COP290 Rust Lab

Arjun Sammi: 2023CS50163; Popat Nihal Alkesh: 2023CS10058; Viraaj Narolia: 2023CS10552

April 2025

1 Introduction

1.1 How to use our TUI

In normal mode:

- e: enter edit mode
- w,a,s,d: move across spreadsheet
- up and down arrow keys: scroll command history
- left and right arrow keys: change spreadsheet
- q: quit instance

In edit mode:

- esc: enter normal mode
- up and down arrow keys: autofill command from history

2 Command Syntax Table

Command	Description
:q	Quit the editor
:w	Move cursor up
:a	Move cursor left
:s	Move cursor down
:d	Move cursor right
:scroll_to <cell>	Scroll to bring cell into view
:enable_output	(Does not apply, does nothing)
:disable_output	(Does not apply, does nothing)
:copy_cell_value <src> <dst>	Copy value from src to dst cell
:copy_cell_formula <src> <dst>	Copy formula from src to dst cell
:copy_range_values <src_start>:<src_end> <dst_start>	Copy values from a range to a starting destination
:copy_range_formulas <src_start>:<src_end> <dst_start>	Copy formulas from a range to a starting destination
:add_sheet <name>	Add a new sheet named name
:remove_sheet <name>	Delete sheet named name
:rename_sheet <old> <new>	Rename sheet old to new
:dup_sheet <source> <copy>	Duplicate source into copy
:resize <width> <height>	Resize spreadsheet to dimensions width x :height
:load_csv <sheet> <path>	Load a CSV file at path into sheet
:export_csv <sheet> <path>	Export sheet contents to a CSV at path
:undo	Undo the last action
:redo	Redo the last undone action
:make_chart <start1>:<end1> <start2>:<end2>	Create chart from selected ranges
autofill_ap <start_addr>:<end_addr>	Autofill arithmetic progression across range based on f values
autofill_gp <start_addr>:<end_addr>	Autofill geometric progression across range based on first ues

3 Why proposed extension could not be implemented?

3.1 Ability to add/delete individual rows/columns

We deliberately did not implement this, because according to our implementation, we would have to iterate through every cell of every sheet in order to add/remove a single row/column. This is because we are storing addresses of cells depending on the current cell in its struct. Implementing this would bloat the time complexity by a lot.

3.2 Relative and absolute addressing

This would have required parsing and converting expressions differently during evaluation, along with additional metadata for each cell reference. Due to time constraints and complications with expression parsing, this was skipped.

3.3 Non evaluated cell functions

We later believed this extension was not impressive enough to implement. Evaluating cells lazily would also introduce space complexity in dependency management.

3.4 Integrating ML models as formula

We lacked the time and resources to safely integrate ML models into our Rust-based formula engine, and the utility was limited in the current TUI setup.

3.5 Branching in undo-redo

This would require a tree-like structure to represent user state history. While we have basic undo-redo, supporting branching would add significant overhead and was not feasible within project constraints.

3.6 'help' and 'man' commands

Basic help is implemented, but comprehensive man-page style documentation is missing due to time constraints.

3.7 Advanced Find and Replace with regex

Regex integration was deprioritized because of time limits and since basic find/replace met user needs.

3.8 Labelling of ranges

This was deprioritized as it did not add much value for a terminal-based spreadsheet.

3.9 Macros

Too complex to implement safely in the time provided. Also posed risks of circular macro definitions and hard-to-debug states.

4 Could we implement extra extensions over and above the proposal?

Yes, we were able to implement 6 additional extensions:

- Conditional range functions
- Copy paste cell/range value/expression
- Graph
- Bool data type and its operations
- Command history
- TUI

Thus, out of 16 proposed extensions, we could implement 8, along with 6 extensions above and beyond the proposal.

5 Explanation of Implemented Extensions

5.1 CSV Import and Export

We implemented support for importing spreadsheets from CSV files and exporting the current spreadsheet to CSV. This allows interoperability with other spreadsheet applications like Excel or Google Sheets.

5.2 Variable Size Spreadsheet

To optimize memory usage, we avoid pre-initializing every cell in the spreadsheet. Instead, cells are created only when accessed or assigned. Additionally, the spreadsheet can dynamically resize with user commands.

5.3 Multiple Sheets

Our system supports creating and managing multiple sheets. Users can add, delete, duplicate, and rename sheets. Each sheet maintains its own cells and formulas.

5.4 Global and Local Addressing

We support both global and local cell addressing. Global addresses can reference cells across sheets, while local addresses refer to cells in the current sheet context. (Format for global addressing: sheetname.A1 etc.)

5.5 Autofill

Autofill is supported for arithmetic and geometric progressions. However, we could not implement relative autofill for expressions due to time constraints.

5.6 Complex Expressions

Our expression parser supports complex nested formulas, allowing the use of nested function calls, arithmetic, logic, and conditional operations in the same expression. This is powered by a lexer and parser which generates an ast.

5.7 Ternary Operator in Formula

We introduced a ternary operator (`cond ? expr1 : expr2`) for conditional expressions inside formulas, enhancing expressiveness and control.

5.8 Conditional Range Functions

Functions such as `SUMIF` and `COUNTIF` were implemented. These allow users to operate over a range based on conditional logic.

5.9 Formula Display Functionality

When a cell is selected, its formula or value is shown in a side panel, making it easier for users to understand the logic behind computed results.

5.10 Undo/Redo

Basic undo and redo functionality is implemented. Due to time constraints, only assignment-type commands can be undone and redone.

5.11 Graphing Support

We support basic graph rendering using ASCII. Currently, only scatter plots are available, due to limited time for implementing more chart types.

5.12 Copy-Paste (Cell and Range)

Users can copy and paste both individual cells and ranges. The system preserves whether the pasted content is a value or a formula. During copy-paste, local references shift appropriately, while global references remain unchanged.

5.13 Data Types

The spreadsheet supports multiple data types including integers, floats, strings, and booleans. Type-aware functions are provided to operate on each data type.

5.14 Command History

Users can navigate through previously entered commands using the arrow keys, making interaction efficient and smooth.

5.15 Command-Line Graphical UI (CLGUI)

A terminal-based user interface was implemented using `crossterm` and `tui-rs`. The interface mimics a VIM-like interactive spreadsheet environment.

6 Primary data structures

6.1 BTreeSet

Used to store cells that depend on the current cell. It is used for topological sorting, which gives order of evaluating cells on every update.

6.2 Hash Table

Hash table is used to store visited cells and detect cyclic dependency in the DFS algorithm used for topological sort.

6.3 Stack

Stack is used to store the order of cells returned by DFS.

7 Interfaces between software modules

Our project was divided into several modules with clear interfaces:

- **ast.rs**: Abstract Syntax Tree definitions for expressions.
- **cell_operations.rs**: Struct implementations of cell, column, and sheet.
- **evaluate_operations.rs**: Functions to evaluate parsed expressions.
- **grammarcmds.lalrpop**, **tokenscmds.rs**: Parser for spreadsheet commands.
- **grammarexpr.lalrpop**, **tokensexpr.rs**: Expression grammar and token definitions.
- **graphic_interface.rs**: Code for rendering ASCII graphs and charts.
- **lexer.rs**: Tokenizes input commands and expressions.
- **main.rs**: Main file that connects all modules and initializes the program.

These modules interact via well-defined structures and function calls, ensuring modularity and maintainability.

8 Conclusion

We designed and implemented a modular, feature-rich spreadsheet application in Rust, with a clean TUI, a functional parser and evaluator, and a powerful expression system. Despite time and complexity constraints, we successfully implemented most of the core features and several advanced ones. The system was tested and verified for accuracy and performance. Future work could include GUI integration, improved memory handling, and collaborative editing features.