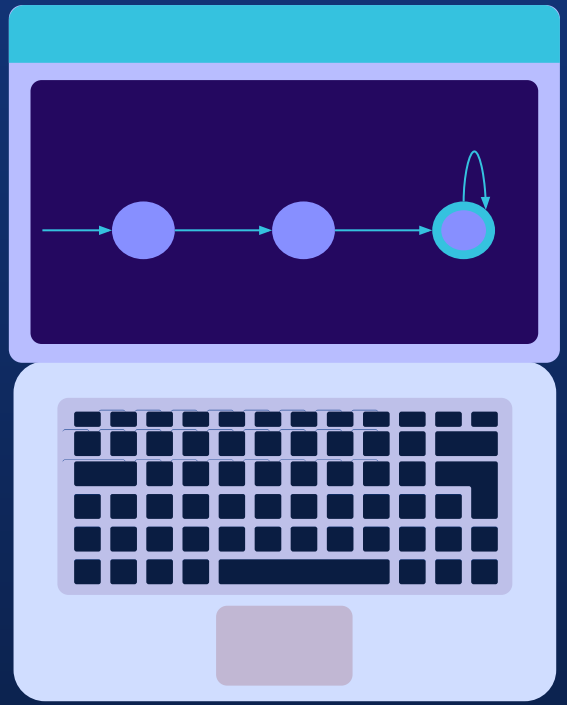
Improved Visualization for Formal Language: **Milestone 3**

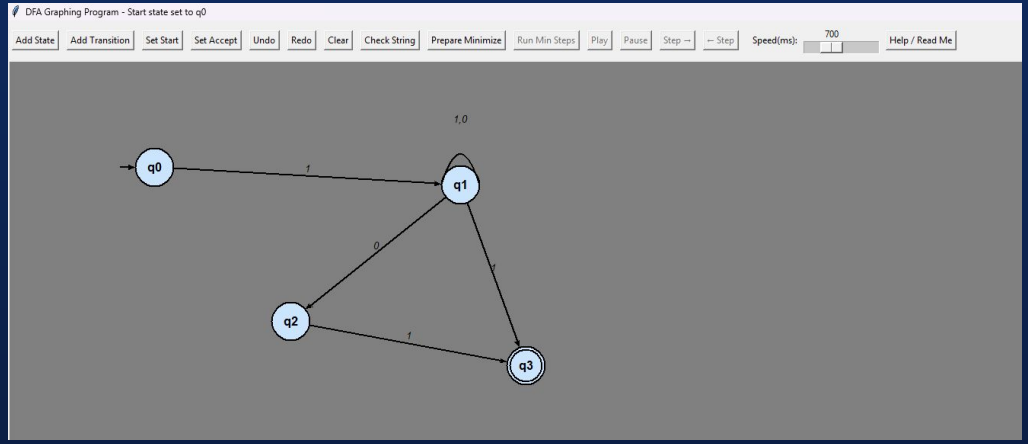


https://kmcnear2022.github.io/

Group Members: Chris Pinto-Font, Vincent Borrelli, Andrew Bastien, Keegan McNear

Milestone 3 Goals

We set out improve our visual graphing environment by adding new requested features and improving visual cohesion and design.



Milestone Three Deliverables

Animated Traversal System

Created a system for the user to check a string by traversing the DFA visually and seeing how Its compiled.

Onboard “Read Me” file

Expanded “Read Me” file for the the new features introduced at this milestone.

Have a rough version of the DFA minimization process

Worked on getting the logic side and visual implementation of a DFA minimization process, still can improve user experience with it and minimization accuracy.

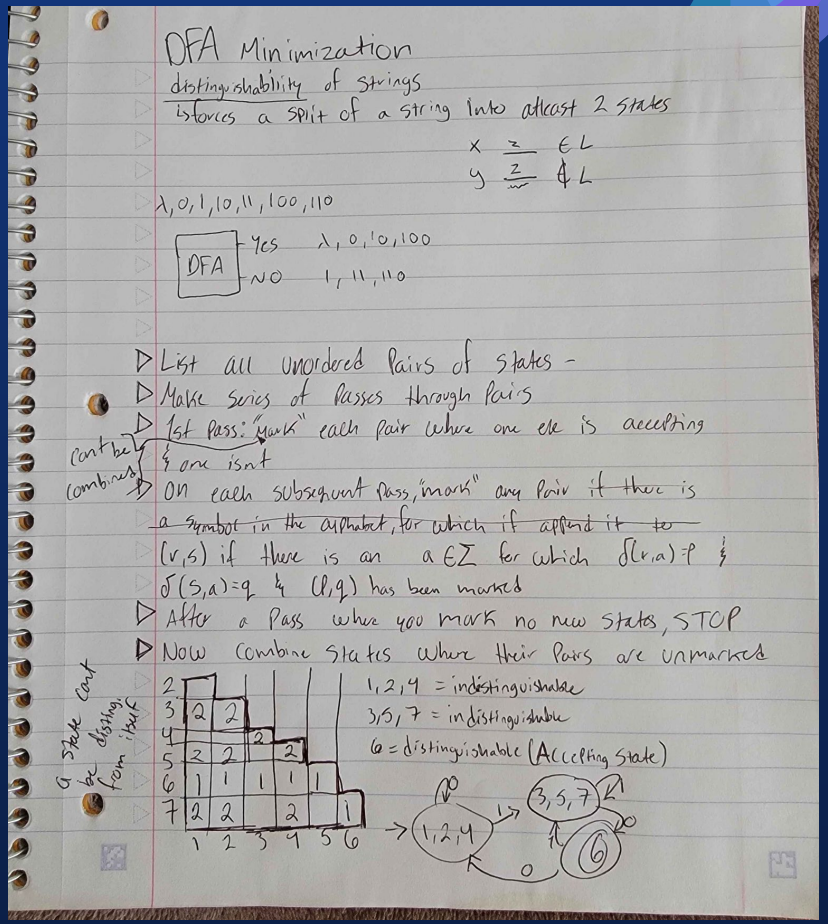
Minimization Research

Met with Dr L to learn more about minimization as a process so that we could implement it into our program accurately

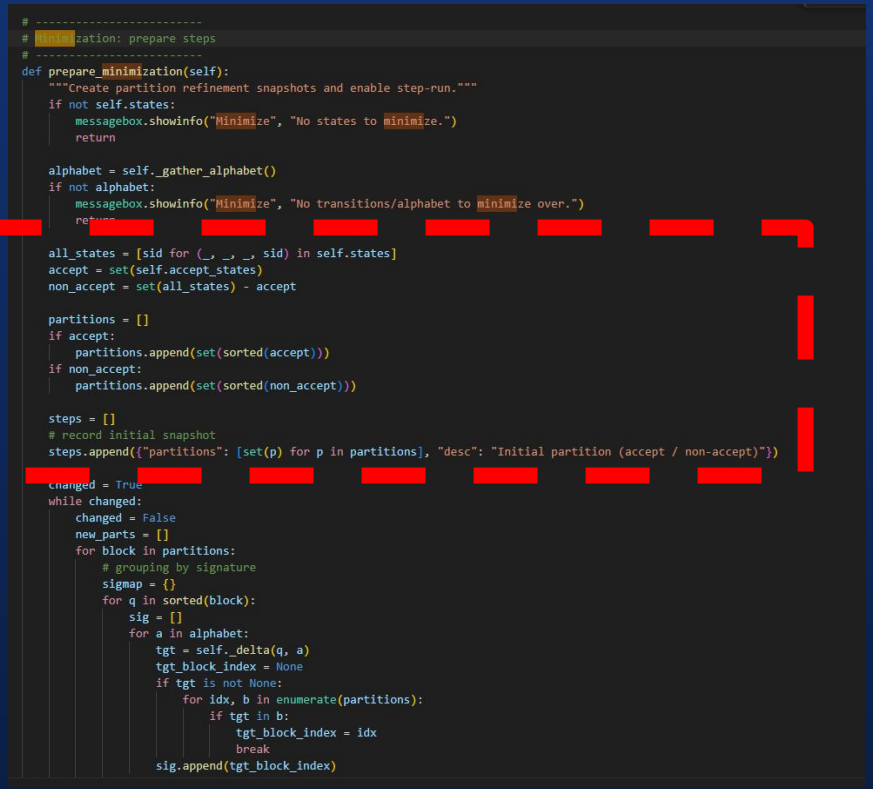
Minimization Research

Group Member Chris Pinto-Font

met with our advisor/client Dr L to discuss how minimization works in a practical sense, allowing us a better understanding of it for our implementation.



Code Excerpts: Minimization Preparation



```python
# ------------------------
# Minimization: prepare steps
# ------------------------
def prepare_minimization(self):
    """Create partition refinement snapshots and enable step-run."""
    if not self.states:
        messagebox.showinfo("Minimize", "No states to minimize.")
        return

    alphabet = self._gather_alphabet()
    if not alphabet:
        messagebox.showinfo("Minimize", "No transitions/alphabet to minimize over.")
        return

    all_states = [sid for (_, _, _, sid) in self.states]
    accept = set(self.accept_states)
    non_accept = set(all_states) - accept

    partitions = []
    if accept:
        partitions.append(set(sorted(accept)))
    if non_accept:
        partitions.append(set(sorted(non_accept)))

    steps = []
    # record initial snapshot
    steps.append({"partitions": [set(p) for p in partitions], "desc": "Initial partition (accept / non-accept)"})

    changed = True
    while changed:
        changed = False
        new_parts = []
        for block in partitions:
            # grouping by signature
            sigmap = {}
            for q in sorted(block):
                sig = []
                for a in alphabet:
                    tgt = self._delta(q, a)
                    tgt_block_index = None
                    if tgt is not None:
                        for idx, b in enumerate(partitions):
                            if tgt in b:
                                tgt_block_index = idx
                                break
                    sig.append(tgt_block_index)
```

Prepares for Minimization Process.

Encompasses the very first partition used in the minimization algorithm, dividing states into accepting states and non-accepting states.

Code Excerpts: Minimization

```
changed = True
while changed:
    changed = False
    new_parts = []
    for block in partitions:
        # grouping by signature
        sigmap = {}
        for q in sorted(block):
            sig = []
            for a in alphabet:
                tgt = self._delta(q, a)
                tgt_block_index = None
                if tgt is not None:
                    for idx, b in enumerate(partitions):
                        if tgt in b:
                            tgt_block_index = idx
                            break
                sig.append(tgt_block_index)
            sig = tuple(sig)
            sigmap.setdefault(sig, set()).add(q)
        if len(sigmap) == 1:
            new_parts.append(set(block))
        else:
            # record split
            for sblock in sigmap.values():
                new_parts.append(set(sblock))
            changed = True
            steps.append({"partitions": [set(p) for p in partitions],
                          "desc": f"Split block {sorted(list(block))} into " + "; ".join(str(sorted(list(s))) for s in sigmap.values())})
    partitions = new_parts

steps.append({"partitions": [set(p) for p in partitions], "desc": "Final partition (no further splits)"})
self.min_steps = steps
self.min_step_index = -1
self.set_status("Minimization prepared. Open Min Steps to view.")
# enable run button
self.run_min_btn.config(state="normal")
# open steps window automatically
self.open_minimization_window()
```

Core of the partition refinement.
For each state in the block, build a signature:
- For each symbol, find which partition the transition leads to
- Example signature: (1, 1, 0) meaning on a → block 1, on b → block 1, on c → block 0

States with different signatures cannot be equivalent → the block must be split.

When a split occurs:
- New blocks replace the old block
- A minimization step snapshot is recorded and shown in the GUI

This implements the formal refinement rule:
Two states are equivalent only if all transitions on all symbols go to the same partitions.

Code Excerpts: Minimization Animation

```
    steps = []
    # record initial snapshot
    steps.append({"partitions": [set(p) for p in partitions], "desc": "Initial partition (accept / non-accept)"})

    changed = True
    while changed:
        changed = False
        new_parts = []
        for block in partitions:
            # grouping by signature
            sigmap = {}
            for q in sorted(block):
                sig = []
                for a in alphabet:
                    tgt = self._delta(q, a)
                    tgt_block_index = None
                    if tgt is not None:
                        for idx, b in enumerate(partitions):
                            if tgt in b:
                                tgt_block_index = idx
                                break
                    sig.append(tgt_block_index)
                sig = tuple(sig)
                sigmap.setdefault(sig, set()).add(q)
            if len(sigmap) == 1:
                new_parts.append(set(block))
            else:
                # record split
                for sblock in sigmap.values():
                    new_parts.append(set(sblock))
                changed = True
                steps.append({"partitions": [set(p) for p in partitions],
                              "desc": f"Split block {sorted(list(block))} into " + "; ".join(str(sorted(list(s))) for s in sigmap.values())})
        partitions = new_parts

    steps.append({"partitions": [set(p) for p in partitions], "desc": "Final partition (no further splits)"})

    self.min_steps = steps
    self.min_step_index = -1
    self.set_status("Minimization prepared. Open Min Steps to view.")
    # enable run button
    self.run_min_btn.config(state="normal")
    # open steps window automatically
    self.open_minimization_window()
```

The program stores each refinement step in a list:

- Every iteration partition configuration
- A human-readable description
- Used later for animation and step navigation

The GUI then displays all steps automatically.

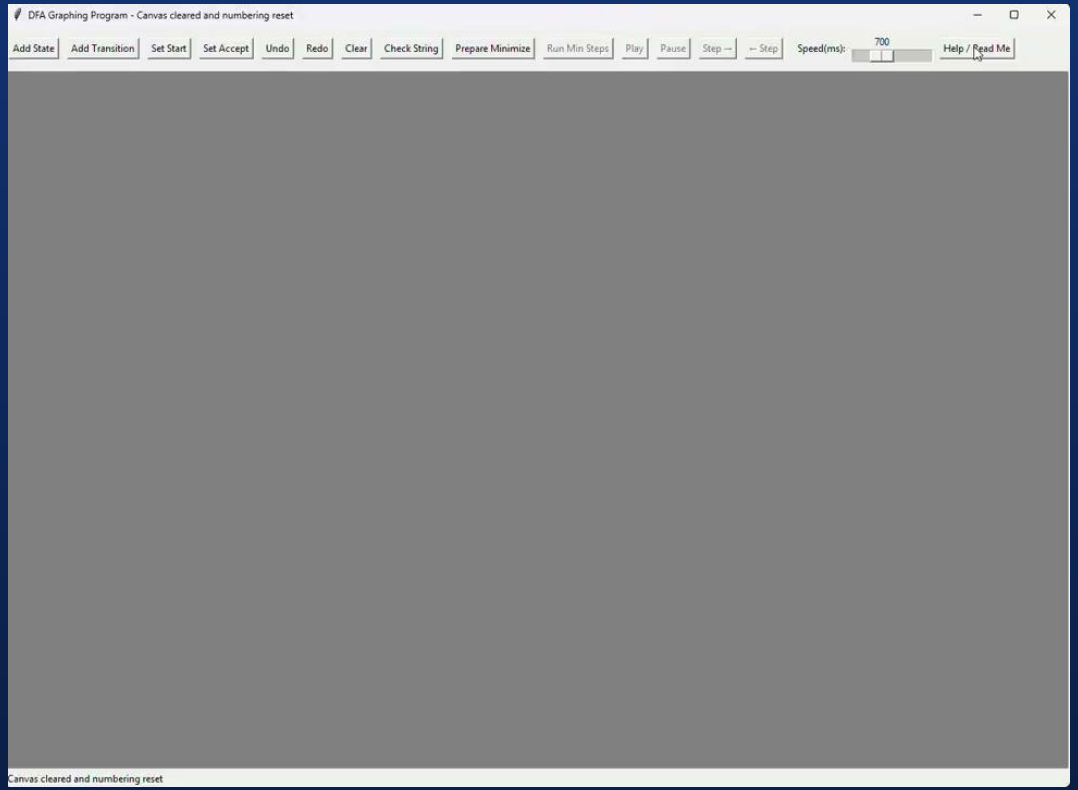
Code Excerpts: Building new DFA

```python
# ------------------------
# Build and apply minimized DFA
# ------------------------
def apply_minimized_dfa(self):
    if not self.min_steps:
        messagebox.showinfo("Minimize", "No prepared minimization steps.")
        return
    final = self.min_steps[-1]["partitions"]
    # map old state -> block index
    state_to_block = {}
    for idx, block in enumerate(final):
        for s in block:
            state_to_block[s] = idx

    # new positions: centroid of members
    new_pos = {}
    for idx, block in enumerate(final):
        xs = []
        ys = []
        for s in block:
            c = self.get_state_center(s)
            if c:
                xs.append(c[0])
                ys.append(
        if xs:
            new_pos[idx] =
        else:
            new_pos[idx] =

    # new alphabet and tra
```

```python
    # new alphabet and transition map
    alphabet = self._gather_alphabet()
    new_map = {}
    for old_state in list(state_to_block.keys()):
        frm_blk = state_to_block[old_state]
        for a in alphabet:
            tgt = self._delta(old_state, a)
            if tgt is not None:
                to_blk = state_to_block.get(tgt)
                if to_blk is not None:
                    # deterministic: last write wins (should be consistent)
                    new_map.setdefault((frm_blk, a), to_blk)

    # combine symbols into edges (frm_blk,to_blk) -> set(symbols)
    combined = {}
    for (frm, sym), to in new_map.items():
        combined.setdefault((frm, to), set()).add(sym)
```

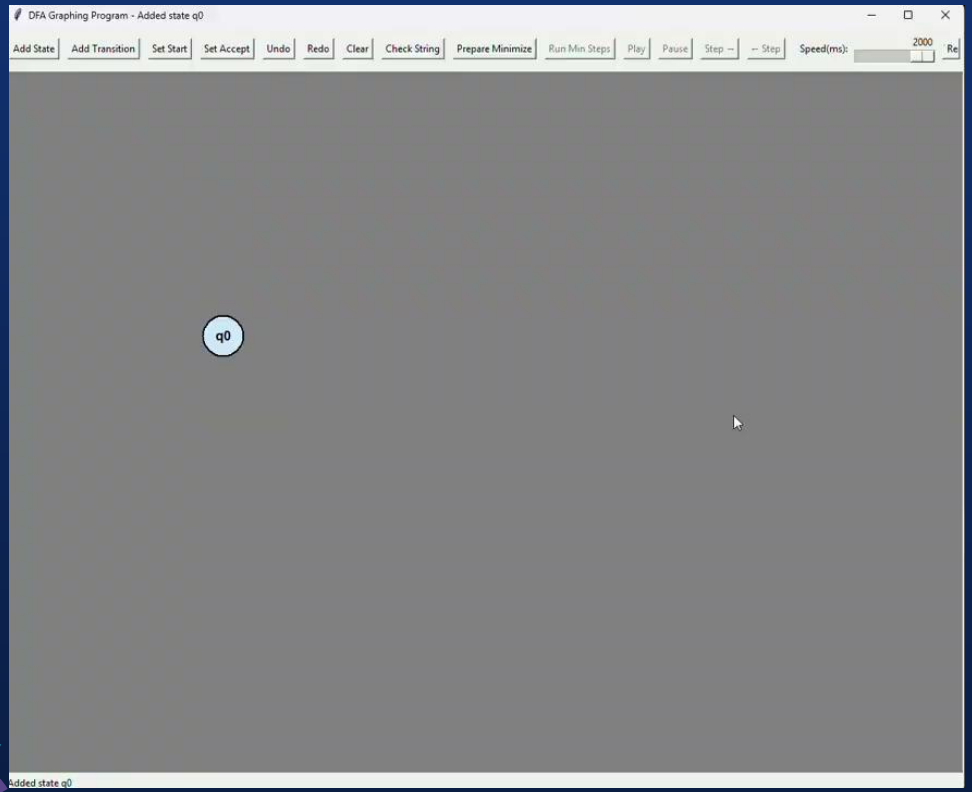
Maps the original states to its minimized DFA state

Video Demonstration - Animated Traversal



Several State DFA traversed via string checking, lets user do it instantly or step through it, both with controllable animation speeds.

Video Demonstration - Rough Minimization

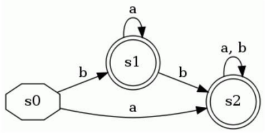


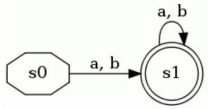
Uses Researched Implementation of minimization process to minimize DFA

(will be expanded and refined with more animations and tested use cases)

DFA Minimization

- So these two DFAs are *equivalent*:





Milestone 3 Progress

Task	Completion %	Chris	Vincent	Andrew	Keegan	To do
Implement interactive canvas space for graphing	100%	25%	25%	25%	25%	N/A
Implement basic animations in graphing space	100%	20%	35%	20%	25%	Add more as features arrive
Tie text based program version to visual version	70%	20%	20%	20%	40%	Continue to improve connection
Implement basic DFA minimization functionality	100%	20%	30%	20%	30%	Improve accuracy and user experience.
Update Readme file to include new feature information	100%	40%	40%	10%	10%	Continue to expand

Milestone 4 Plan

Task	Chris	Vincent	Andrew	Keegan
Refine and expand minimization	Bug Fixer/Code Contributor and designer	Bug Fixer/Code Contributor and designer	Co-Lead coder and development head	Co-Lead coder and development head
Develop graph builder based on submitted string	Bug Fixer/Code Contributor and researcher	Bug Fixer/Code Contributor and researcher	Co-Lead coder and development head	Co-Lead coder and development head
Start developing “teacher mode” for user interactive DFA building	Bug Fixer/Code Contributor and researcher	Bug Fixer/Code Contributor and researcher	Co-Lead coder and development head	Co-Lead coder and development head
Heavily bug test and ensure standards of current and Milestone 4 features	Bug tester/code refining	Bug tester/code refining	Problem Identifier/Code Refiner	Problem Identifier/Code Refiner
Updated “Read Me” File	Co-Writer	Co-Writer	Code Side implementation	Code Side implementation

Questions?

Visit Our Site

https://kmcnear2022.github.io/