jsFLAP

A JavaScript Formal Languages and Automata Package for Computer Science Education

Elijah Cirioli

Background

Visualization in computer science education

Visualization in education

- Interactive visualization and simulation serve an important role in the education of abstract topics
- They complement more traditional learning materials rather than replacing them
- They allow students to engage with concepts very directly, applying what was once highly theoretical

Visualization in education

- In computer science education, visualizations can be especially important

- A model for how this is done is called Visualization-Reinforced Instruction (VRI)*
  - “A form of active learning, in which traditional instructional material is complemented and enhanced with carefully designed animations and simulations of key concepts, models, and algorithms”

*M. Quweider and F. Khan, “Visualization as effective instructional and learning tools in the Computer Science Curriculum,” *2017 ASEE Annual Conference & Exposition Proceedings*, 2017.

Benefits of visualization

- Researchers studied the use of VRI modules for educating about topics including algorithms, the fundamentals of programming, and network security

- VRI were found to have many benefits
  - 76% of students responded that the visualizations helped them to understand the underlying concepts
  - Allowed educators to better understand student misconceptions and misunderstandings which they could then correct
  - Less time overall (lecture + visualization) was required for students to learn the concepts

Challenges for visualizations

- Effective visualizations that enhance education are not easy to create
- Researchers discuss both the benefits and challenges of implementing visualizations in computer science education*
- Two major obstacles to widespread adoption of such visualization tools were identified
  - The learner may not find them to be educationally beneficial
  - The educator may find them to incur too much structural overhead to be worthwhile to implement in their course

*T. L. Naps, G. Rößling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, and J. Á. Velázquez-Iturbide, “Exploring the role of visualization and engagement in Computer Science Education,” ACM SIGCSE Bulletin, vol. 35, no. 2, pp. 131–152, 2003.

Designing good visualizations

- Effective visualization requires more than just showing a picture or animation of an algorithm, interactivity is critical

- The researchers define a set of best practices for designing visualizations
  - Include execution history
  - Support flexible execution control
  - Support custom input sets
  - Support learner-built visualizations
  - And more…

- These practices help to create a proper education tool with which the user can actively engage with the relevant concepts

Benefits of good visualizations

- When done well, these researchers also espouse the benefits of visualizations for computer science education
  - All respondents to one survey agreed that “using visualizations can help students learn computing concepts”
  - Improved levels of student participation
  - Educators reported a more enjoyable teaching experience
  - Students reported a more fun learning experience

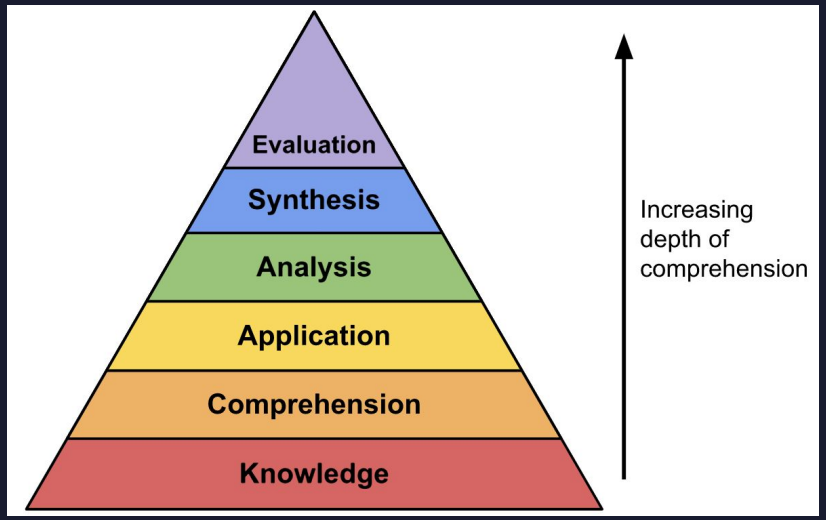
- How does one quantify the benefit that a visualization provides?

Educational theory

- Bloom's taxonomy of the cognitive domain is a hierarchical framework that is used to classify a learner's depth of comprehension along six increasingly sophisticated level*

- This has been used in some form for decades to define educational objectives that educators strive for their students to reach

[1]



*B. S. Bloom, Taxonomy of educational objectives: The classification of educational goals. New York City, New York: Longman, 1984.

Educational theory

- A visualization tool can be judged by which level of Bloom's taxonomy it allows students to reach

- Any visualization tool following the best practices defined earlier should bring students to the Application level
  - At this level, students can take concepts they have learned and apply them to new problems

- An even better tool would bring students to the Synthesis level
  - At this level students can generalize from many facts that they have learned and draw new conclusions by combining multiple concepts

Applications in computing theory

- The theory of computation is a prime candidate for visualization-reinforced learning

- The mathematical definitions for formal languages and finite automata can be difficult for students to understand on their own
    - They already have agreed-upon visual representations in literature, it just isn't interactive

- The concepts of computational theory are foundational to much of computer science, so understanding it is critical

- Because the need is so great, some tools already exist (more on this later)

Background

Formal languages and finite state automata

Formal languages

- A formal language consists of a set of valid words constructed from a set of characters in the language's alphabet according to certain rules

- Many applications in math, logic, linguistics, and computer science including programming language design and the theory of computation

- There are a number of tools that can be used to define formal languages
  - Set notation
  - Grammars
  - Regular expressions
  - Finite state automata

Finite automata

- A finite-state automaton is a form of computer that defines a language by accepting or rejecting different words that are passed to it as inputs

- There are different forms of finite automata with different levels of computational power to accept different types of languages, including
  - Finite automata to accept regular languages
  - Pushdown automata to accept context-free languages
  - Turing machines to accept recursively enumerable languages

- They are all modeled as finite state machines with a finite set of states and transitions to move between those states based on certain criteria
  - Depending on how an automaton has been defined, it may have non-determinism which can alter its computing power in some cases

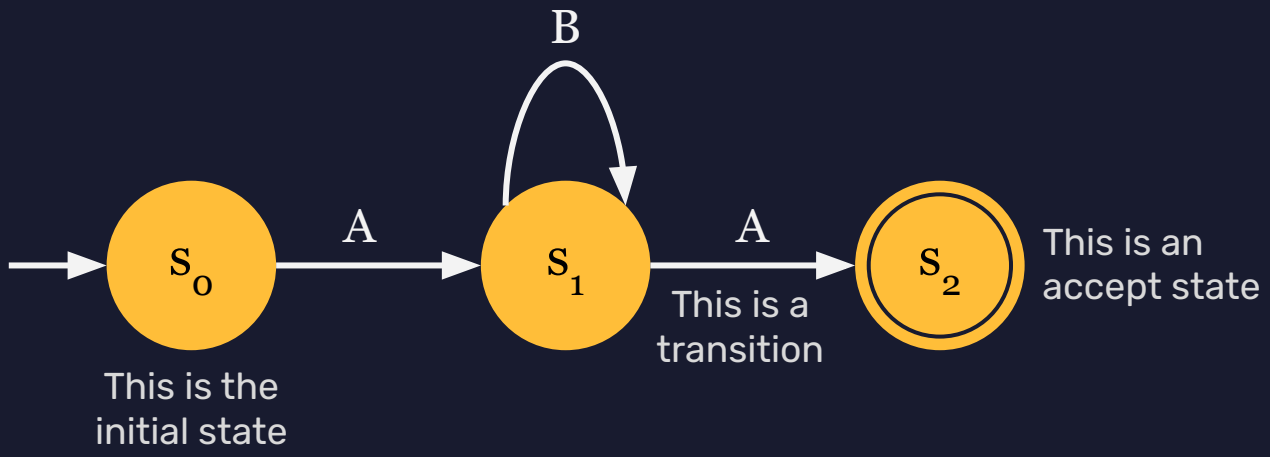
Simple finite automata

- Finite automata for accepting regular languages (NFAs and DFAs) are defined as a 5-tuple $(Q, \Sigma, \sigma, q_0, F)$ where
  - Q is a finite set of states
  - Σ is a finite set of tokens called the alphabet
  - σ is the transition function mapping $Q \times \Sigma \rightarrow Q$
  - q_0 is the initial state
  - $F \subseteq Q$ is the set of accept states

- The language constructed of all words that are the character A followed by any amount of Bs followed by another A could be defined by the automaton $(\{s_0, s_1, s_2\}, \{A, B\}, f, s_0, \{s_2\})$
  Where f is the transition function mapping $(s_0, A) \rightarrow s_1$, $(s_1, B) \rightarrow s_1$, $(s_1, A) \rightarrow s_2$
  Since this transition function does not contain transitions for all combinations of states and alphabet characters, the automaton is non-deterministic

Simple finite automata

- This finite automaton $(\{s_0, s_1, s_2\}, \{A, B\}, f, s_0, \{s_2\})$ where f is the transition function mapping $(s_0, A) \rightarrow s_1$, $(s_1, B) \rightarrow s_1$, $(s_1, A) \rightarrow s_2$ can be represented visually as a directed graph



Background

JFLAP

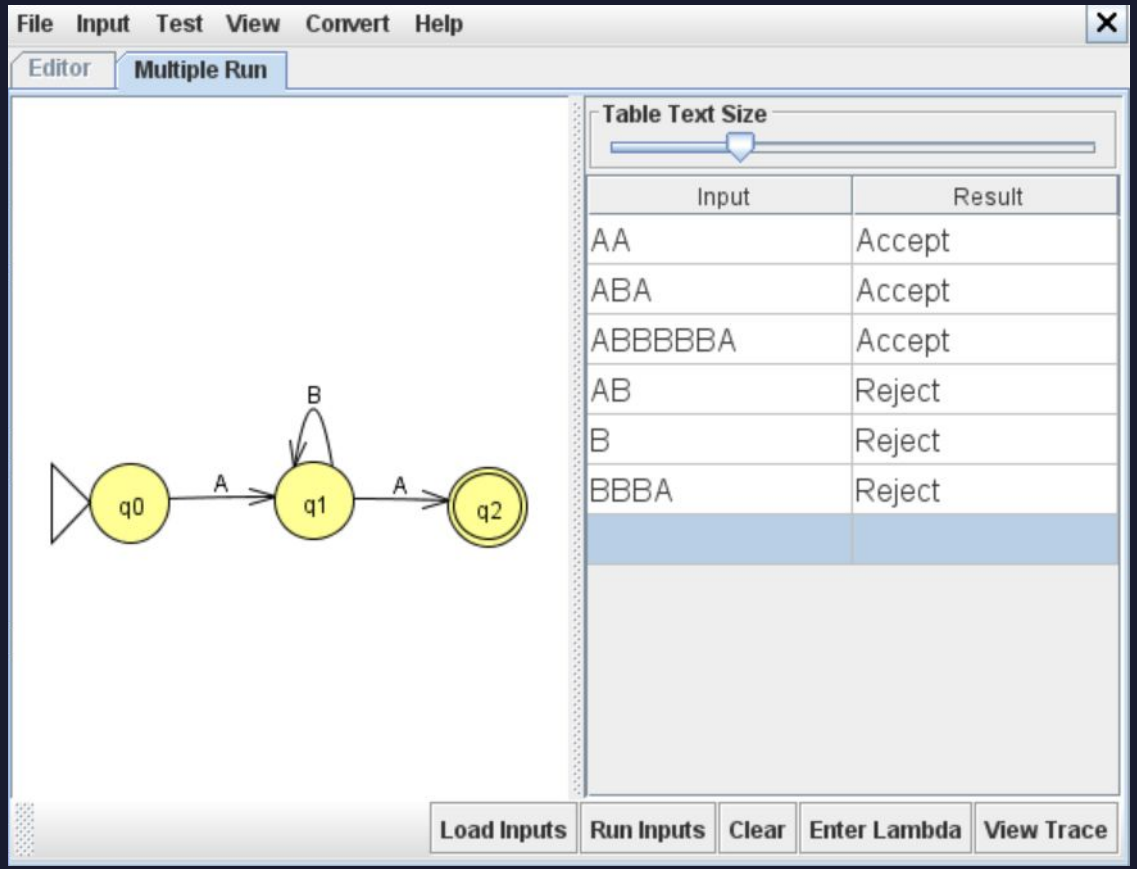
JFLAP

- The Java Formal Languages and Automata Package (JFLAP) is a Java program whose development began in 1990 at Rensselaer Polytechnic Institute

- Allows for the construction and simulation of different types of finite state automata, as well as regular expressions and grammars

- Downloaded over 64,000 times by people in 161 countries, and is used in over 20,000 college courses (including at OSU!)

JFLAP


File
Input
Test
View
Convert
Help
Editor
Multiple Run
Table Text Size
Input
Result
AA
Accept
ABA
Accept
ABBBBBA
Accept
AB
Reject
B
Reject
BBBA
Reject
B
A
A
q0
q1
q2
Load Inputs
Run Inputs
Clear
Enter Lambda
View Trace

The negatives of JFLAP

- Technological overhead
  - Students need to install a third-party program and JVM
  - Can’t use Chromebooks or mobile devices
- Functionality
  - Lacks some educationally useful algorithms like product automata and cycle detection
- Usability
  - User interface reflects the time it was made
  - Generally only one way to achieve anything
  - Missing quality of life features like keyboard shortcuts, editor tabs, and good layout algorithms

The goal

- Create a modern, accessible tool for constructing and simulating finite state automata
  - Support non-deterministic finite automata, pushdown automata, and Turing machines
  - Implement useful algorithms to demonstrate core concepts of computing theory

- Follow best practices for visualization programs
  - High degree of user control for construction and execution
  - Easy for instructors to integrate
  - Avoid student frustration

- Allow users to reach at least the Application stage of Bloom's taxonomy and hopefully the Synthesis stage

Implementation

Technologies

- jsFLAP: The JavaScript Formal Languages and Automata Package
- Implemented using HTML5, CSS3, and JavaScript as well as the jQuery library



*Wikimedia Commons

Design strategy

- Composed of compartmentalized, object-oriented modules
- Code is documented and freely available on GitHub under the MIT license
- High extensibility through modifying modules or adding new ones
  - Should allow customization to suit course-specific needs or to connect to existing systems

Using jsFLAP

- All code is run in the client's browser, no external server connection is required

- Can be accessed at elijahcirioli.com/jsflap, embedded within another site, hosted statically by a school, or packaged into an executable so that students will not require an internet connection

- Only thing required is a modern-enough browser and possibly an internet connection depending on how it is hosted

Development Timeline

- Work began in October 2021 and lasted approximately 11 months
- All originally-intended functionality is present
  - Three types of automata plus, two types of parsing, and many tools
- Additional development may occur in the future to fix bugs and add features
- The goal was to create software that is easy to come back to and easy for others to modify

Features

Automata construction

- The core functionality revolves around constructing finite state automata by clicking and dragging to create states and transitions
  - There are various tools and keyboard shortcuts that can help with this

- When starting the program, the user gets a choice for what kind of automaton they wish to create
  - Finite state automaton
  - Pushdown automaton
  - Turing machine

Simulation

- There are two modes of simulation
  - Multiple
  - Step-by-step

- Simulation outcomes depend on what type of automaton is being simulated
  - Standard and pushdown automata return whether a given word is in their language
  - Turing machines show tape transformations and whether the machine halted (within a set amount of time)

- All simulations update in real-time

Messages

- The importance of features that decrease user frustration should not be discounted

- The editor features a messages window that updates in real time with different information about the automaton being constructed
  - What type of automaton (including non-determinism)
  - The alphabet
  - Whether it contains cycles

- Warnings for common mistakes
  - Unreachable states
  - Lack of initial state
  - Lack of any final states

Usability

- There are a number of features focused on improving the user experience
  - Copy/paste
  - Undo/redo
  - Auto saving
  - Editor themes
  - Keyboard shortcuts
  - Multiple selection
  - Editor panning and zooming (including auto zoom)
  - Optionally snap to grid
  - Multiple ways to access all tools
  - Multiple layout algorithms to make automata look nicer

Equivalence

- NFA to DFA conversion tool
  - Convert any non-deterministic finite automaton to a deterministic finite automaton
  - Demonstrates their equal computing power

- Compare equivalence of two automata
  - Show that multiple automata can represent the same language
  - Test correctness of student solutions
  - Again show NFA and DFA correspondence

Regular expressions

- Input regular expressions and generate equivalent NFAs
- Demonstration of equivalence between representations for regular languages
- Combining concepts to promote synthesis learning

Product automata

- Constructs the cartesian product automaton from two finite automata

- Perform set operations on their final states to perform the same operations on the languages they accept
  - Union, intersection, difference
  - Difference can be used prove equivalence

- Deepen student understanding of connection between finite automata and the languages they represent, as well as providing more tools to solve problems

Future Improvements

Accessibility

Accessibility

- Great work was put in to make jsFLAP accessible, but it should remain a key area of focus for future development
  - Tooltips, color settings, user-driven timing

- Reliance on mouse

- Improved support for screen readers

Future Improvements

Functionality

Grammars

- Grammars are another common way of representing formal languages
- They are not graph-based at all, so they would require a whole new suite of interfaces
- Supported in JFLAP
- Functionality to convert between automata and grammars in both directions

Proofs

- Currently, jsFLAP has functionality for students to prove concepts to themselves, but no formalized proofs that walk them through that concept
- Walk students through using the pumping lemma to show that a language is not regular
- Demonstrate the halting problem for Turing machines
- Broadly moves jsFLAP from an educational tool to a learning platform

Educational platform

- Currently jsFLAP is a tool used within an educational context rather than a place that students can learn computing theory concepts without needing any other resources

- Would need to provide users with lessons and challenges to teach different concepts
  - Could either be preprogrammed or specified in some file format

- Requires system for formally evaluating student work

- This would be a large undertaking, but it could be very powerful if done well

Educational Usage

Use cases

- In an educational context jsFLAP has many use cases
  - Live demonstrations
  - In-class activities
  - Homework assignments
  - Create graphics for slideshows and textbooks
- Cover both basic and advanced topics
- Adding visualization to courses that currently lack it
- Replacing JFLAP
  - Interoperable file formats
  - Feature differences

Specific courses

- jsFLAP is already being used by professor Xanda Schofield for the finite automata unit of their introductory computer science class at Harvey Mudd College
  - They have connected jsFLAP to their autograding system, demonstrating its extensibility

- Oregon State University could follow Harvey Mudd's lead by replacing JFLAP with jsFLAP in the CS 321 Theory of Computation undergraduate course

Conclusion

Conclusion

- Effective visualization tools can greatly benefit students, especially when learning about abstract topics

- Theory of computation is foundational for much of computer science, so jsFLAP was created as a tool to construct and simulate different types of finite state automata

- Provides an easy way for students to engage with the material in a hands-on manner
  - Work through misconceptions
  - Apply practically what was once theoretical

Conclusion

- While not the first tool for this purpose, jsFLAP follows best practices in order to be the most effective learning tool
  - Easy for instructors to integrate into their curriculum
  - Easy for students to use and access
  - Engaging, student-driven visualizations
  - Synthesis of multiple concepts

- Extensible and customizable to suit different learning environments

- jsFLAP has potential to benefit students at all levels of computer science education