MS3 Report - Group 57

Vision

Our vision for this project is to build a fully interactive Propositional Logic Solver that allows users to construct and explore formal proofs step by step. Since MS2, this vision has expanded from a simple Modus Ponens solver into a more complete proof engine capable of complex inference and derivation. To support our goal of having an interactive educational tool, we expanded our architecture to include more complete functionality that integrates with the original system. Our code is also now more organized, as we implemented proper development best practices and documentation to ensure maintainability.

Summary of progress

Between MS2 and MS3, our team transformed an early prototype into a functional propositional logic solver with significantly expanded capabilities. At MS2, we had implemented the AST for formulas, a working parser, support for modus ponens, and a REPL for pattern matching parsed expressions. For MS3, we clearly defined and built out the supported rules for a complete logic solver.

- We made sure our code implemented best development practices, implementing OCaml Doc clientside and implementerside specifications, .mli files, and ensuring proper separation of concerns.
- We implemented multiple inference rules, including Modus Tollens, conjunction introduction, double negation, contraposition, and more, integrating them into a general purpose derivation engine that repeatedly applies all available rules until no new information can be inferred.

- We expanded the frontend to support more commands, such as the option to display all supported derivation logic. There is also now the option to remove specified premises, remove just the goal, or show your session statistics (number of premises, goal reached, etc.), allowing the user for a more precise user experience and a clearer understanding of their proof state.
- We added Bisect and a proper test suite to ensure full line coverage and cohesive black box system tests.

Activity breakdown

Anna

Responsibilities

- Implement Modus Tollens logic
- Add .mli interface files across core modules

Activities & Features Delivered

- Implemented Modus Tollens (¬B and A → B ⇒ ¬A) as a new inference rule in rule.ml and exposed it in rule.mli
    - Added proper test cases for full coverage
- Created .mli interface files for:
    - ast.mli, parser.mli, rule.mli, sequent.mli, providing clearer module boundaries and documentation

Hours Spent: 3-4

Shirley

Responsibilities

- Implement Conjunction Introduction logic

- Programming guidelines and separation of frontend and backend
- Optimize frontend system to ensure it prints out concise derivation process messages

Activities & Features Delivered

- Implemented Conjunction Introduction (From A and B, derive A & B) in sequent.ml file and added helper functions
    - Added proper test cases for full coverage
    - Individuals premises are now automatically added to derived proposition list in proof state if conjunction formula is input
- Documented OCamldoc specifications and comments (including preconditions and postconditions) for interface files:
    - ast.mli, parser.mli, rule.mli, sequent.mli
- Modify frontend code to get rid of redundant conjunction introduction and avoid repeating derived formulas

Hours Spent: 3-4

Tyson (Frontend / CLI + Input Validation + Conflict Handling)

Responsibilities

- Expand frontend to fit new backend logic
- Add double negation logic rule (!!A ⇒ A)
- Fix bug where users could add conflicting premises into the proof state

Activities & Features Delivered

- New recursive handling of negation (!!!A ⇒ !A)
    - Added test cases for full coverage
- User's premise is now gracefully rejected if it conflicts with any already-existing premises or derivations

- Cleaner frontend that matches the scope of our backend and needs of our user
- Implement fix so premise removal updates sequent state

Hours Spent: 3-4

Nicole

Responsibilities

- Implement Hypothetical Syllogism and Contraposition inference rules
- Add more general functionality regarding derivations and explanations
- Set up comprehensive testing framework with Bisect code coverage
- Add file operations (load/export) and statistics features

Activities & Features Delivered

- Implemented Hypothetical Syllogism (From A -> B and B -> C, derive A -> C) and Contraposition (From A -> B, derive !B -> !A) in rule.ml
- Created `apply_all_rules` function that exhaustively applies all inference rules until no new formulas can be derived
- Implemented `find_derivations` function to discover possible ways to derive a given formula
- Enhanced derivation explanation system to show which rule was used for each derivation
- Added file loading functionality (`load` command) to run proof scripts from files
- Added state export functionality (`export` command) to save proof state to files
- Implemented `stats` command to display proof statistics (premise count, derived count, goal status)
- Set up Bisect_ppx for code coverage analysis and achieved 86.20% coverage
- Expanded test suite from 119 to 147 tests, adding comprehensive tests for AST utilities and Sequent operations

- Fixed tautology filtering bugs and implemented filtering to prevent excessive formula generation

Hours Spent: 3-4

Productivity analysis

Overall, our team remained very productive throughout MS3, even as the project became more complex and time consuming. Compared to earlier milestones, this sprint required a lot more coordination because we needed to expand our scope from MS2, fix some of our MS2 implementation, and get our code ready for submission. Despite the increased difficulty, our task delegation continued to be effective, and we were able to work asynchronously with minimal merge conflicts. We did this by planning internal deadlines for each member and ensuring that we could all work, even if we were waiting for another member's push. Our deadlines were reasonable, although some things took longer than usual. For example, writing specification comments took a significant amount of time, as we had to write them for our old MS2 code and also the new, extended MS3 code. Communication remained strong throughout the sprint, and we consistently used Slack to communicate new pushes, ensure code consistency, and keep the integration process smooth. Overall, we met all of our planned MS3 goals and maintained great team productivity in the process.