SlangOGi

EECS 510, Theory of Computing – Final Project

Dr. Jennifer Lohoefener

Designed by: Kundana Dongala, Navya Nittala, and Nimra Syed

Part 1: Designing SlangOGi

Intent

The SlangOGi is designed to model a Deterministic Finite Automaton (DFA) that processes and validates slang sequences, such as abbreviations and emojis, commonly used in digital communication. Its main goal is to provide a flexible framework for identifying valid slang and ensuring it follows predefined rules.

Using states and transitions, the class processes input step by step, validating each slang term or emoji consistently. It is designed for scalability, allowing new slang terms or transitions to be added easily. The intent is not just to recognize current slang but to adapt as online communication evolves.

Structure

The SlangOGi effectively captures the components of a DFA to validate slang sequences. Its structure emphasizes simplicity and adaptability, with components organized using Python's dictionaries and lists to ensure quality and efficiency:

- States List: This list enumerates all the states the DFA can occupy: the states include:
    - Start: The initial state where the input processing begins, expecting an abbreviation or slang term.
    - Middle: An intermediate state encountered after recognizing valid slang, allowing further abbreviations.
    - Emoji: A specified state for detecting emoji-like inputs, marking the transition toward accepting state.
    - Accept: The final state indicating the sequence is valid.

- Transitions Dictionary: This dictionary defines state transitions, mapping the current state and input to the next state. For instance, transitions include moving from *Start* to *Middle* upon recognizing an abbreviation and from *Middle* to *Emoji* upon encountering an emoji. Once in the *Emoji* state, the DFA transitions to *Accept*, finalizing the validation.
- Acceptance state: This designated state signifies the successful processing of the input sequence. The DFA concludes in this state to validate that all inputs adhere to the predefined rules.

This modular design ensures efficient slang sequence validation and supports easy integration of new states or transitions, making it adaptable to evolving communication trends.

Purpose

The SlangOGi DFA serves as an educational resource aimed at facilitating a deeper understanding of formal languages and automata theory. Initially developed to validate sequences of slang, the DFA exemplifies the practical applications of automata in addressing real-world challenges. By simulating the recognition of patterns in informal communication, it lays a foundational framework for the design of more sophisticated systems, such as chatbots.

Future enhancements to the SlangOGi DFA could broaden its scope to process and interpret human communication within informal contexts, including social media and messaging platforms. Such advancements could enable chatbots to accurately identify slang, emojis, and conversational tone, allowing for more natural and contextually appropriate responses. By integrating formal computational models with the nuances of human interaction, the SlangOGi DFA demonstrates the potential of automata theory to improve user experiences and contribute to the development of intelligent, responsive technologies.

Part 2: Grammar

The grammar for SlangOG is a context free grammar that outlines the rules for generating valid sequences of slang abbreviations followed by emojis. Here is the structured breakdown:

Components of the Grammar:

1. Start Symbol (S): represent the start state from which all valid sequences are derived.
2. Non-Terminals (S, Phrase, Abbr, Emoji): denote constructs for phrases, abbreviations, and emojis.
3. Terminals:
    - Abbreviations (LOL, OMG, SMH, IDK) and emojis (😂 , 🤦, 🙄, 😱), representing the finite set of symbols in the language.
4. Production Rules:
    - S → Phrase Emoji: Emojis are only allowed at the end of the sequence.
    - Phrase→Abbr | Abbr Abbr | Abbr: A phrase can be a single abbreviation, multiple abbreviations, or an abbreviation followed by an emoji.
    - Abbr→LOL | OMG | SMH | IDK: All valid abbreviations for the language.
    - Emoji→😂| 🤦| 🙄| 😱: Defined emojis forming part of the valid terminal symbols.

Valid and Invalid Test Case Examples:

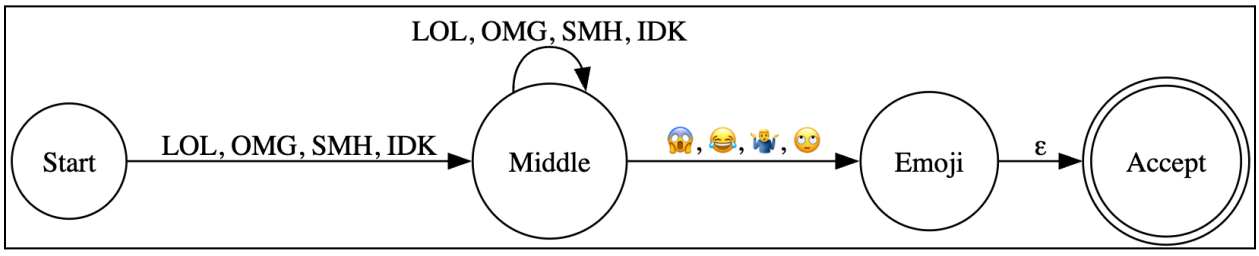
- Valid Sequences:
    1. ["LOL", "😂"]: A single abbreviation followed by its paired emoji.
    2. ["IDK", "🤦"]: A single abbreviation followed by its paired emoji.
    3. ["LOL", "OMG", "SMH", "😱"]: Multiple abbreviations followed by a single emoji that matches one of the abbreviations.

- Invalid Sequences:
    1. [“SMH” , “🎁”]: Ensure the user provides the correct emoji, as the code is designed to detect valid emojis that match the corresponding abbreviation. If an invalid emoji is provided, the code will display a "reject" message.
    2. [“IDK”, “🤦‍♂️”]: Make sure the user uses a standard yellow emoji(🤦‍♂️) instead of a personalized one with a skin tone.
    3. ["😂","SMH", "OMG"]: Emoji are in the beginning of the sequence, violating the rule that emojis must appear only at the end.
    4. ["SMH", "😂", "OMG"]: Emoji is in the middle of the sequence, violating the rule that emojis must appear only at the end.

Conclusion:

The grammar for SlangOGi models modern, informal communication patterns where the order of abbreviations and emojis matter. Enforcing an order where abbreviations come before emojis models a tonal marker to better understand the message being passed. By validating sequences using these rules, the context free grammar highlights how formal language can be applied to structure and analyze informal expressions effectively.

Part 3: Automaton



States

1. Start State (labeled "Start"): This is the initial state of the automaton.

- Transition: Moves to the "Middle" state upon encountering a valid slang word from the defined set.

2. Middle State (labeled "Middle"): This state represents one or more slang words
    - Self-loop: Allows multiple slang words in sequence
    - Transition: Moves to the "Emoji" state upon encountering a valid emoji.
3. Emoji State (labeled "Emoji"): This state is reached when the automaton recognizes an emoji following the slang words.
    - This state transitions to the "Accept" state upon encountering the end of the input (denoted by ε, representing no more input).
4. Accept State (labeled "Accept"):
    - This is the final accepting state of the automaton, indicating that the input sequence is valid according to the defined rules.

Transitions

1. From Start to Middle**:**
    - Condition: Recognizes a valid slang word (LOL, OMG, SMH, IDK).
    - Action: Begins recognition of a valid input sequence.
2. Within Middle (Loop)**:**
    - Condition: Recognizes additional slang words
    - Action: Allows the automaton to process multiple slang words in order.
3. From Middle to Emoji:
    - Condition: Recognizes a valid emoji (😂,😱,🙄,🤦‍♂️).
    - Action: Moves the automaton to the emoji state
4. From Emoji to Accept:

- Condition: Recognizes the end of input (ε)
- Action: Indicates that the sequence is complete and valid, leading to acceptance.

Description of Functionality

This automata models sequences where:

1. A valid sequence begins with one or more slang words (LOL, OMG, SMH, IDK).
2. Slang words can repeat any number of times.
3. Emoji (😂,😱,🙄,🤦‍♂️) must appear only after all slang words.
4. The sequence is valid if it ends with one emoji and transitions to the "Accept" state.

Examples of valid and invalid input are described earlier in Part 2.

Part 4: SlangOGi Data Structure

Class Purpose:
SlangOgi is designed to represent a Deterministic Finite Automata (DFA) that can recognize valid sequences of online chat abbreviations and match them with the correct emoji

Key Components:

- **States**: The states list (self.states) contains all the possible states that the DFA contains:
  - “Start”
  - “Middle”
  - “Emoji”
  - “Accept”: an implicit state that can indicate a valid end state for the sequence
- **Transitions**: The transitions dictionary (self.transitions) defines how the DFA moves between states based on the input:
  - For example, if the current state is “Start” and the input is something like “LOL”, “OMG”, “SMH”, or “IDK”, the DFA will transition to the “Middle state”, and so on for the others.
- **Current State**: The current state (self.current_state) represents where the DFA is in the process of analyzing the input. It starts off in the “Start” state

- **Valid Pairs**: The valid pairs (valid_pairs) keeps track of valid abbreviation-emoji pairs, ensuing that the emoji that follows an abbreviation is valid

Methods:

- **Reset Method:** This method (self.reset) is responsible for resetting the DFA to its initial state, allowing it to start fresh with a new input sequence
- **Process Input Method (process_input)**: This method processes a sequence of input symbols, transitioning between states based on the defined rules. It ensures that the sequence ends in the "Emoji" state, and that the final emoji corresponds to the abbreviation before it, making the sequence valid

Please refer to the python class in main.py.

Part 5: Testing Function

Function Purpose:

The test_dfa function is used to evaluate the performance of the SlangOGi DFA by testing it with different input sequences to see whether it accepts or rejects them.

Parameters:

- **dfa**: An instance of the SlangOGi class, representing the DFA to be tested.
- **test_cases**: A list of input sequences (each one is a list of strings) that will be tested against the DFA

Functionality:

- **Results Dictionary**: The function initializes an empty dictionary (results) to store the outcomes of each test case
- **Input Processing**: It goes through each input sequence in test_cases, calling the process_input method on the DFA instance with the current sequence. This will determine if the sequence is accepted or rejected by the DFA.
- **Storing Results**: For each input sequence, the result ("accept" or "reject") is stored in the results dictionary, using the input sequence (joined into a string) as the key

Return Value: The function returns the results dictionary, which contains the results for all test cases, mapping each input sequence to its corresponding outcome

Please refer to the test_dfa function in main.py.