more than the specified number of coins of each type, we could replace them using fewer coins that is the same value. We note that if we had five pennies we could replace them with a nickel. We note that if we had two dimes and a nickel, if we had two nickels we could replace them with a dime, if we had five pennies we could replace them with a nickel, and if we had two dimes and a nickel we could replace them with a quarter. Because we can have at most two dimes, one nickel, and four pennies, but we cannot have two dimes and a nickel, it follows that 24 cents is the most money we can have in dimes, nickels, and pennies when we make change using the fewest number of coins for n cents. \square

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

$n = 92$  **Optimal # of coins**

$n = 30$  **Optimal # of coins**

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?

- No mistake.
- We cannot prove this theorem by contradiction. We need to perform an induction on n .
- The proof does not correctly establish that 24 cents is the most money we can have in dimes, nickels, and pennies being the fewest number of coins.
- This is a direct proof, not a proof by contradiction, since we do not assume a hypothesis and negate it later.

Algorithm Greedy Change-Making

Input: n , positive integer.
Input: c_1, c_2, \dots, c_r : values of denominations of coins, where $c_1 > c_2 > \dots > c_r$.
Output: $\{d_1, d_2, \dots, d_r\}$ d_i is the number of coins of denomination c_i in the change for $i = 1, 2, \dots, r$.

```

1: for  $i = 1$  to  $r$  do
2:    $d_i := 0$ 
3:   while  $n \geq c_i$  do
4:      $d_i := d_i + 1$ 
5:      $n := n - c_i$ 
6: return  $\{d_1, d_2, \dots, d_r\}$ 
  
```

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 theorem by contradiction. We will show that if we had more than the specified number of coins of each type, we could replace them using fewer coins that is the same value. We note that if we had five pennies we could replace them with a nickel, and if we had two dimes and a nickel, we could replace them with a quarter. Because we can have at most two dimes, one nickel, and four pennies, but we cannot have two dimes and a nickel, it follows that 24 cents is the most money we can have in dimes, nickels, and pennies when we make change using the fewest number of coins for n cents. \square

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

$n = 92$  **Optimal # of coins**

$n = 30$  **Optimal # of coins**

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?

- No mistake.
- We cannot prove this theorem by contradiction. We need to perform an induction on n .
- The proof does not correctly establish that 24 cents is the most money we can have in dimes, nickels, and pennies using the fewest number of coins.
- This is a direct proof, not a proof by contradiction, since we do not assume a hypothesis and negate it later.

RQ3: Preliminary Results

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

Algorithm Greedy Change-Making

Input: n : positive integer.
Inputs: c_1, \dots, c_r : values of denominations of coins, where $c_1 > c_2 > \dots > c_r$.
Output: $\{d_1, d_2, \dots, d_r\}$ d_i is the number of coins of denomination c_i in the change for n .

```

1: for  $i = 1$  to  $r$  do
2:    $d_i := \lfloor n / c_i \rfloor$ 
3:    $n := n - d_i c_i$ 
4:    $d_i := d_i + 1$ 
5:    $n := n - c_i$ 
6: return  $\{d_1, d_2, \dots, d_r\}$ 
  
```

Theorem: If n is a positive integer then n cents can be paid using quarters, dimes, nickel, 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 theorem by contradiction. We will show that if we had more than the specified number of coins of each type, we could replace them using fewer coins that had the same value. We note that if we had two dimes we could replace them with a quarter and a nickel, if we had two nickels we could replace them with a dime, if we had five pennies we could replace them with a nickel, and if we had two dimes and a nickel we could replace them with a quarter. Because we can have at most two dimes, one nickel, and four pennies, but we cannot have two dimes and a nickel, it follows that 24 cents is the most money we can have in dimes, nickels, and pennies when we make change using the fewest number of coins for n cents. \square

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.

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

$n = 92$  **Optimal # of coins**

$n = 30$  **Optimal # of coins**

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

- No mistake.
- We cannot prove this theorem by contradiction. We need to perform an induction on n .
- The proof does not correctly establish that 24 cents is the most money we can have in dimes, nickels, and pennies being the fewest number of coins.
- This is a direct proof, not a proof by contradiction, since we do not assume a hypothesis and negate it later.

Algorithm Greedy Change-Making

Input: n : positive integer.
Inputs: c_1, \dots, c_r : values of denominations of coins, where $c_1 > c_2 > \dots > c_r$.
Output: $\{d_1, d_2, \dots, d_r\}$ d_i is the number of coins of denomination c_i in the change for $i = 1, 2, \dots, r$.

```

1: for  $i = 1$  to  $r$  do
2:    $d_i := 0$ 
3:   while  $n \geq c_i$  do
4:      $d_i := d_i + 1$ 
5:      $n := n - c_i$ 
6: return  $\{d_1, d_2, \dots, d_r\}$ 
  
```

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 theorem by contradiction. We will show that if we had more than the specified number of coins of each type, we could replace them using fewer coins that had the same value. We note that if we had two dimes we could replace them with a quarter and a nickel, if we had two nickels we could replace them with a dime, if we had five pennies we could replace them with a nickel, and if we had two dimes and a nickel we could replace them with a quarter. Because we can have at most two dimes, one nickel, and four pennies, but we cannot have two dimes and a nickel, it follows that 24 cents is the most money we can have in dimes, nickels, and pennies when we make change using the fewest number of coins for n cents. \square

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.

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

$n = 92$  **Optimal # of coins**

$n = 30$  **Optimal # of coins**

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

- No mistake.
- We cannot prove this theorem by contradiction. We need to perform an induction on n .
- The proof does not correctly establish that 24 cents is the most money we can have in dimes, nickels, and pennies using the fewest number of coins.
- This is a direct proof, not a proof by contradiction, since we do not assume a hypothesis and negate it later.

⇒ Students may benefit from teaching materials that facilitate perusal with ease (e.g., without requiring multiple page flips).

Formalism Comprehension: Wrapping it Up

Can we use objective measures to investigate how students read and understand computer science formalisms (i.e., **mathematical logic**)?

Formalism Comprehension: Wrapping it Up

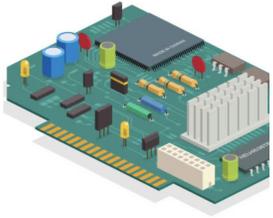
Can we use objective measures to investigate how students read and understand computer science formalisms (i.e., **mathematical logic**)?



Yes, we can!

Proposal Overview

Three components:



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



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



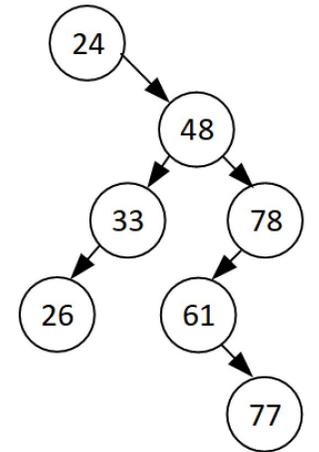
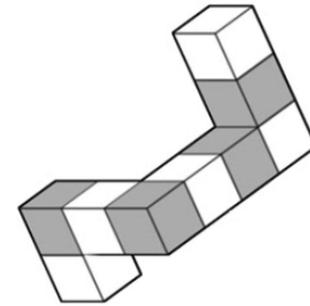
Using TMS to codify the relationship between spatial reasoning and programming



TMS for Programming

Does disrupting brain regions associated with spatial reasoning impact a programmer's ability to reason about code (i.e., **programming logic**)?

Programming and Spatial Reasoning



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 Huang¹, 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

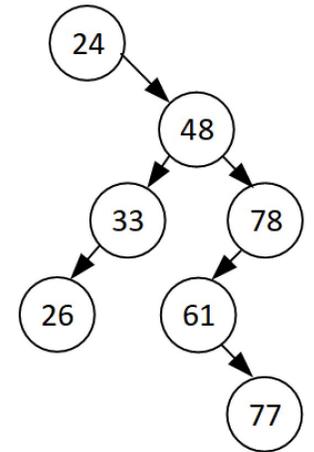
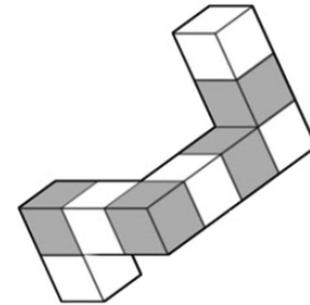
Chris Parnin
NC State University
Raleigh, North Carolina, USA

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^τ, and André Brechmann^θ



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 Huang¹, 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

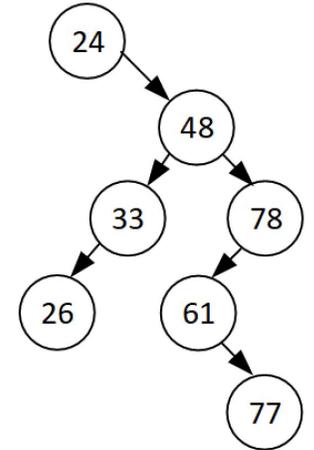
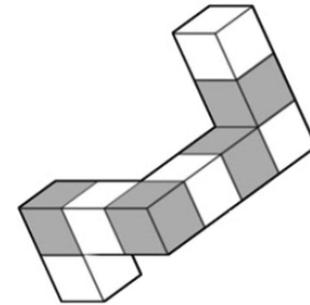
Chris Parnin
NC State University
Raleigh, North Carolina, USA

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^τ, and André Brechmann^θ

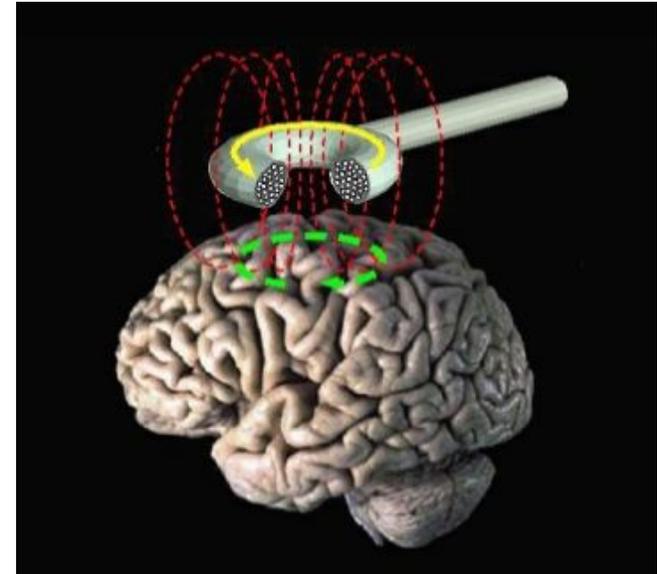


Programming and Spatial Reasoning

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

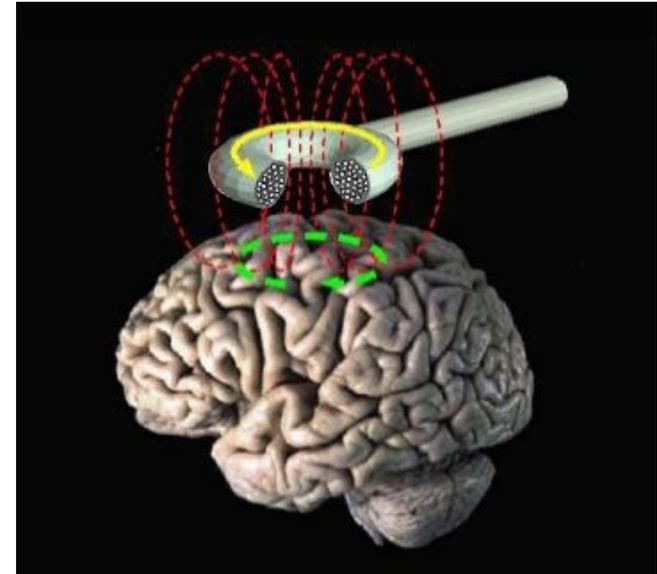
Enter: Transcranial Magnetic Stimulation

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



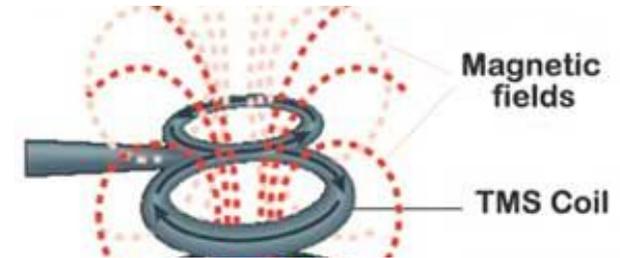
Enter: Transcranial Magnetic Stimulation

- **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** (e.g., compared to longitudinal studies)



How does TMS work?

TMS pulses produce a magnetic field around the TMS coil

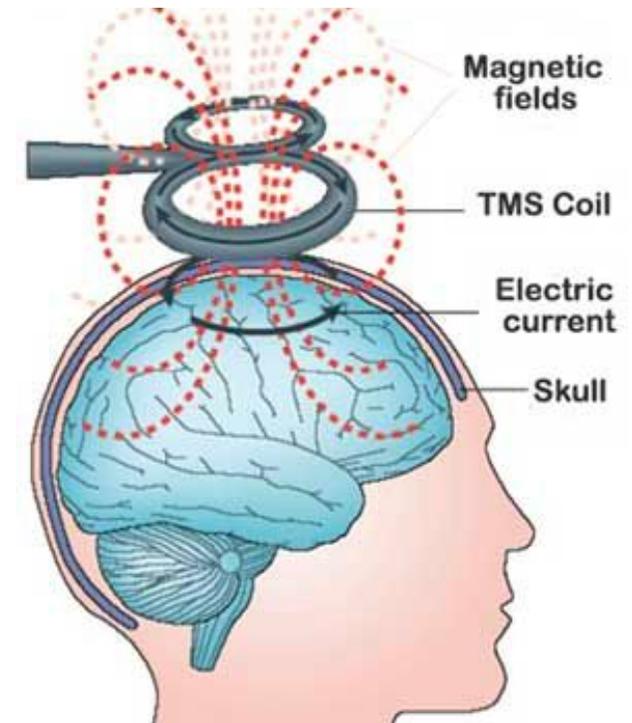


How does TMS work?

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

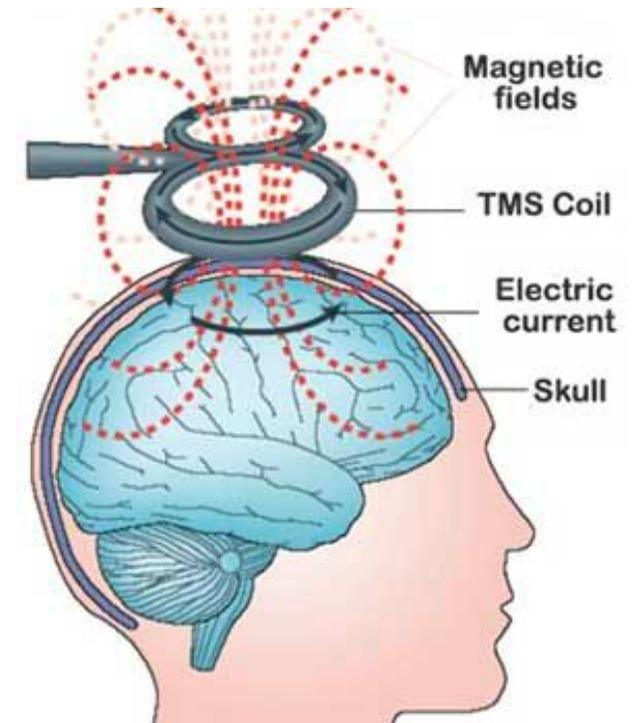


How does TMS work?

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

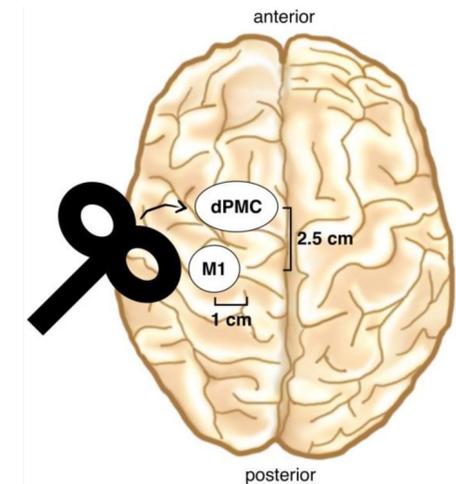
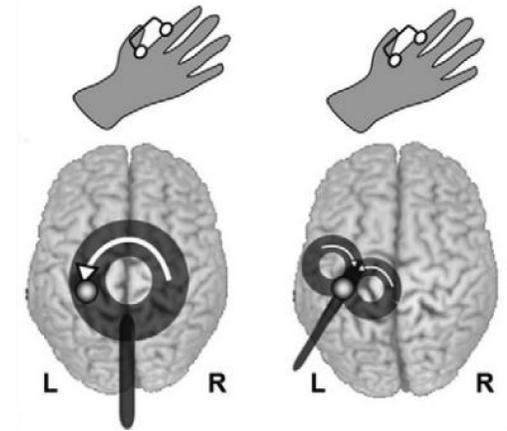


By altering activity in certain brain regions, we can investigate causal relationships between tasks and brain activity!

Localization for TMS

Problem: Identifying the location for TMS coil placement is a challenging task (e.g., due to anatomical differences in individual brains)

How do identify and precisely target brain regions for TMS treatment?



Localization for TMS

Problem: Identifying the precise location for TMS coil placement is a challenging task

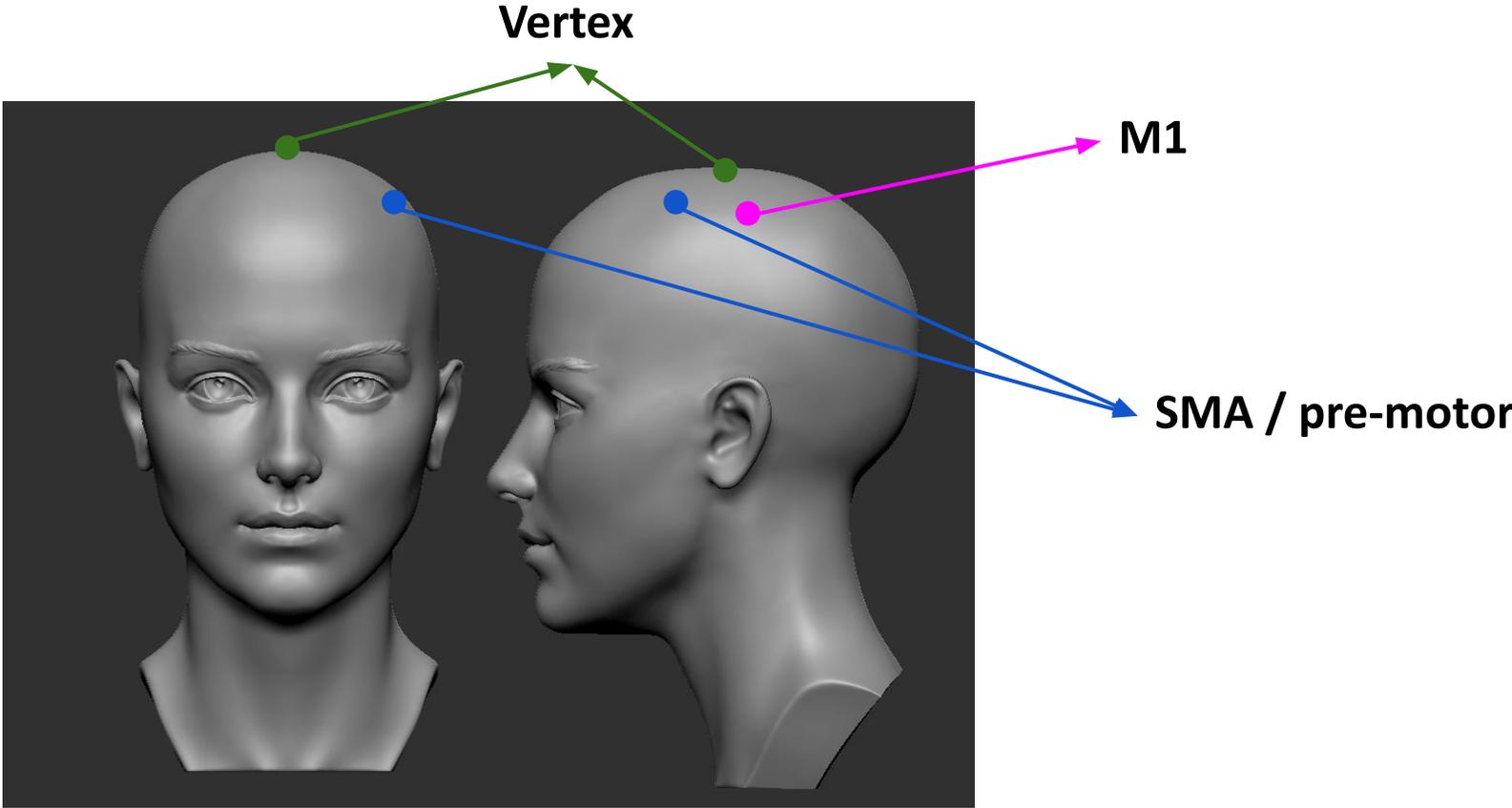
Solution: High-resolution, per-participant **brain scans** with widely-accepted, **anatomical landmark-based localization approaches** from the scientific community



TMS for Programming: Experiments and Metrics

- Two phase experimental process: **fMRI session** to obtain anatomical brain scan, followed by 2-4 **TMS sessions** (each on a different day)
- Each TMS session can correspond to treatment or control conditions
 - Treatment conditions: **supplementary motor area (SMA)** or **primary motor cortex (M1)**, both responsible for motor actions and associated with spatial reasoning
 - Control condition: **cranial vertex** region, not associated with spatial reasoning

TMS for Programming: Brain Regions



TMS for Programming: Protocol

- Per-participant **active motor threshold** (AMT) to obtain the “lowest stimulation intensity” that still influences brain activity
- Each TMS session conducted using an off-the-shelf, widely-used **continuous theta-burst stimulation (cTBS) protocol**
 - 3 pulses of stimulation at 50 Hz, repeated every 200ms, for a total of 600 pulses in **40 seconds**
 - cTBS applied at **80% AMT** to comply with commonly-accepted safety standards

TMS for Programming: Stimuli Design

Data structure manipulation (including arrays, linked lists, 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:

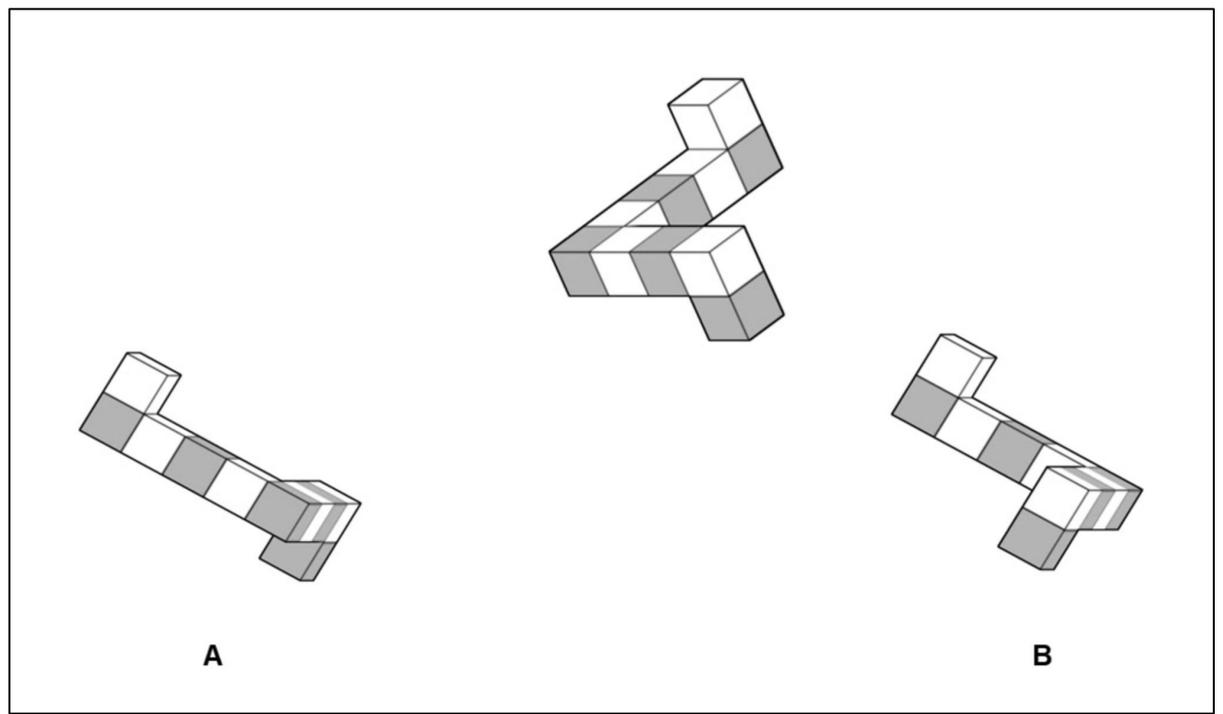
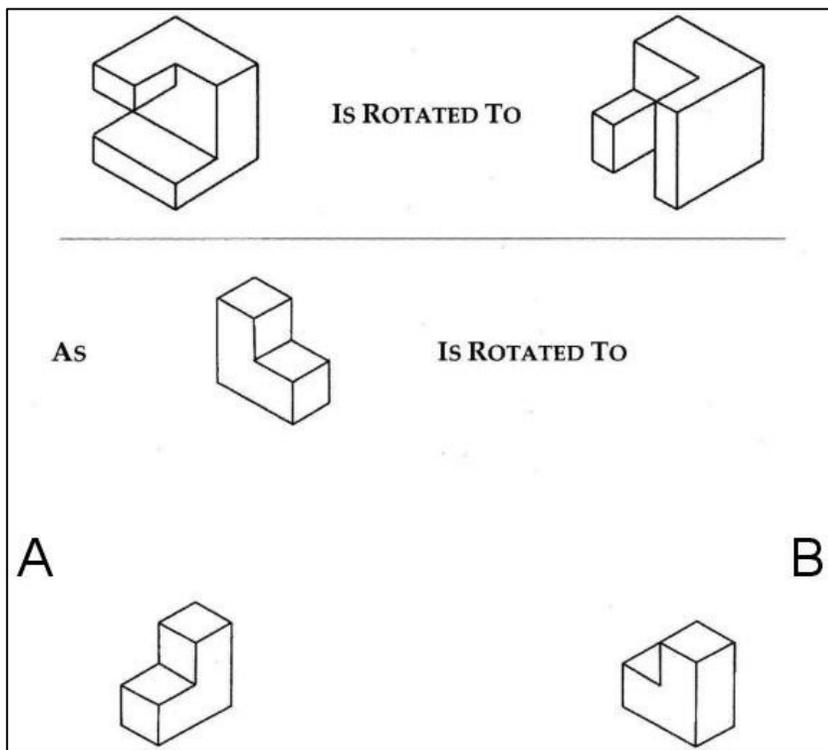
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
9	78	53	21	11	63	98	1	82	39	90	54	68	15	13

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: Stimuli Design

Mental rotation



TMS for Programming: Stimuli Design

Code comprehension (including tracing code, analyzing complexity)

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: Experiments and Metrics

- After each cTBS session, participants work on study stimuli in front of a regular computer (i.e., in a more ecologically valid setting)
 - **35 minutes** of study stimuli
- Participant performance will be evaluated by looking at changes in **response times** and **accuracies** as a result of TMS treatment

TMS for Programming: RQs

Three RQs:

- Does disrupting brain regions associated with spatial reasoning affect a programmer's **ability** to correctly reason about code?
- Does disrupting brain regions associated with spatial reasoning affect the **time taken** for a programmer to reason about code?
- How does **demographic information** (e.g., incoming preparation or expertise) mediate the effect of TMS on task outcomes?

TMS for Programming: Project Status

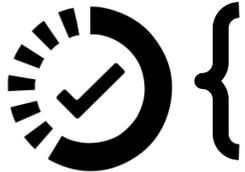


- Internal grant of \$17,000 from the University of Michigan Functional MRI Lab
- IRB approval (**HUM00216195**)
- Stimuli design
- TMS safety training
- Research access to fMRI and TMS equipment

TMS for Programming: Project Status



- Internal grant of \$17,000 from the University of Michigan Functional MRI Lab
- IRB approval (**HUM00216195**)
- Stimuli design
- TMS safety training
- Research access to fMRI and TMS equipment



- Participant recruitment
- Data analyses

TMS for Programming: Wrapping it Up

Does disrupting brain regions associated with spatial reasoning impact a programmer's ability to reason about code (i.e., **programming logic**)?

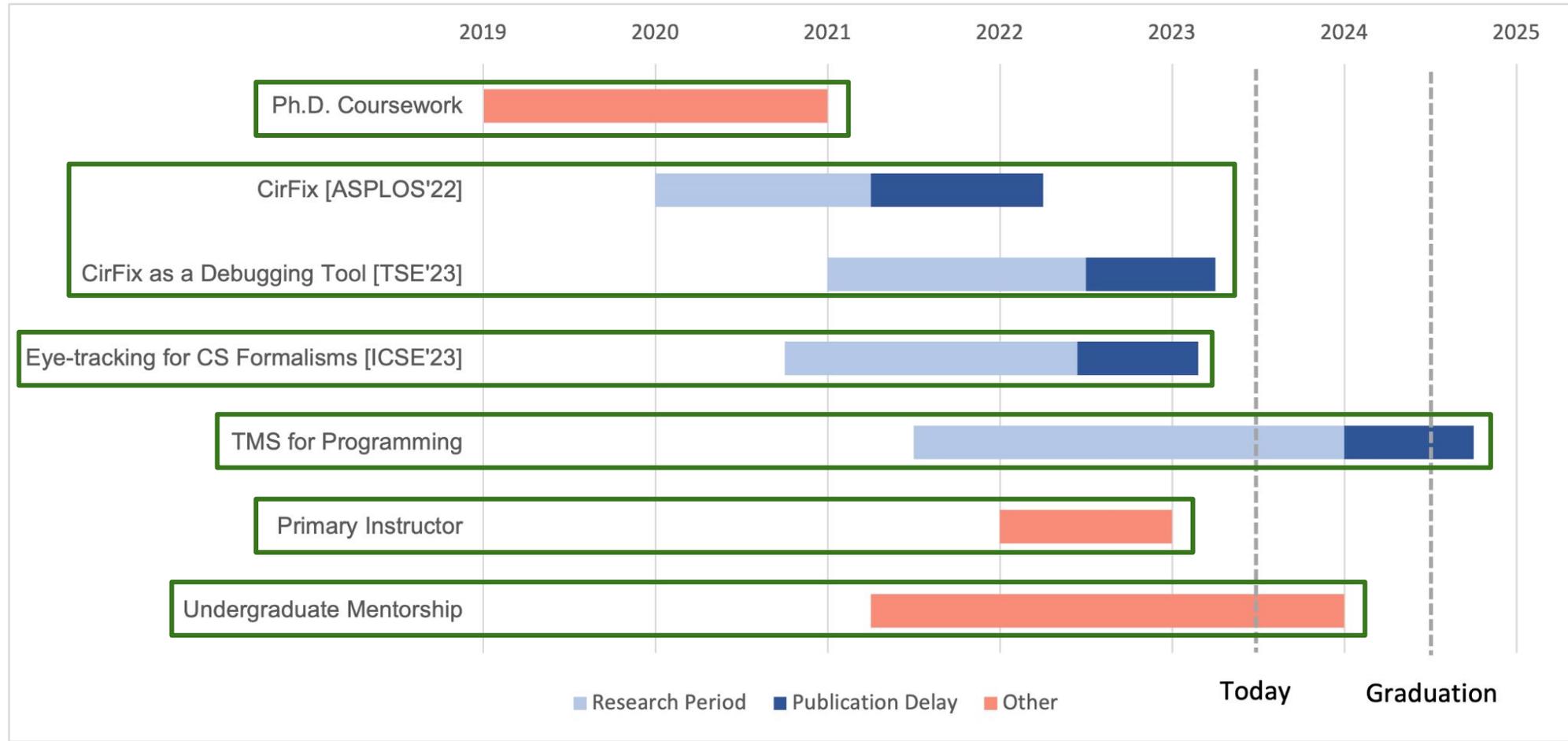
TMS for Programming: Wrapping it Up

Does disrupting brain regions associated with spatial reasoning impact a programmer's ability to reason about code (i.e., **programming logic**)?



???

Ph.D. Timeline



Relevant Publications

1. **CirFix: Automated Hardware Repair and its Real-Word Applications.** Priscila Santiesteban, Yu Huang, Westley Weimer, Hammad Ahmad. *TSE (2023)*. [9.22 impact factor]
2. **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)*. [26% acceptance rate]
3. **CirFix: Automatically Repairing Defects in Hardware Design Code.** Hammad Ahmad, Yu Huang, Westley Weimer. *ASPLOS (2022)*. [20% acceptance rate]
4. **Applying Automated Program Repair to Dataflow Programming Languages.** Yu Huang, Hammad Ahmad, Stephanie Forrest, Westley Weimer. *GI Workshop @ ICSE (2021)*.

Other Publications

1. **LOGI: An Empirical Model of Heat-Induced Disk Drive Data Loss and its Implications on Data Recovery.** Hammad Ahmad, Colton Holoday, Ian Bertram, Kevin Angstadt, Zohreh Sharafi, Westley Weimer. *PROMISE (2022)*.
2. **Digging into Semantics: Where Do Search-Based Software Repair Methods Search?** Hammad Ahmad, Padraic Cashin, Stephanie Forrest, Westley Weimer. *PPSN (2022)*.
3. **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)*. [16% acceptance rate]
4. **A Program Logic to Verify Signal Temporal Logic Specifications of Hybrid Systems.** Hammad Ahmad, Jean-Baptiste Jeannon. *HSCC (2021)*. [35% acceptance rate]

Broader Impact

Mentorship

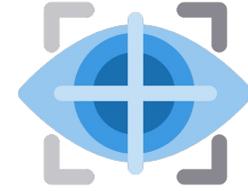
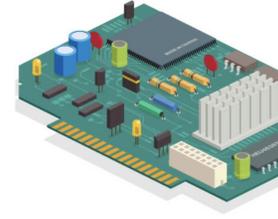
- **Undergraduate involvement** in research activities (particularly from groups **underrepresented in CS**)
 - Mentored four undergraduate / non-traditional students on research activities included in this proposal
 - Written text for an NSF REU proposal (that was funded for \$8000 total) to fully support an additional undergraduate student

Broader Impact

Pedagogy

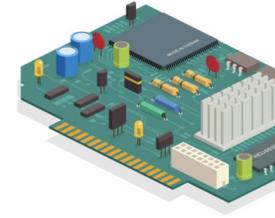
- **Recommendations for educators and suggestions for pedagogical intervention studies**
 - Approached by an educator at the University of Michigan to deploy a hardware debugging assistant in a classroom setting
 - Approached by an educator at the University of Washington to use preliminary eye-tracking results for undergraduate theory
- **Accounting for incoming preparation in proposed research**
 - Results and suggestions from our research can be applied to a wider variety of student groups with varying levels of preparation for the CS major

Proposal Summary



- **Three components:**
 - Using automated program repair for hardware as a debugging assistant for designers
 - Using eye-tracking to understand cognition for computer science formalisms
 - Using TMS to codify the relationship between spatial reasoning and programming

Proposal Summary



- **Three components:**
 - Using automated program repair for hardware as a debugging assistant for designers
 - Using eye-tracking to understand cognition for computer science formalisms
 - Using TMS to codify the relationship between spatial reasoning and programming
- **Thesis statement:**
 - We can use objective measures to obtain mathematical models of logical cognition, and these models can accurately explain student behavior.
 - **Obtaining such an understanding may impact how educators better teach logical reasoning to students.**