Spreadsheet Project

Rajat Soni 2023CS10229
Krish Bhimani 2023CS10712
Priyanshu Gaurav 2023CS10129

April 25, 2025

Contents

1 Design and Software Architecture

The design of the spreadsheet project is modular, with clear separation of concerns between different components.

1.1 Directory Structure

The project is organized into the following directory structure:

- `src/`
  - `main.rs` - The entry point of the application, containing the main function.
  - `utils/` - A directory containing utility modules for various functionalities.
    - `display.rs` - Handles the display logic for the spreadsheet.
    - `input.rs` - Processes user input and validates formulas.
    - `operations.rs` - Implements core spreadsheet operations like addition, subtraction, etc.
    - `toposort.rs` - Contains the implementation of Kahn's algorithm for topological sorting.
    - `gui/` - A subdirectory for GUI-related modules.
      - `gui.rs` - Manages the graphical user interface using the `egui` framework.
      - `loadnsave.rs` - Handles loading and saving spreadsheet data.
      - `plot.rs` - Provides functionality for generating plots and charts.
      - `stats.rs` - Implements statistical functions for the spreadsheet.
      - `assets/` - A directory for storing GUI assets like icons and styles.
- `Cargo.toml` - The configuration file for managing dependencies and project metadata.
- `Cargo.lock` - A lock file to ensure reproducible builds by locking dependency versions.
- `README.md` - A markdown file providing an overview and instructions for the project.
- `report.tex` - The LaTeX file for generating the project report.
- `Makefile` - A file containing build automation rules for the project.

1.2 Command-Line Interface (CLI) Decisions

The design of the Command-Line Interface (CLI) was guided by the following considerations:

- **Manual Input Handling:** The application processes user input manually instead of relying on regular expressions. This approach ensures better performance and avoids the computational overhead associated with regex matching, especially for complex patterns. By directly parsing and validating the input, the system achieves greater control over the input handling process, reducing the likelihood of errors and improving efficiency.
- **Adjacency List for Circular Dependency Detection:** To manage and detect circular dependencies between cells, the application maintains an adjacency list representation of the dependency graph. Each cell is represented as a node, and its dependencies are stored as edges in the list. This structure allows efficient traversal and cycle detection, ensuring that invalid states are identified and prevented during formula updates.
- **Kahn's Algorithm for Topological Sorting:** The application uses Kahn's algorithm to perform topological sorting of the dependency graph. This ensures that cells are recalculated in the correct order, respecting their dependencies. This approach is efficient and ensures that the spreadsheet's dependency graph is processed correctly, avoiding invalid states caused by circular dependencies.

1.3 Graphical User Interface (GUI) Decisions

The design of the Graphical User Interface (GUI) was influenced by the following factors:

1.3.1 Integration of Egui Eframe and CLI

The `egui` framework, provided by `eframe`, is used to build the graphical user interface (GUI) for the spreadsheet application. This framework allows for the creation of a responsive and interactive interface with minimal overhead. The integration between the GUI and the Command-Line Interface (CLI) is achieved as follows:

- **Input Redirection:** User inputs made through the GUI, such as entering formulas or updating cell values, are forwarded to the CLI for processing. This ensures that the same logic used for CLI interactions is reused, maintaining consistency across both interfaces.
- **Processing and Feedback:** The CLI processes the inputs, validates them, and updates the underlying data structures. Any errors or feedback, such as invalid formulas or circular dependency warnings, are returned to the GUI for display.
- **GUI Updates:** After processing, the GUI is updated to reflect the changes made to the spreadsheet. This includes recalculating dependent cells, updating displayed values, and highlighting any errors.

This design ensures that the GUI remains lightweight and focuses on user interaction, while the CLI handles the core logic and data processing.

1.3.2 Libraries Used in the Project

The project leverages several Rust libraries to enhance functionality and streamline development:

- `csv`: This library is used for handling CSV files. It provides functionality for reading and writing CSV data, which is useful for exporting spreadsheet data to a standard format. However, due to the limitations of the CSV format in preserving formulas and dependencies, loading from CSV was not implemented.
- `genpdf`: The `genpdf` library is used for generating PDF reports. It allows for the creation of structured and styled documents, making it ideal for exporting spreadsheet data or generating summaries.
- `plotters`: This library is used for creating plots and charts within the spreadsheet application. It supports a wide range of chart types, enabling users to visualize data effectively.
- `serde`: The `serde` library is used for serialization and deserialization of data. It facilitates saving and loading spreadsheet data in formats like JSON, ensuring that the application's state can be preserved and restored efficiently.

These libraries were chosen for their performance, ease of use, and compatibility with the project's requirements.

2 Why Proposed Extensions Could Not Be Implemented

2.1 Cannot Implement Range Copy-Paste

Copying and pasting a range of formulas introduces several challenges:

- **Relative Shifting of Formulas:** When formulas are copied, their cell references need to be adjusted relative to the new location. This requires parsing and modifying each formula, which can be computationally expensive for large ranges.

- **Cycle Detection:** Adjusted formulas may inadvertently create circular dependencies. Detecting such cycles in a spreadsheet with numerous cells is computationally intensive and requires a robust dependency graph analysis.
- **Reverting Changes:** If a cycle or invalid state is detected after pasting, reverting the spreadsheet to its previous state is non-trivial. This would require maintaining a history of changes, which adds further complexity.

Given these inefficiencies and the potential for introducing bugs by our algorithm, we decided to defer the implementation of this feature for now.

2.2 Cannot Implement Sleep Timer

- **Thread Blocking:** Implementing a sleep timer would block the main thread, halting all other operations in the spreadsheet application during the sleep duration.
- **User Interface Responsiveness:** A blocked thread would make the application unresponsive, leading to a poor user experience.
- **Concurrency Management:** To avoid blocking, the sleep timer would require multithreading or asynchronous programming, which adds significant complexity to the codebase.
- **Timer Display Challenges:** Displaying a countdown timer while the thread is paused is not feasible without additional mechanisms to update the user interface asynchronously.

Adding a sleep timer would require integration with asynchronous event handling. This feature was deprioritized as it was not critical to the core functionality of the spreadsheet application.

2.3 Cannot Implement Load from CSV

Loading data from a CSV file was not implemented due to the following reasons:

- **Inability to Save Formulas:** The CSV format is inherently designed for storing plain data and does not support the storage of formulas or their dependencies. As a result, only raw data could be saved, making it impossible to reconstruct the spreadsheet's computational logic when loading from a CSV file.
- **Loss of Context:** Without formulas, the relationships and dependencies between cells would be lost, leading to an incomplete and potentially incorrect representation of the spreadsheet.

Given these limitations, we decided to prioritize other features and defer the implementation of loading from CSV files.

3 Possibility of Additional Extensions

While some features could not be implemented in the current version, there are several possibilities for future extensions to enhance the functionality of the spreadsheet application:

3.1 Advanced Formula Support

- **Feature Description:** Support for more complex formulas, including statistical, financial, and logical functions.
- **Implementation Challenges:** This would involve extending the formula parser and ensuring compatibility with the dependency graph.

3.2 Password Protected Spreadsheet

- **Feature Description:** Enable users to protect their spreadsheets with a password, restricting unauthorized access.
- **Implementation Challenges:** This would require implementing encryption mechanisms to securely store the spreadsheet data and password. Additionally, user authentication and password recovery mechanisms would need to be designed.

3.3 Automatic Sequence Generation

- **Feature Description:** Automatically generate sequences (e.g., numbers, dates, or custom patterns) in selected cells.
- **Implementation Challenges:** This would involve designing a user-friendly interface for specifying sequence rules and ensuring compatibility with existing cell data and formulas.

4 Primary Data Structures

The spreadsheet application relies on several primary data structures to manage and organize data efficiently:

4.1 Vectors for Database and Error Flags

- **Database Storage:** The application uses vectors to store the spreadsheet's database. Each vector represents a row, and the elements within the vector correspond to the cells in that row. This structure allows for efficient indexing and manipulation of cell data.
- **Error Flags:** Vectors are also used to maintain error flags for each cell. These flags indicate whether a cell contains a division by zero error, enabling quick error reporting.

4.2 Vectors of Vectors for Sensitivity List

- **Graph Representation:** The dependency graph's sensitivity list is implemented as a vector of vectors. Each vector corresponds to a cell, and its elements represent the cells that depend on it.
- **Purpose:** This structure facilitates efficient traversal of dependencies during recalculation, ensuring that only affected cells are updated when a change occurs.

4.3 Structs for GUI State Management

- **State Representation:** The state of the graphical user interface (GUI) is managed using structs. These structs encapsulate various aspects of the GUI, such as the current selection, input focus, and user preferences.
- **Integration with Egui Eframe:** The GUI state is seamlessly integrated with the `egui` framework provided by `eframe`. This allows for reactive updates and ensures that the interface remains responsive to user interactions.

5 Interfaces Between Software Modules

The interfaces between software modules are designed to ensure seamless communication and functionality:

5.1 Input Module and CLI for Formula Processing and Cell Updates

- **Formula Processing:** The Input module is responsible for processing formulas entered by the user. It parses the input, validates the syntax, and generates the necessary instructions for updating the spreadsheet.
- **Cell Updates in CLI:** The Command-Line Interface (CLI) handles cell updates by decoding the instructions generated by the Input module. The same function is used for updating cells, ensuring consistency between formula processing and cell updates.

5.2 GUI Integration with CLI

- **GUI Framework:** The graphical user interface (GUI) is built using the `egui` framework provided by `eframe`. This framework enables the creation of a responsive and interactive interface.
- **Forwarding Updates to CLI:** Updates made through the GUI are forwarded as terminal inputs to the CLI. The CLI decodes these inputs and updates the cells using the same function used for direct CLI interactions. **This ensures that the GUI is as robust as CLI**.

6 Approaches for Encapsulation

Encapsulation is achieved through the following approaches:

6.1 Encapsulation with Structs

The application uses structs to encapsulate data and functionality. Each struct represents a specific module or component, ensuring that internal details are hidden from other parts of the application.

6.2 Public Functions

Only the necessary functions are made public, exposing a minimal interface for interacting with the struct. This prevents unintended modifications and ensures that the internal state remains consistent.

7 Justification for Design Choices

The design of the spreadsheet application is justified as follows:

- **Manual Input Handling:** Instead of relying on regular expressions for parsing user input, manual input handling is implemented. This approach ensures better time efficiency by directly processing the input without the overhead of regex matching, which can be computationally expensive for complex patterns.
- **Adjacency List for Circular Dependency Check:** To detect circular dependencies in the spreadsheet, an adjacency list is maintained. This data structure allows efficient representation of the dependency graph and facilitates quick traversal to identify cycles.
- **Kahn's Algorithm for Topological Sorting:** Kahn's algorithm is used for topological sorting of the dependency graph. This ensures that cells are recalculated in the correct order, improving time efficiency during updates. The algorithm processes nodes with zero in-degree iteratively, minimizing redundant computations.
- **Lightweight GUI Framework:** The `egui` framework provided by `eframe` is used for building the graphical user interface. This lightweight framework is chosen for its simplicity and responsiveness, ensuring a smooth user experience.

- **Forwarding GUI Inputs to CLI:** To ensure robustness, inputs made through the GUI are forwarded to the Command-Line Interface (CLI). This design ensures that the same logic is used for processing inputs, maintaining consistency and reducing the likelihood of discrepancies between the GUI and CLI functionalities.

8 Design Modifications During Development

During the development process, some modifications were made to the initial design:

- **User Interface Adjustments:** The GUI module was modified frequently to provide a more intuitive and responsive interface.

9 Links and References

- Github Repo: `https://github.com/rajat-soni07/rust-spreadsheet`