The Affordances and Constraints of Diagrams on Students' Reasoning about State Machines

Geoffrey L. Herman*
University of Illinois at Urbana-Champaign
201 N. Goodwin Ave
Urbana, IL 61801
glherman@illinois.edu

Dong San Choi
University of Illinois at Urbana-Champaign
1308 W. Main St.
Urbana, IL 61801
choi88@illinois.edu

ABSTRACT

While the concept of state is foundational to computing, students possess a myriad of misconceptions about it and the role it plays within computing systems. Research on students' misconceptions reveals that their ability to use conceptually appropriate information varies based on the task they are performing and the representational tools they are provided. Critically, the tacit information in these representations influences this process, hindering or helping students. In this paper, we present a qualitative research study, in which we interviewed 24 students as they transformed finite state machines into synchronous, sequential logic circuits. We found that students generally had profound skill with procedures. However, their ability to reason about the four components of state, next-state, inputs, and outputs, were constrained by the representations that they were given or created themselves. Conversely, the order in which students produced their drawings provided complementary insights into students conceptual understanding. These findings revealed that students possess conceptions of computers as input-output systems rather than state-based systems. We suggest potential interventions and future research based on these findings.

CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; • **Hardware** → **Finite state machines**; **Sequential circuits**; • **Human-centered computing** → *Visualization systems and tools*;

KEYWORDS

State, Misconceptions, Visual Representations, Diagrams, Finite State Machines

1 INTRODUCTION

The concept of state and finite state machines (FSMs) undergirds computer architecture and programming practices [7, 8, 18, 22]. Some have even argued that the concept of state is a threshold concept, a prerequisite concept without which students cannot truly understand how a computer operates [17, 20]. From this perspective, a computer has two fundamental operations, it 1) stores states and 2) manipulates those states in response to the systems' current state and inputs from users or other devices. This state information can also be used to derive system-level outputs to be used by other devices or users. While the concept of state is foundational to computing, students possess a myriad of misconceptions about state and how information is stored in a computer [15, 21].

In this paper, we examine how students reason about FSMs in the context of digital logic and synchronous, sequential logic circuits. State can be thought of functionally as all stored bits in a digital circuit [3, 27] or it can be thought of conceptually as all the necessary information about the past to account for a system's future behavior [28]. Prior studies have documented that students reveal multiple (as many as seven), mutually exclusive, conceptions of state when reasoning about digital logic circuits and computer architecture [15]. Students consider the state of the circuit to be its input bits, output bits, or some combination of those with the stored bits. It has been posited that state may be hard to learn, in part because instructors have so internalized the concept of state that they fail to make it explicit [7, 20].

The tacit nature of the state concept suggests that we should explore the ways in which state concepts are tacitly encoded within the representations used to teach finite state machines and their physical implementations, sequential logic circuits. In line with this, research on spatial reasoning and visual representations underscores that the tacit vs. explicit information that is presented in diagrams affects what information novices are able to learn [11]. We present a qualitative research study in which we interviewed students as they transformed FSM state diagrams into synchronous, sequential logic circuits. We specifically investigate two research questions, 1) What information about state is explicitly or tacitly encoded within traditional representations of finite state machines? and 2) How does this explicit versus tacit information affect how students reason about state?

2 LITERATURE REVIEW

We review literature on spatial reasoning and the effect of visualizations on students' reasoning. We then provide a brief background on FSMs and synchronous sequential logic circuits.

*Corresponding Author

ICER '17, August 18-20, 2017, Tacoma, WA, USA.

978-1-4503-4968-0/17/08…$15.00
DOI: http://dx.doi.org/10.1145/3105726.3106172

2.1 Making Vital Information in Diagrams Explicit Improves Students' Performance

Spatial reasoning skills are one of the strongest predictors of students' success in Science, Technology, Engineering, and Mathematics, including computer science [26]. The visual representations that we provide students can effect their ability to access the right concepts or procedures at the appropriate times [10, 11]. Hegarty argues that part of this challenge derives from the fact that visual representations of information (i.e., displays) tacitly encode vital information about how to interpret the display and thus require external conceptual understanding to be interpreted [4, 11]. For example, an arrow in a display may indicate motion of an object, draw attention to an object, indicate a force vector, or something entirely different [13]. Knowing which interpretation is correct likely depends on knowledge that is not encoded in the display. Additionally, a novice must also know when to appropriately aggregate objects into meta-objects or disassemble what appears to be a single object into several smaller objects.

Consider how chess experts are able to more quickly memorize the arrangement of chess pieces mid chess game but are unable to replicate this performance gain when the chess pieces are arranged randomly [5]. These experts have learned how to see configurations of pieces based on what known strategies those configurations enable rather than simply perceiving the individual pieces. Conversely, consider the challenges that students have when solving classical mechanics problems. When students are learning how to construct free body diagrams, they struggle to identify when/how to group multiple distinct bodies into a single "free body [24]."

Recent studies into students' interactions with displays has revealed that displays that are informationally equivalent to experts can lead to dramatically different student performance on what appear to be isomorphic tasks [10]. Consider the task of adding or subtracting two vectors. Vectors are often taught using three different types of displays: arrows with a magnitude on a grid, arrows with a magnitude and an angle from a reference point, or $\hat{i}, \hat{j}, \hat{k}$ format (i.e., $\vec{x} = 6\hat{i} + 8\hat{j}$). Each of these displays is informationally equivalent in that all information needed to derive one display is available in the other displays.

However, these displays make different information more or less explicit. The arrow magnitude displays make the magnitude and direction of the arrows explicit but requires the viewer to extract the component parts of the vector. In contrast, the $\hat{i}, \hat{j}, \hat{k}$ format makes the component parts of a vector explicit but the magnitude and direction implicit. When physics students were given these different types of displays, all students were able to reliably perform vector addition (e.g., 90% accuracy) when given $\hat{i}, \hat{j}, \hat{k}$ format [10]. In contrast, while the strong students could still reliably perform vector addition with the other display types, weaker students' performance dropped precipitously (50% with the magnitude and angle format and 25% with the magnitude and grid format). Because adding vectors depends on adding the vector components, the $\hat{i}, \hat{j}, \hat{k}$ format best helps students perform [10]. Parallel findings exist in other disciplines such as organic chemistry [12, 25].

There is a growing number of tools for visualizing how programs run and Sorva et al [23] provide an excellent review of these visualizations and their usefulness in helping students understand their code. For example, providing structured visualizations for tracing code has been found to improve students' ability to code and trace code [16]. Other studies have suggested that RAM diagrams that illustrate how memory is allocated may improve students' learning more than traditional trace tables [19].

2.2 Finite State Machine Displays

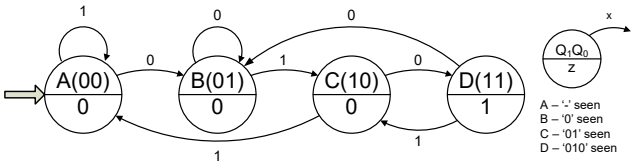
Digital logic instruction frequently progresses through a sequence of teaching a set of analysis tools (visualizations) that enable students to transform FSM *state diagrams* to *circuit diagrams* (See textbooks such as [3, 27, 28] for examples of this approach). Examples of these displays are illustrated in Figure 1). In this progression, students are first taught how to transform tabular representations such as truth tables (display 2 but without state variables) and Karnaugh maps (K-maps) into Boolean logic expressions (display 3) in the context of combinational circuits (stateless, input-output circuits). They are also then taught how to transform these Boolean logic expressions into logic gates in a circuit diagram (see the D-shaped and triangle-shaped objects in display 4). Finally, students are taught how to interpret state diagrams (display 1) and how to transform those diagrams into tabular representations.

Once students have learned how to use each display, they are expected to be able to transform first from a (1) state diagram to a (2) tabular representation, then to a (3) Boolean logic expression, and finally to a (4) circuit diagram in that order. Notably, each of these displays is informationally equivalent like the vector representations. However, it is generally easier to transform between adjacent displays (i.e., transforming between displays 1 and 2 or 3 and 4) than between distant displays (transforming between displays 1 and 3 or 2 and 4). Arguably, the rationale for even introducing all four display types is to make the process of translating from state diagrams to circuit diagrams a mechanistic one that reduces the likelihood of errors relative to translating directly from state diagrams to circuit diagrams without the intermediate displays.

As described earlier, a FSM does two things: store state and manipulate state. State diagrams illustrate this concept (see Figure 2) by displaying states as "bubbles" (e.g., circles labeled with letters). If the system is in a current state (e.g., state A), then the system will transition to a next state according to the value of the system input (the next-state will be A if the input is 0 and the next-state will be B if the input is 1). We will call this the "origin+transition=destination" algorithm. In a Moore model state machine, the output of the system is a function of the current state (e.g., the system output is 0 in state A and the system output is 1 in state B). To create a circuit implementation of the state machine, the state needs to be encoded into a set of bits (frequently represented using a vector of Q_i variables, see variable key in Figure 2). A minimal encoding will need $\lceil log_2 n \rceil$ state bits to encode n states. The system inputs and the system outputs are also encoded with bits.

The system can be represented using tabular representations or a system of Boolean expressions. The system input bits and state bits (e.g., x and Q_1Q_0 respectively in Figure 1 displays 2 and 3) are treated as inputs in these representations and the system output and the system's next-state (e.g., z and $Q_1^+Q_0^+$ respectively) are treated as outputs (i.e., the state and system inputs are treated as independent variables, while the next-state and system outputs are

Display 1: State Diagram



Display 2: Next-state Table

Current State		Input	Output	Next State	
Q_1	Q_0	x	z	Q_1^+	Q_0^+
0	0	0	0	0	1
0	0	1	0	0	0
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	0	1	1
1	0	1	0	0	0
1	1	0	1	0	1
1	1	1	1	1	0

Display 3: Boolean expression

$$\begin{aligned} Q_1^+ &= Q_0 x + Q_1 \bar{Q}_0 \bar{x} \\ Q_0^+ &= \bar{x} \\ z &= Q_1 Q_0 \end{aligned}$$

Display 4: Circuit Diagram

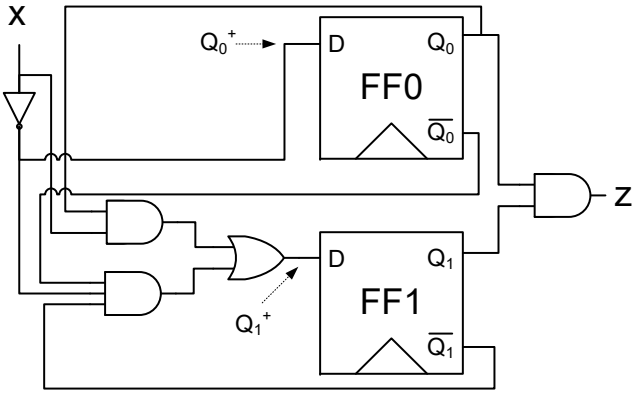


Figure 1: Examples of four displays used to teach finite state machines. Students are expected to be able to transform from the state diagram (display 1) through the other displays to design a circuit diagram (display 4).

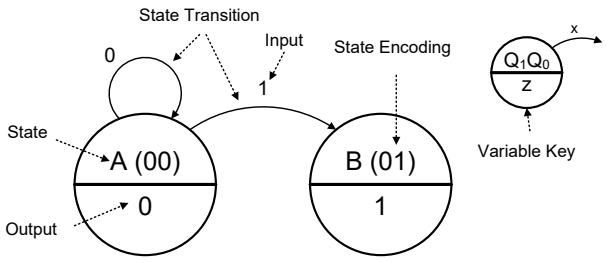


Figure 2: An example partial state diagram. Dotted arrows highlight which parts of the diagram correspond to which components of the state system.

Current state		Input	Next state (FF input)	
Q_1	Q_0	x	Q_1^+ (D_1)	Q_1^+ (D_0)
0	0	0	m_0	n_0
0	0	1	m_1	n_1
0	1	0	m_2	n_2
0	1	1	m_3	n_3
1	0	0	m_4	n_4
1	0	1	m_5	n_5
1	1	0	m_6	n_6
1	1	1	m_7	n_7

Q_1^+

x \ Q_1Q_0	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

Q_0^+

x \ Q_1Q_0	00	01	11	10
0	n_0	n_1	n_3	n_2
1	n_4	n_5	n_7	n_6

Figure 3: A next-state table (left) and two equivalent Karnaugh maps (right). m_i and n_i identify which rows of the table correspond to which cells in the Karnaugh maps.

dependent variables). The next-state variables have a superscript + to indicate the discrete-time-dependent relationship between the current state and next-state variables in sequential circuits.

Figure 1 display 2 shows a next-state table as an example tabular representation. A K-map is an alternate tabular representation that facilitates deriving minimal Boolean expressions (Boolean expressions with fewer operators and variables). There is a one-to-one mapping between the rows of a next-state table and the cells of a K-map (See Figure 3). A K-map is intended to be used with only one output variable at a time. While new outputs in a next-state table can be appended to an existing table by adding columns on the right, a new K-map needs to be drawn for each new output variable being analyzed.

In synchronous sequential circuits, the state encoding bits are stored in flip-flops (see boxes labeled FF in Figure 1). The current state of the system is the output of the flip-flops (e.g., Q_1, Q_0 in Figure 1 display 4). The system input is usually drawn on the left (e.g., x) and the system output on the right (e.g., z). The state bits are combined with the system input using logic gates to calculate the next-state of the system (e.g., the D inputs to the flip-flops).

3 METHODS

Because we knew of no prior research that specifically examined students' interactions with the aforementioned displays, we conducted a qualitative research study to develop rich descriptions of students' interactions. We used standard cognitive interview techniques and analyzed these interviews using the Constant Comparative Method [1].

3.1 Interview Protocol

During the interview, subjects were given a simple state diagram (see left diagram in Figure 4 in Section 4.1) for a sequential comparator that compares two streams of system input bits and sends a system output which indicates which stream has a greater value when interpreted as an integer. They were then asked to design a sequential circuit based on the diagram. Subjects were not told what approach to use during this task. After they finished designing the sequential circuit, subjects were then asked to draw the state diagram for a counter FSM (a canonical FSM that all subjects had previously been assigned to design in their coursework). Rather

than let subjects design the sequential circuit for the counter FSM using the approach outlined in Figure 1, subjects were asked to simply sketch the final sequential circuit diagram using black boxes wherever needed (e.g., using a black box in place of logic gates for calculating the system output). Subjects were told to skip to the sequential circuit in the interest of time and because we had already seen how the subjects executed that process. This forced change in procedure enabled us to examine what type of information subjects could extract when transforming directly from a state diagram to a circuit diagram versus going though the series of three transformations (i.e., jumping from display 1 to 4 instead of stepping through from display 1 to 2 to 3 to 4).

3.2 Sampling and Data Collection

In Fall 2014 and Spring 2015, the authors interviewed 11 and 13 undergraduate students, respectively, who had just completed a large enrollment course in digital logic and computer architecture at Research University. Interviews were conducted until the interviewers perceived that each new interview provided no new insights into students' problem solving processes. The course taught FSMs using a method similar to that described in Section 2.2. All interviewed students (i.e., subjects) were traditional-age (18-22 years old), first- or second-year undergraduates majoring in electrical or computer engineering. The sample had 7 females and 17 males, reflecting the demographics of the department (14% female students). Students were recruited via e-mail solicitation after completing the aforementioned course.

Subjects were interviewed for one hour. Interviews were conducted in a think-aloud format: Subjects were instructed to vocalize their thoughts as they solved problems and responded to questions. They were told not to expect feedback during the interviews about whether their designs were valid, but to expect frequent requests to elaborate on what they were doing.

Interviews were conducted on a tablet computer running Windows 8. Subjects wrote in Microsoft OneNote using a digitizer stylus to draw. Interviews were recorded using Camtasia for screen capture and audio recording. Subjects were given a brief training exercise to familiarize them with the tablet computer and stylus interface. Subjects were paid $10 for their participation, and all subjects gave written consent to be interviewed under IRB approval.

3.3 Analysis

Interviews were analyzed using the Constant Comparative Method without an a priori coding scheme [1, 6], but with a structured method for ensuring that specific types of comparisons were made at each stage of the analysis process. Analysis was conducted by a team of two researchers. The researchers first analyzed the interviews independently to increase their familiarity with the data as well as to provide a record of personal observations and biases. The researchers directly coded (a qualitative research process for tagging data) the video data using MaxQDA. Team members then collaboratively reconciled codes and interpretations of subjects' statements and actions.

Analysis and comparisons were made along three units of analysis of different granularities: 1) the problem, 2) the transformation and 3) the statement.

Problem unit of analysis: Each subjects' approach to solving each problem was also described holistically. For example, did a subject invoke the concept of state when performing the procedures described in Section 2.2 during this problem? Did the subject analyze the system output bits separately from the next-state bits during this problem?

Transformation unit of analysis: We coded at what times each subject was transforming from one display to a different display. Codes included things like "State Diagram to Next-State Table" or "Boolean to Circuit."

Statement unit of analysis: Each statement a subject made or figure they drew was analyzed to document a subjects' conceptual understanding as revealed in the moment. For example, a subject might refer to the output of the circuit (O) as the next state (Q^+) revealing a contextual conflation of the two concepts.

The goal of the Constant Comparative Method is to make comparisons within and across these units of analysis to collect evidence for and against emergent themes and theories. In this analysis approach, the researchers focus on maintaining thick descriptions of observations to facilitate comparisons for as long as possible before reducing these observations to a strict coding scheme. After analyzing each of the interviews once, the researchers developed and finalized a coding scheme that captured core observations. The researchers applied this coding scheme to the data independently before comparing and reconciling any disagreements. The researchers had 92% inter-rater agreement after their independent coding and were able to reconcile all disagreements.

4 RESULTS

In accordance with our Constant Comparative Method, we first present our findings organized by the transformations that subjects performed during the interviews and the order in which they performed them. While the results are organized according to the transformations, the results draw upon the statement and problem units of analysis. We provide comparative analysis across transformations to explore effects of the tacit and explicit information in the different displays.

4.1 Transformation: State diagram to tabular representations

All subjects immediately attempted to transform the state diagram into a tabular representation, although one subject stopped almost immediately and began drawing his circuit diagram directly from the state diagram. Everyone was able to correctly articulate the "origin+transition=destination" algorithm (see Section 2.2). However, five of these subjects failed to use the tacit Q_i^+ variables for next-state and instead used the explicit c_i output variables (See right diagram in Figure 4). Additionally, three of the subjects who did use the tacit Q_i^+ variables applied the "origin+transition=destination" algorithm to both their next-state and output variables. In total, 8 of 23 subjects who completed a tabular representation mistakenly treated their output variables as if they were next-state variables.

Of the 23 subjects who used tabular representations, 18 first transformed the state diagram into a next-state table. Fifteen of these subjects then transformed their next-state table into a K-map (see Figure 3) while the other three began transforming their next-state

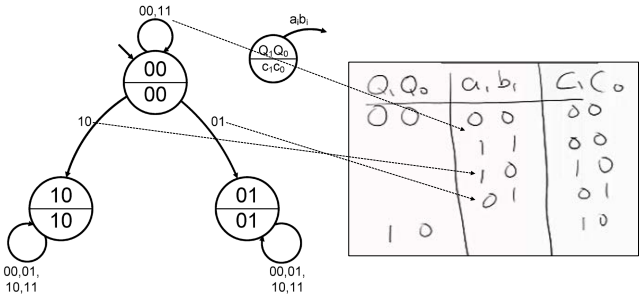


Figure 4: Left: Finite state diagram for a bit-serial comparator. Right: A subject's next-state table derived from the state diagram. Dotted arrows show how the subject transferred information from the state diagram to the next-state table.

tables into Boolean expressions or circuit diagrams. Five subjects first transformed the state diagram into a K-map directly, skipping the next-state table entirely. Notably, three subjects explicitly transformed their state diagram into a tabular representation twice, first transforming into a next-state table, then erasing the table, and then transforming into a K-map.

4.1.1 Transformation: State diagram to next-state table. When transforming the state diagram in Figure 4 to a next-state table, 16 of 18 subjects finished their next-state table (one switched to a K-map partway while the other skipped directly to drawing her circuit diagram). Of these 16 subjects, eight did not analyze the tacitly encoded state 11 (note: because the system uses 2 bits to encode the state, there will actually be four distinct states (2^2) in the physical circuit even if one of them is unreachable by the FSM algorithm). When drawing their next-state tables, some subjects drew a table with 16 rows with every possible encoding before performing their transformation while others drew their next-state tables as they analyzed the state transitions (notice how the next-state table in Figure 4 does not have any values listed for the inputs $a_i b_i$ for state 10). Only subjects who drew their next-state tables as they analyzed the state transitions forgot the 11 state. Subject created a new row in their next-state table for every state transition they analyzed. When there were no more state transitions to analyze, these subjects stopped creating new rows in their next-state table.

When performing this transformation, 7 of 18 subjects analyzed the state transitions in natural reading order, writing that the system would return to state 00 after an input of either 00 or 11 in state 00 (see dotted arrows in Figure 4). These subjects then analyzed the state transition from state 00 with input 10 and then the state transition with input 01. The remaining subjects use some variant of numeric order for analysis, analyzing the state transitions from $Q_1Q_0a_ib_i = 0000$ (binary 0) to $Q_1Q_0a_ib_i = 1111$ (binary 15).

4.1.2 Transformation: State diagram to Karnaugh map. Nineteen subjects used a K-map. When performing this transformation, only 1 out of 19 subjects did not analyze the tacitly encoded state 11. Many subjects even exclaimed something to the effect of "Oh! that's right. There's a 11 state" when performing this transformation. When drawing their K-maps, all 19 subjects fully drew all 16 cells of the K-map before filling in any of them cells.

When performing this transformation, 7 of 19 subjects analyzed the state transitions in natural reading order.

4.1.3 Comparing "State diagram to next-state table" with "State diagram to Karnaugh map". Only 50% of subjects analyzed the tacit 11 state when creating a next-state table but 95% of subjects analyzed the tacit state when creating a K-map. The one subject who failed to analyze the tacit 11 state was also the only subject to incorrectly draw a K-map, making a 4×3 grid rather than the required 4×4 grid. Notably, when using a K-map, the subjects always enumerated all possible combinations of state and input variables before beginning analysis, whereas when using a next-state table, more than half of the subjects enumerated combinations of state and input variables as they analyzed the state diagram.

Similar percentages of subjects used reading order analysis in both types of transformations (39%*vs*.37%). However, five subjects revealed different analysis orders when transforming to a next-state table than when they transformed to a K-map. Four of these subjects used reading order analysis when transforming to a next-state table, but switched to a numeric order analysis when transforming to a K-map. Only one subject did the opposite.

4.2 Transformation: Tabular displays to Boolean expressions

Most subjects (22 of 24) transformed a tabular display into a Boolean expression. Of these subjects, 19 (86%) correctly performed these transformation. Two subjects could not remember the process for performing the transformation and did not write any Boolean expressions, despite efforts to remember. The remaining subject forgot to add NOT operators to two variables during the transformation (a mistake similar to changing the sign of a number).

4.3 Transformation: Tabular to circuit diagram

The two subjects who could not create Boolean expressions, both transformed their tabular representations into circuit diagrams. Both subjects used black boxes instead of logic gates and attempted to draw flip-flops but could not fully remember how they were drawn. Both subjects drew their approximations of a flip-flop first and then drew their next-state logic black boxes, progressing from left-to-right.

4.4 Transformation: Boolean expressions to circuit diagram

All 20 subjects who wrote Boolean expressions were able to correctly transform those expressions into a set of logic gates that implemented those expressions. However, seven of these subjects did not draw flip-flops as part of their circuit diagrams. Another two subjects attempted to draw flip-flops but drew them incorrectly.

When the 11 subjects who included correctly drawn flip-flops in their circuits drew their state diagrams, they began by drawing the flip-flops first (See Figure 5). After drawing the flip-flops, these subjects then transformed their Boolean expressions into logic gates. These subjects exhibited a "flip-flops to next-state logic" drawing order (See Figure 5). In contrast, eight of the nine subjects who failed to draw flip-flops correctly or at all, used a "left-to-right"

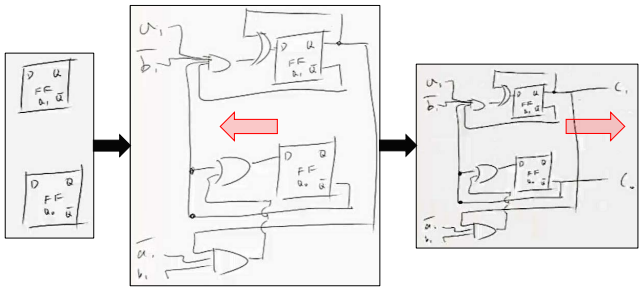


Figure 5: Example of a subject drawing their circuit diagram with flip-flops first. Left panel: Subject draws flip-flops. Middle panel: subjects adds next-state logic. Right panel: subject adds system outputs as dependent on current state.

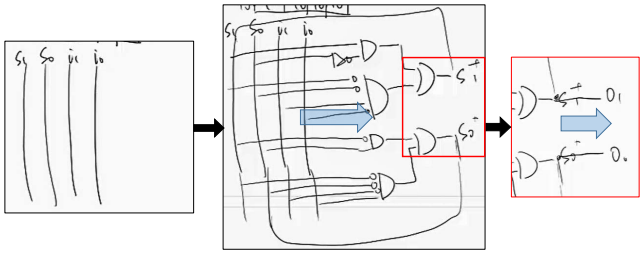


Figure 6: Example of a subject drawing their circuit diagram left-to-right without flip-flops. Left panel: Subject draws a set of rails, treating state (S_1S_0) and input (i_1i_0) variables as inputs. Middle panel: subject adds logic gates. Right panel: subject does not add flip-flops and adds system outputs as dependent on next state.

drawing order (see Figure 6). These subjects treated the independent variables of their Boolean expressions (state and system input bits) solely as inputs to their Boolean expressions. Some subjects even assigned state variables to rail notations, a notation typically reserved for system inputs (see the long vertical lines in Figure 6), explicitly treating state variables the same as system inputs. The one subject, who failed to draw flip-flops and did not use "left-to-right" drawing order, treated the OR logic gate (the D-shaped object with a curved concave left side) that would have been the fed into a flip-flop as if it was a flip-flop. He exhibited an "OR gate to next-state logic" drawing order.

When subjects failed to draw flip-flops, they labeled their current state variables on the left and their next-state variables on the right. Because the next-state variables were on the right, subjects would assign their output variables to be the same as or dependent on their next-state variables (see last panel of Figure 6) instead of being dependent on their current-state variables.

4.5 Comparing "state diagram to tabular" and "Boolean to circuit"

In the "state diagram to tabular" transformation (Section 4.1) and "Boolean to circuit" transformation (Section 4.4), we observed that some subjects either treated their next-state variables as equivalent to their output variables or they treated their output variables as dependent on their next-state variables (see Figures 4 and 6).

4.6 Transformation: State diagram to circuit diagram

Only 21 of 24 subjects reached the final portion of the second interview question, translating directly from the state diagram to a circuit diagram. Three subjects got stuck when drawing the counter FSM (interview question 2). Of the 21 subjects who performed the state diagram to circuit diagram transformation, 20 included flip-flops in their circuit diagrams. The remaining subject simply described his circuit as a black box with logic gates inside. Of the 20 subjects who included flip-flops, 15 used the "flip-flops to next-state logic" drawing order. Two subjects began by drawing next-state logic black boxes before indicating that the current state and inputs would be fed into this black box. These subjects then showed that their next-state logic boxes fed into flip-flops. The remaining three did not draw flip-flops correctly and used a "left-to-right" drawing order.

Seven subjects also spontaneously performed transformations from state diagrams to circuit diagrams during the first, undirected problem. These subjects used the state diagram to determine that the final circuit would need flip-flops and immediately drew those flip-flops. Two of these subjects finished drawing these circuit diagrams directly from the state diagram. The remaining five subjects abandoned or erased their initial circuit diagrams and switched to transforming the state diagram into a tabular representation.

4.7 Comparing "State diagram to circuit" and "Boolean to circuit"

When transforming Boolean expressions into circuit diagrams, 35% of subjects failed to remember to use flip-flops in their final circuit diagrams. Notably, 2 of the 7 subjects who failed to remember flip-flops in their final circuit diagrams were also unable to design a counter FSM and could not progress to the final part of the interview. In contrast, only 5% of subjects failed to remember to use flip-flops when transforming state diagrams into circuit diagrams. Further 4 of 5 subjects who forgot to draw flip-flops during the Boolean to circuit transformation remembered to add flip-flops during the state diagram to circuit transformation. When reasoning about why they needed flip-flops, these subjects would appeal to the logarithmic relationship between the number of states and the number of flip-flops.

Across all transformations, a failure to include flip-flops or a failure to remember how they are structured was associated with a left-to-right drawing order. When subjects knew the structure of a flip-flop, they began by drawing the flip-flop first or by explicitly drawing attention to the next-state function of their logic circuit.

5 DISCUSSION

These findings suggest that the digital logic students in this study had strong procedural knowledge. The vast majority of subjects were able to accurately execute the procedures associated with transforming from one display type to another. Notably, subjects generally did not make mistakes on tasks that do not explicitly rely

on the state concept to be solved correctly (i.e., next-state table to K-map, tabular representations to Boolean expressions, and Boolean expressions to logic gates). Students' mistakes primarily reflect an inability to access tacit information about the state concept or the distinct components of a state-based system.

5.1 Tacit information in the state diagram

The use of arrows to indicate transitions between state-like ideas can be found in a variety of contexts, from decision-making to the metamorphosis of animals. Students are able to successfully build on this knowledge to extract the "origin+transition=destination" algorithm for interpreting the state diagram. However, this study revealed three examples of tacit information in the state diagram that obscured domain-specific knowledge behind conventions: a tacit 11 state, a tacit variable assignment for the next-state, and a tacit relationship between the state and system output.

The clearest example of the relevance of tacit information is how 50% of subjects failed to think about the tacit 11 state when creating next-state tables but 95% mentioned it when creating K-maps. Many subjects created rows of their next-state tables on an "as-needed" basis. Consequently, the analysis of these subjects focused on the explicit states in the state diagram. This finding is reminiscent of the idea of Proof by Incomplete Enumeration previously found by Herman et al., in which students evaluate the goodness of their Boolean logic expressions based only on readily perceptible cases [14]. In contrast, when creating a K-map, subjects created the entire grid for the K-map, which incidentally makes the 11 state encoding explicit. Subjects were then forced to grapple with the existence of this tacit state and were able to handle it appropriately. Notably, the only subject who never analyzed the tacit 11 state was also the only subject to incorrectly draw a K-map. In this case, the explicit information provided by the K-map provided subjects with the missing information they needed to identify the tacit information in the state diagram.

One interesting thing to consider is why subjects drew their next-state tables on an "as-needed" basis rather than fully enumerating all state and input combinations before beginning analysis. One reason may be the amount of writing required to fully enumerate the combinations. To enumerate all combinations in a K-map requires the subject to write only 16 1's and 0's, but the next-state table would require writing 64 1's and 0's. The overhead of fully creating the next-state table may have been a deterrent. Alternatively, subjects used K-maps to derive Boolean expressions but never successfully did so with the next-state table. A K-map's topology is its primary affordance in deriving Boolean expressions, so subjects prioritized creating its full topology. In contrast, the next-state table appeared to either be a form of mental scratch work (many subjects erased their next-state tables before creating their K-maps) or as a means to the end of creating a K-map. It is possible that a combination of these two factors influenced these different behaviors with the different tabular representations.

Another tacit piece of information in the state diagram was the encoding for the next-state variables (Q_i^+). While subjects knew how to perform the "origin+transition=destination" procedure, they did not know to search for four unique sets of variables: one set each for the state, the next-state, the system input, and the system output. Consequently, subjects were forced into situations where they either had to discover this missing information or had to incorrectly treat the next-state and output as the same thing.

Finally, the relationship between the state and system output is also tacit. In the given problem, the system output happens to have the same value as the state. The line dividing the state from the system output was insufficient to help the subjects understand the relationship between these two values, leading to subjects either ignoring the system output entirely or treating the system output as if it were the next-state. Because we only have data from this one problem, this finding is tenuous and will need to be investigated further in future studies that can compare the impact of different relationships between the values of the state and the system output.

These findings reveal a critical insight into students' conceptual understanding of state systems. Students' problem solving process is not guided by a strong conceptual framework that distinguishes and organizes four distinct constructs of state, next-state, system input, and system output. This finding aligns with prior findings by Herman et al. [15], but extends these findings by suggesting that the very representations we use to teach this topic may be in part responsible for students' failures to create this conceptual structure.

5.2 Explicit information in the state diagram and implicit information in Boolean expressions

A comparison of the "state diagram to circuit" and "Boolean to circuit" transformations reveals another set of tacit vs. explicit information that guides students' reasoning about state systems. In a state diagram, the number of states is an explicit piece of information. When transforming from a state diagram to a circuit, students readily determined the number of flip-flops in the circuit from the number of states using a logarithm. Seven subjects spontaneously performed this procedure when given the state diagram before determining their next-state logic. Additionally, four out of the five subjects who failed to include flip-flops in their first circuit added flip-flops to their second circuit when constrained to not use tabular representations. These subjects each invoked the logarithmic relationship in this context.

In contrast, the concept of state is tacit in the Boolean representation: System inputs and state variables are equivalently treated as equation inputs in Boolean expressions. Boolean expressions build on students' prior knowledge of algebra and combinational logic, which treat variables to the right of the equals sign as inputs and variables to the left of the equals sign as outputs. Because students are so adept at transforming Boolean expressions into logic gates, it is likely that subjects so automated this process that they simply executed the process, failing to extract the tacit information that state variables are not system inputs. Notably, when subjects made these mistakes, they were forced again into situations in which they either needed to discover the difference between system inputs and state or they needed to treat the next-state variables as if they were either the system output or the same as the state. Without flip-flops separating state and next-state, subjects had to incorrectly map distinct concepts onto the same wire.

This finding underscores that students' problem solving is not guided by a proper conceptual distinction of state, next-state, system inputs, and system outputs. Critically, subjects do not search the problem for this information, despite its central importance.

5.3 Reading order analysis and creation of displays

Many subjects analyzed the state transitions in natural reading order (top-left to bottom-right). This finding was particularly salient when students were creating the next-state tables on an "as needed" basis. Rather than create rows in the table in numeric order (i.e., 0000, 0001, 0010, 0011), subjects created rows in reading order (see Figure 4, notice the 0000, 0011, 0010, 0001 order). This ordering reinforces the observation that subjects may be treating next-state tables as mental scratchwork rather than systematically approaching the tool. Future research will need to explore more whether this type of reading order analysis is actually a detriment on students' performance.

Similarly, when subjects revealed a weak conceptual organization of the four distinct components of a state system, they also used a left-to-right drawing order associated with stateless, input-output systems. Students revealed this weak understanding by failing to add flip-flops to their circuits, drawing flip-flops incorrectly, or being unable to correctly name flip-flops. In contrast, when subjects revealed a strong connection of flip-flops to state and distinct from the other system components, they began their drawings by adding flip-flops first and then drawing next-state logic. Because the tabular representations and equations lack strong perceptual cues to help subjects distinguish between state and system inputs, students defaulted to using natural and mathematical reading schema to interpret their tables and equations.

This finding suggests again that students do not maintain strong conceptual distinctions between the four components of a state-based system and instead treat state systems as stateless, input-output systems. This finding also suggests that tracking the order in which students draw diagrams may provide insights into students' underlying conceptual understanding. Future studies will need to further investigate the robustness of this finding, but it could have implications for future research on students' cognitive processes as well as potentially new modes of assessment that can use the visible processes of students' drawings to diagnose hidden cognitive processes.

6 CONCLUSION AND IMPLICATIONS FOR INSTRUCTION

The core finding of this study is that many digital logic students do not develop a conceptual understanding of state-based systems that maintains a strong distinction between the four components of a state system: state, next-state, system input, and system output. Consequently, when subjects encountered representations that tacitly encoded elements of these distinctions, yet relied on these distinctions for correct interpretation, subjects made mistakes. This study suggests that the primary challenge for students learning digital logic and computer architecture may be that they simply have not created the conceptual distinctions necessary to navigate each new representation of a state machine.

One concern is that this lack of conceptual distinction may be a result of the current instructional model of digital logic. Current instructional models typically spend several lectures or weeks ingraining how to create logic gates from Boolean variables (e.g., [3, 27, 28]). It appears that this part of instruction is successful as students were adept at these skills. However, it appears that students' adeptness with these skills drowns out the need to extract the four distinct components from their Boolean expressions. The accurate and effortless creation of logic gates from Boolean expressions helps students remain oblivious to their mistakes. Since students can master the mechanistic parts of these transformations, perhaps we should begin with state-based systems and teach these mechanistic transformations in the context of a state-based system.

This suggestion certainly defies tradition, but it aligns with shifts in instructional models in other disciplines. For example, biology courses have shifted away from teaching taxonomies of species toward computationally modeling how species interact or develop [2]. The core idea behind such shifts is that a better understanding of basic skills or knowledge results from understanding how those skills and knowledge fit in the context of core concepts and models. In teaching students digital logic, our goal should be teach students how a computer functions. Much like how we can expect that any student who has taken an integrative biology course will have an understanding of predator-prey models, we should expect that our students should have an understanding of how to model state-based systems. If students cannot even consistently, correctly distinguish the components of this model after passing a digital logic course, is the current method of teaching FSMs only after students have mastered combinational logic really working?

While changing the fundamental organization by which we teach digital logic may be overwhelming, these results also suggest other, potentially simpler avenues for improving learning. Instruction could focus on helping students know how to interpret diagrams better or creating more informative diagrams that may help students perceive the different components of state. For example, adding the naming conventions of next-state variables to the legend of state diagrams may help students remember to consider these variables. The goal of this intervention is to aligning perceptually salient distinctions in our diagrams with conceptual important distinctions. Alternatively, using representations that constrain students to use expert-like problem solving schema (e.g., using K-maps which encourage full enumeration of input combinations) may accelerate students' learning and improving their performance [9].

Beyond the implications of these findings for digital logic instruction, these findings also suggest avenues for future research. What tacit information is hiding in the representations that we use in other computing topics such as programming? Are our canonical examples hiding essential information from novices, hindering their learning?

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under grant EEC 1429348. The opinions, findings, and conclusions do not necessarily reflect the views of the National Science Foundation or the author's institution.

REFERENCES

[1] H. Boeije. 2002. A purposeful approach to the constant comparative method in the analysis of qualitative interviews. *Quality & Quantity* 26 (2002), 391–409.

[2] C. A. Brewer and D. Smith (Eds.). 2009. *Vision and Change in Undergraduate Biology Education: A Call to Action.* American Association for the Advancement of Science, Washington, DC.

[3] S. Brown and Z. Vranesic. 2009. *Fundamentals of Digital Logic with VHDL Design.* McGraw-Hill, New York.

[4] M. Canham and M. Hegarty. 2010. Effects of knowledge and display design on comprehension of complex graphics. *Learning and Instruction* 20, 2 (2010), 155–166.

[5] W. G. Chase and H. A. Simon. 1973. Perception in chess. *Cognitive Psychology* 4 (1973), 55–81.

[6] J. Corbin and A. Strauss. 2007. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory.* Sage, Thousand Oaks, CA.

[7] B. duBoulay. 1989. *Some difficulties learning to program.* Lawrence Erlbaum, Hillside, New Jersey.

[8] B. duBoulay, T. O'Shea, and J. Monk. 1989. *The black box inside the glass box: Presenting computing concepts to novices.* Lawrence Erlbaum, Hillside, New Jersey, 431–446.

[9] Robert J. Dufresne, William J. Gerace, Pamela T. Hardiman, and Jose P Mestre. 1992. Constraining novices to perform expert like problem analyses: Effects on schema acquisition. *Journal of the Learning Sciences* 2 (1992), 307–331.

[10] A. F. Heckler and T. M. Scaife. 2015. Adding and subtracting vectors: The problem with the arrow representation. *Physical Review Special Topics-Physics Education Research* 11 (2015), 010101.

[11] M. Hegarty. 2011. The cognitive science of visual-spatial displays: Implications for design. *Topics in Cognitive Science* 3 (2011), 446–474.

[12] M. Hegarty, M. Stieff, and B. L. Dixon. 2013. Cognitive change in mental models with experience in the domain of organic chemistry. *Journal of Cognitive Psychology* 25, 2 (2013), 220–228.

[13] J. Heiser and B. Tversky. 2006. Arrows in comprehending and producing mechanical diagrams. *Cognitive Science* 30 (2006), 587–592.

[14] G. L. Herman, M. C. Loui, L. Kaczmarczyk, and C. Zilles. 2012. Discovering the what and why of students' difficulties in Boolean logic. *ACM Transactions on Computing Education* 12, 1 (2012), 3:1–28.

[15] G. L. Herman, C. Zilles, and M. C. Loui. 2012. Flip-flops in students' conceptions of state. *IEEE Transactions on Education* 55, 1 (2012), 88–98.

[16] Matthew Hertz and Maria Jump. 2013. Trace-based Teaching in Early Programming Courses. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13).* ACM, New York, NY, USA, 561–566. DOI: https://doi.org/10.1145/2445196.2445364

[17] Colleen M. Lewis. 2012. *Applications of Out-of-Domain Knowledge in Students' Reasoning about Computer Program State.* Ph.D. Dissertation.

[18] Colleen M. Lewis. 2012. The Importance of Students' Attention to Program State: A Case Study of Debugging Behavior. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research (ICER '12).* ACM, New York, NY, USA, 127–134. DOI: https://doi.org/10.1145/2361276.2361301

[19] L. J. Mselle. 2010. Enhancing comprehension by using Random Access Memory (RAM) diagrams in teaching programming: Class experiment. In *Proceedings of the Psychology of Programming Interest Group (PPIG 2010).*

[20] D. Shinners-Kennedy. 2008. *The everydayness of threshold concepts: State as an example from computer science.* Sense, Rotterdam, The Netherlands, 119–128.

[21] Juha Sorva. 2007. Students' Understandings of Storing Objects. In *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research - Volume 88 (Koli Calling '07).* Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 127–135. http://dl.acm.org/citation.cfm?id=2449323.2449337

[22] Juha Sorva. 2013. Notional Machines and Introductory Programming Education. *Trans. Comput. Educ.* 13, 2, Article 8 (July 2013), 31 pages. DOI: https://doi.org/10.1145/2483710.2483713

[23] Juha Sorva, Ville Karavirta, and Lauri Malmi. 2013. A Review of Generic Program Visualization Systems for Introductory Programming Education. *Trans. Comput. Educ.* 13, 4, Article 15 (Nov. 2013), 64 pages. DOI: https://doi.org/10.1145/2490822

[24] P. S. Steif, J. Lobue, A. L. Fay, and L. B. Kara. 2010. Improving problem solving performance by inducing talk about salient problem features. *Journal of Engineering Education* 99 (2010), 135–142.

[25] M. Stieff, M. Hegarty, and G. Deslongchamps. 2011. Identifying representational competence with multi-representational displays. *Cognition and Instruction* 29 (2011), 123–145.

[26] D. H. Uttal and C. A. Cohen. 2012. *Spatial thinking and STEM education: When, Why, and How?* Vol. 57. 147–181.

[27] F. Vahid. 2006. *Digital Design.* Wiley, Hoboken, NJ.

[28] J. F. Wakerly. 2006. *Digital Design: Principles and Practices.* Prentice-Hall, Upper Saddle River, NJ.