Interfaz Web para Maude
Maude Web Interface

Trabajo de Fin de Grado
Curso 2023–2024

Autor
Beatriz Carrancio Casero

Director
Adrián Riesco Rodríguez

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Interfaz Web para Maude
Maude Web Interface

Trabajo de Fin de Grado en Ingeniería Informática

Autor
Beatriz Carrancio Casero

Director
Adrián Riesco Rodríguez

Convocatoria: Septiembre2024

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

3 de septiembre de 2024

Dedicatoria

A todos los que han puesto su granito de arena.

Resumen

Interfaz Web para Maude

Este proyecto busca una forma alternativa más accesible de utilizar el lenguaje de programación Maude.

Actualmente, el diseño de este leguaje está orientado prinicpalmente para entornos de Linux y macOS, y puede resultar menos accesible para algunos desarrolladores. A través de una interfaz web, este programa ofrece una alternativa para utilizar los módulos y comandos de Maude, evitando al usuario todo el proceso de instalación y configuración de softwares adiccionales y simplificando su uso. También permite al usuario iniciar sesión y almacenar el historial de comandos y módulos, facilitando el seguimiento de fallos recurrentes, la identificación de patrones y el análisis de estadísticas de errores.

Palabras clave

Lenguaje de Programación Maude, Interfaz Web, Lógica de Reescritura, Accesibilidad.

Abstract

Maude Web Interface

The aim of this project is to create an alternative and more accessible way of using the Maude Programming Language.

Currently, Maude is primarily designed for Linux and macOS environments, which can make it less approachable to some developers. Through a web interface, this program offers another way of using Maude modules and commands while sparing the user of the whole process of installing and setting up any needed software and simplifying its use. It also allows the user to log in and store command and module history, facilitating the tracking of recurring mistakes, identification of patterns, and analyzing error statistics.

Keywords

Maude Programming Language, Web Interface, Rewriting Logic, Accessibility.

Contents

List of figures

List of tables

Chapter 1

Introduction

"Overcome space, and all we have left is Here. Overcome time, and all we have left is Now."
— Richard Bach

This thesis introduces the "Maude Web Interface" project, designed to make the Maude programming language more accessible to a wider audience. Its original design, primarily for Linux and macOS, can create a barrier for those unfamiliar with these environments or for those who prefer web-based platforms.

This project addresses this challenge by developing a web interface that provides an alternative and user-friendly way to interact with the Maude System. By eliminating the need for local installation and configuration, this web interface aims to simplify access for users, particularly students.

The web application itself can be accessed through this link:

`http://maudeonline.atwebpages.com/`

All source code implemented for this project is publicly available in the following GitHub repository:

`https://github.com/bcarrans/maude_interface.git`

1.1. Motivation

The Maude Programming Language is an advanced tool that uses rewriting logic to perform formal modeling, specification, and system analysis (Clavel et al., 2007). It is a valuable tool for researchers and developers working in fields that require a stricter accuracy. However, despite its strengths, Maude's usability can be limited by its reduced compatibility with certain operating systems. This can create a significant barrier for developers and students who may not be familiar with these

environments or who prefer to work in more widely-used platforms like Windows or through web-based tools.

Given the growing demand for cross-platform accessibility and the increasing popularity of web-based applications, particularly in the educational sector, this project seeks to bridge the gap between Maude's strengths and its setbacks. The interface will not only replicate the functionalities of the traditional Maude environment but also enhance the user experience by integrating additional features. One of the key enhancements is the ability to log in and store command and module history. This feature will allow users to track their previous commands, monitor recurring mistakes, identify learning patterns, and review error statistics. Such capabilities are particularly beneficial in educational contexts, where students can use these insights to improve their understanding of the language and refine their problem-solving skills.

Moreover, by providing a centralized online platform, the Maude Web Interface opens up new possibilities for collaborative learning and remote teaching. The interface could be used by instructors to demonstrate concepts, assign tasks, and provide feedback in real-time, all within a single, accessible environment. The online nature of the tool also ensures that it is always up-to-date and accessible from anywhere, further lowering the barriers to entry for new users.

The **"Maude Web Interface"** project represents a significant step toward making the Maude Programming Language more accessible and user-friendly. By transitioning to a web-based environment, this project not only simplifies the technical setup required to use Maude, but it could also enrich the user experience with features that support learning and productivity. This initiative is poised to expand the reach of Maude, making it more approachable for a broader audience, from students and educators to professional developers and researchers.

1.2. Objectives

The core objective of this project is to offer an alternative way to access Maude through a web interface, eliminating the traditional installation process. The key goals include:

- Developing a functional web interface.
- Implementing the interface to make web calls to Maude using Python bindings, displaying the results within the interface.
- Identifying the necessary Maude syntax for web implementation and ensuring readability.
- Analyzing and mitigating potential security vulnerabilities..

- Incorporating the necessary additional functionalities and features to transform the interface into a a comprehensive web application.
- Making the platform accessible to all users.

1.3. Work plan

The development of the project has been carried out in the following stages:

1. General study of the different possibilities of using the Maude programming language, assessing compatible operating systems, subsystems, or adaptations available for Windows. Evaluating the Windows Subsystem for Linux (WSL) as a potential environment for Maude.
2. Familiarization with the available resources: Review of the official Maude documentation to understand its usage and syntax to define possible use cases.
3. Detailed breakdown of the Maude Bindings developed by FADoSS, that allow the integration of Maude with other programming languages.
4. Study of the possible ways to apply these resources to the proposed concept considering the requirements, constraints, and potential benefits.
5. Implementation of the selected syntax.
6. Analysis of possible security breaches and application of risk mitigation strategies.
7. Testing and Debugging: Identify and resolve any issues or bugs that arise during the development process.
8. Deployment: Set up the production environment, ensure that all dependencies and configurations are correctly applied, and verify that the system operates as expected in the live environment

1.4. Development tools

The resources used in this project include the following:

- **Microsoft Visual Studio Code**, which serves as the primary development environment for all programming languages involved in the project.
- **XAMPP**, employed for comprehensive testing and local server management.
- **GitHub**, which facilitates version control and project management to ensure an efficient workflow, with **GitHub Desktop** used as a graphical interface to simplify version control tasks, and the integrated package management platform **GitHub Packages**, used for deployment arrangements.

- **Docker**, a containerization platform used in the deployment process.

The programming languages that were employed during the development include:

- **PHP**, a general-purpose scripting language that is especially suited for server-side web development, where it typically handles backend logic, like database interaction or form processing (Welling y Thomson, 2017).
- **HTML** (Hypertext Markup Language), a standard markup language which is used to structure the web page, defining the content and layout (Robbins, 2018).
- **JavaScript**, a high-level programming language that enables dynamic behavior on web pages, is used for frontend development, and interactive elements (Flanagan, 2020).
- **CSS** (Cascading Style Sheets), a styling language used to control layout, design and formatting of the web page (Meyer, 2018).
- **Python**, a high-level, general-purpose language responsible for handling the core functionality of the project: integrating Maude execution into the user interface. Its flexibility, readability and extensive collection of libraries, make it highly suitable for web development and integration with other systems or languages (Matthes, 2016).
- **SQL** (Structured Query Language), a standard programming language used for the relational database system design, data storage and retrieval operations and overall database management (Beaulieu, 2009).

Chapter 2

State of the Art

To better understand the problem this project is trying to address, the following section provides a general overview of The Maude System and its capabilities.

2.1. The Maude Programming Language

Maude is a high-performance reflective language and system supporting both equational and rewriting logic specification and programming for a wide range of applications (The Maude System, 2024). The system has been significantly influenced by the OBJ3 language, which is considered an equational logic sublanguage (Goguen y Malcolm, 1996). Besides supporting equational specification and programming, Maude also supports rewriting logic computation.

Rewriting logic is a logic of concurrent change that can naturally deal with state and with concurrent computations. It works as a general semantic framework for giving executable semantics to a wide range of languages and models of concurrency. In particular, it supports very well concurrent object-oriented computation. The same reasons making rewriting logic a good semantic framework make it also a good logical framework, that is, a metalogic in which many other logics can be naturally represented and executed (Meseguer, 1992).

One of the key features of Maude is its support for logical reflection, which is both systematic and efficient. This makes Maude remarkably extensible and powerful, supports an extensible algebra of module composition operations, and allows many advanced metaprogramming and metalanguage applications (Clavel et al., 1999). Some of the most interesting applications of Maude are metalanguage applications, in which Maude is used to create executable environments for different logics, theorem provers, languages, and models of computation.

Declarative programming as is, is commonly overlooked by many developers. While more experienced programmers may be familiar with it, it remains relatively

unknown to the average citizen developer. Maude is no exception. Aside from the fact that its syntax is complex and under documented, its lack of availability on Windows operating systems limits its already scarce use.

To actually broaden the access to this subject to a wider audience it would be useful to make it more approachable. There is a Windows version of Maude distributed by FADoSS, available at:

`http://moment.dsic.upv.es/`

Aside from this, there are no platforms or solutions readily accessible for Windows, although the Windows Subsystem for Linux (WSL) serves as a potential workaround for those who need to access it on a Windows environment (Maude Installation Guidelines, 2022). This seems to be the most widespread solution, since it avoids the necessity of a partition or other methods. The Figure 2.1 shows the Maude System in the WSL.

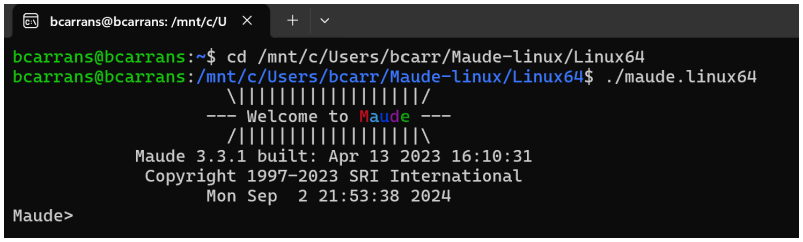

Figure 2.1: Maude in the Windows Subsystem for Linux

2.2. Online Compiling Platforms

Drawing inspiration from the online compiling platforms available for other programming languages seems like the best approach. These platforms allow users to write, compile, and execute code directly in their web browsers without the need for local installation, making them an invaluable resource for learning and experimentation.

Tools of this kind are widely available. **Replit** and **JSFiddle** are some examples of platforms that allow users to write and execute code in various programming languages. These platforms provide an interactive environment where users can experiment with code, see real-time outputs, and receive immediate feedback on errors. Another example is **JDoodle**, which also offers features such as code sharing, collaboration, and detailed error reporting.

The benefits of such platforms are manifold:

- **Accessibility:** They eliminate the need for complex installation procedures, making programming more accessible to beginners and those with limited technical expertise.

- **Convenience:** Users can quickly test snippets of code without setting up a full development environment.

- **Learning Support:** These platforms often include features like syntax highlighting, auto-completion, and error diagnostics, which help users learn and debug more effectively.

Unfortunately, only the most widespread programming languages are available in most of these platforms, and no such tool currently exists for Maude. This limits its accessibility, particularly for beginners who may be deterred by the initial setup process. Creating an online Maude compiler might broaden its user base. Such a tool would allow users to write, test, and debug Maude code directly in their browsers, making learning and experimentation easier.

Another relevant tool would be the Hets (Heterogeneous Tool Set) REST API, available at:

`http://rest.hets.eu/`

Hets is a framework designed for parsing, analyzing, and managing heterogeneous formal specifications across various logical systems. It offers a web-based interface that allows users to access the tool's functionalities remotely, like logic translations, proof management, or theory operations, (Mossakowski et al., 2007).

2.3. UPV's Maude Graphical Tools

The ELP group is a research group on Extensions of Logic Programming based in the Valencian Research Institute for Artificial Intelligence (VRAIN), of the Universitat Politècnica de València. They work on automated software developement, advanced software techniques, and various programming paradigms, including formal methods, semantics, and machine learning. Their research focuses on analyzing, specifying, verifying, debugging, testing, learning, certifying, transforming, and optimizing multi-paradigm programs (Research Group on Extensions of Logic Programming 2024).

This group has developed several tools that offer a graphical environment for Maude that includes advanced visualization and debugging features. They can be found here:

`https://elp.webs.upv.es/soft.html`

These platforms have specific technical purposes that add to the Maude System's original capabilities. They are focused on improving Maude programs through optimization, like **PRESTO**, **iPRESTO** and **Meta-Maudest**; partial evaluation, like **Victoria**; debugging, like **ABETS**, or analysis, like **ATAME** and **Narval**.

Chapter 3

Project Implementation

The purpose of this project is to develop a platform that enables users to interact with the Maude programming language through a web interface. This chapter details the implementation, architecture, structure, and design of the web application.

3.1. Initial Setup and Basic Website Development

The first step in the development process was to create a basic website that would serve as the foundation for the Maude Web Interface. For simplicity and efficiency, the XAMPP tool was used. XAMPP is an open-source cross-platform web server solution that provides a local server environment and includes Apache, MySQL, PHP, and Perl, which are essential for developing and testing web applications (XAMPP, 2024).

- **Apache HTTP Server**: The open-source HTTP server Apache was utilized as the main web server. Apache's comprehensive documentation and strong community support made it an ideal choice for this project. It also integrates seamlessly with PHP, which was used for server-side scripting.

- **MySQL**: The database management system MySQL, managed through XAMPP, was employed to handle the database needs of the application. The database, described in detail below, in section 3.6, stores user data and command history, which is useful for tracking user progress and improving the user experience.

- **PHP**: PHP was chosen for server-side programming due to its ease of integration with HTML and its extensive support for database interactions. PHP scripts handle user inputs, interact with the database, and manage the execution of Maude commands.

The initial version of the website provided a simple interface where users could input Maude commands and receive an output. This basic setup was crucial for validating the core concept before moving on to more complex features.

3.1.1. Apache and XAMPP Configuration

Apache, the open-source HTTP server included in XAMPP, was configured to serve the project's web pages. This setup involved creating a virtual host for the Maude web interface, ensuring that the web pages were accessible via a local URL during development.

3.2. Handling Maude Calls

One of the first challenges in implementing the web interface was integrating the ability to execute Maude commands. Initial attempts to use Common Gateway Interface (CGI) for this purpose were considered. CGI allows web servers to execute external programs and pass the output back to the user. This approach, however, was ultimately discarded due to performance limitations and complexity. (Figure 3.1)

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import maude
import cgi
import cgitb
import warnings
import contextlib

cgitb.enable()

print("Content-type: text/html\n")

maude.init()
warnings.filterwarnings("ignore", category=DeprecationWarning, module="maude")

form = cgi.FieldStorage()
maude_command = form.getvalue("maude_command")
```

Figure 3.1: Code extraction from the first approaches

3.2.1. Transition to PHP and Python

The final solution was to utilize a combination of HTML, PHP, and Python. The implementation involves a PHP script that receives a user input. The PHP

script located in a file called ‘run_script.php‘ is executed (Figure 3.2). This script sanitizes the input and invokes the Python script ‘script.py‘. The Python script handles the core functionality of interacting with Maude, using the Maude Bindings to execute commands, and returns the output to the PHP script, which then displays the results to the user through the interface.

```php
<?php
chdir(__DIR__);

$maude_module = $_POST["maude_module"];
$maude_command = $_POST["maude_command"];

$maude_module = str_replace(array("\r", "\n"), ' ', $maude_module);
$maude_command = str_replace(array("\r", "\n"), ' ', $maude_command);

$escaped_maude_module = escapeshellcmd($maude_module);
$escaped_maude_command = escapeshellcmd($maude_command);

file_put_contents("temp_module.txt", $maude_module);
file_put_contents("temp_command.txt", $maude_command);

$result = shell_exec("C:/Python311/python.exe script.py \"$escaped_maude_module\" \"$escaped_maude_command\"");

unlink("temp_module.txt");
unlink("temp_command.txt");

echo $result;
?>
```

Figure 3.2: run_script.php

Depending on the kind of module input provided and its format, the data is handled differently by the python script, as shown in the Figure 3.3. The command input, which has to follow proper syntax, is addressed using the code displayed in the Figure 3.4.

```javascript
function fillModule(example) {
    document.getElementById("maude_module").value = example;
}

function fileSelected() {
  var fileInput = document.getElementById('file');
  var file = fileInput.files[0];

  if (file) {
    var reader = new FileReader();

    reader.onload = function(e) {
      var fileContent = e.target.result;
      document.getElementById('maude_module').value = fileContent;
    };

    reader.readAsText(file);
  }
}
```

Figure 3.3: fillModule()

```
function executeMaudeCommand(event) {
  event.preventDefault();

  var maudeModule = document.getElementById("maude_module").value;
  var maudeCommand = document.getElementById("maude_command").value;
  document.querySelector('button[type="submit"]').disabled = true;
  runPythonScript(maudeModule, maudeCommand);
}

function runPythonScript(maudeModule, maudeCommand) {

  var xhr = new XMLHttpRequest();

  xhr.onreadystatechange = function() {
    if (xhr.readyState == 4 && xhr.status == 200) {
      document.getElementById("resultContainer").innerHTML = xhr.responseText;
      document.querySelector('button[type="submit"]').disabled = false;
    }
  };

  xhr.open("POST", "run_script.php", true);
  xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
  xhr.send("maude_module=" + encodeURIComponent(maudeModule) + "&maude_command=" + encodeURIComponent(maudeCommand));
}
```

Figure 3.4: executeMaudeCommand()

This explanation provides a general overview of the process involved. This flow and the relationships between the various elements can be seen in the diagram shown in Figure 3.5, which displays a structured representation of the components and how they work together.

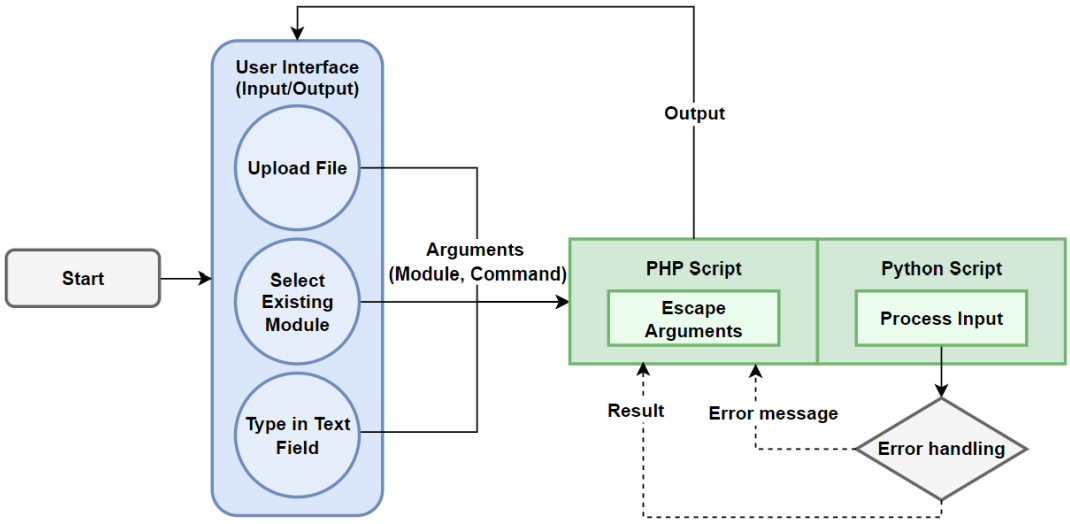

Figure 3.5: Flow diagram

The Maude prompt execution is managed within the Python script. The fist step is to get rid of the unnecessary characters introduced by the system when delegating the data to the script. 3.6. Immediately after we call the first function defined in the bindings. this function stores the module as is. no need to change stuff.

To parse the command introduced by the user the process is more intricate. more steps involved. The function `getCommand()` is called. Its definition can be seen in

```
123  try:
124      maude_module = sys.argv[1].replace("^", "")
125      maude_command = sys.argv[2].replace("^", "")
126
127      maude.input(maude_module)
128
129      command, params = getCommand(maude_command)
130      m,  params = getCommandModule(params)
131
132      p = params.split()
```

Figure 3.6: Maude initialization

the Figure 3.7. Here the input is divided between the term (command parameters) and the command key word, which can be the whole name or an abbreviation of it. This function also checks if the input line follows the proper syntax by raising an exception whenever a prompt does not end in the characteristic "." of the Maude Programming Language.

```
8   def getCommand(c):
9       c2 = c.split()
10
11      if len(c2) < 2 or not c2[-1].endswith('.'):
12          raise Exception("Incorrect sytanx.")
13
14      command = c2[0]
15      term = ' '.join(c2[1:])[:-1]
16      return command, term
```

Figure 3.7: getCommand()

Once the prompt has been divided in command name and term, another function, `getCommandModule()` (Figure 3.8) is in charge of identifying if the term parameters contain a Module definition.

```
19  def getCommandModule(params):
20      if 'in' in params:
21          aux = params.split(':')
22          mod = aux[0][3:].strip()
23          params = aux[1]
24          return maude.getModule(mod), params
25
26      else:
27          return maude.getCurrentModule(), params
```

Figure 3.8: getCommandModule()

Each command is parsed differently and uses the functions defined in the Maude

bindings accordingly. Some have a certain type of parameters and others pues otras. Some use this the `getBrackets()` function shown in the Figure 3.9, others dont.

```
30  def getSquareBrackets(params):
31      a = None
32      b = None
33      if'[' in params:
34          paramsAux = params.split(']')
35
36          if(',' in paramsAux[0]):
37              paramsAux2 = paramsAux[0].split(',')
38              a = paramsAux2[0][1:].strip()
39              b = paramsAux2[1].strip()
40
41          else:
42              a = paramsAux[0][1:]
43
44          params = paramsAux[1]
45          if a == '':
46              a = None
47
48      return params, a, b
```

Figure 3.9: Parsing the parameters

The methods involved in this implementation, developed by Rubén Rubio (2022) in the Maude Bindings, were crucial to the project. These bindings allow Python scripts to interact with Maude, making it possible to execute Maude commands from within the web application. The bindings are available through the Python package index and can be installed using the command:

```
pip install maude
```

Detailed documentation for these bindings is available in the Maude Bindings Documentation (2022).

3.3. Integration of Maude Bindings

The successful integration of Maude bindings into the web interface marked a significant milestone in the project. The bindings allowed Python scripts to invoke Maude functions directly, enabling the platform to execute Maude commands input by the user.

3.3.1. Command Parsing and Execution

Parsing and executing Maude commands introduced additional complexity due to the intricacies of the Maude syntax. The process begins with the elimination of unnecessary characters added by the system when passing data to the script. Next, the system stores the module as-is, without modification.

The command parsing process is more involved. It begins with the getCommand() function, which separates the command keyword from its parameters. This function also ensures that the input adheres to the correct Maude syntax by raising exceptions when necessary, such as when a prompt does not end with a period.

Once the command and its parameters have been isolated, the getCommandModule() function determines whether a module definition is included. This function allows the user to omit the module in the prompt, defaulting to the last module provided through the interface or uploaded as a file.

Each command is parsed differently based on its specific syntax and requirements. For example, some commands utilize the getBrackets() function to handle parameter parsing, while others do not. The detailed parsing and execution logic for each supported command is outlined in the following sections.

3.4. Implemented commands

The platform currently supports a subset of Maude commands, as detailed in the Appendix A of the Maude Manual (Clavel et al. 2007). Each command is parsed and executed according to its specific syntax and requirements. These commands are divided into several sections, and the ones implemented for the Maude Web Interface only cover three of the sections.

- Rewriting Commands: These commands are used to perform operations on Maude terms, such as ‘reduce‘, ‘rewrite‘, ‘frewrite‘, and ‘erewrite‘. For example, the ‘reduce‘ command simplifies a term using the equations and membership axioms in a module.

- Matching Commands: Commands like ‘match‘ and ‘xmatch‘ perform pattern matching on terms. These are crucial for operations that involve determining if a term fits a specific structure defined by the module.

- Search Commands: The ‘search‘ commands perform a breadth-first search for rewrite proofs. It allows users to find terms that match a specific pattern within a given depth or number of steps.

=>1	one step proof
=>+	one or more steps proof
=>*	zero or more steps proof
=>!	only canonical final states, that cannot be further rewritten, are allowed as solutions
=>#	states having multiple distinct successors

Table 3.1: Searchtype values for the search command

Each command has been implemented following Maude's original structure and logic, and the syntax it adheres to is identical, considering every parameter separately as it varies between commands.

The optional parameter {**in *module* :**}, for example, is present in all commands. It references the module to apply to the specified prompt, mentioned above in the getCommandModule() figure and description. If this clause is omitted the current module, last module introduced by the user through the module text input or though the file input, is assumed.

There are other parameters to consider, read by finding the specific characters defining them like square brackets, for the bound, number or depth values, *such that* for conditions, or other certain symbols, like the ones stated in Table 3.1, which identify the the searchtype a search command would apply.

3.5. Input Handling and Execution Limits

The interface allows users to input modules and commands freely, and while the system's parsing mechanism ensures that the input is both readable and executable, certain commands can pose potential issues. For instance, a search command, which iterates based on the input parameters, may run indefinitely if not properly managed. To address this, an iteration limit has been implemented, constraining the number of iterations a command can perform. This ensures that the platform maintains control over execution time, preventing excessive resource consumption and potential system overloads caused by endlessly running commands.

In addition to the iteration limits, to address the possibility of commands getting stuck or taking too long to execute, a timeout mechanism has been introduced. If an unforeseen error occurs or an operation requires more time than expected the timeout ensures that the executions are terminated after a predefined period, safeguarding the platform against performance bottlenecks or crashes.

3.6. Database Integration and Structure

Once the tricky part of command execution was addressed, the focus shifted from developing a basic web page to creating a fully-featured web application. With this goal in mind, a MySQL database was implemented. The database, managed through XAMPP, stores user data and command history. The database schema is designed to support the functionalities of the Maude web interface, allowing for integration of user data and command execution history into the platform (Figure 3.10).

```sql
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
START TRANSACTION;
SET time_zone = "+00:00";

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

DROP TABLE IF EXISTS `Users`;
CREATE TABLE `Users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(20) NOT NULL,
  `password` varchar(70) NOT NULL,
  `name` varchar(100) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

DROP TABLE IF EXISTS `Input`;
CREATE TABLE `Input` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `module` TEXT NULL,
  `command` TEXT NULL,
  `result` TEXT NOT NULL,
  `error` TEXT NULL,
  `sort` varchar(50) NOT NULL,
  `user` int(11) NULL,
  `session_id` TEXT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

ALTER TABLE `Input`
  ADD CONSTRAINT `Input_user` FOREIGN KEY (`user`) REFERENCES `Users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

Figure 3.10: Tables implemented using SQL

- User Data: The database stores user credentials and session information, allowing users to log in and track their command history across sessions.

- Command History: The command history feature enables users to revisit and analyze their previous inputs. This is particularly useful for educational purposes, as it allows users to identify patterns and recurring errors.

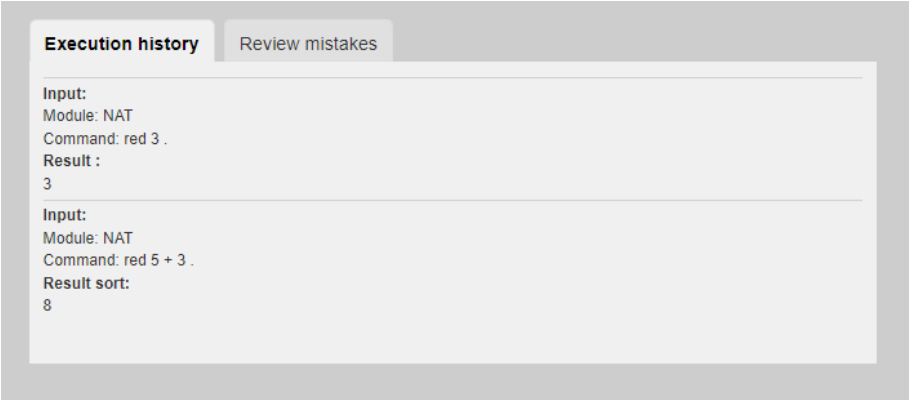

Figure 3.11: Execution history tab

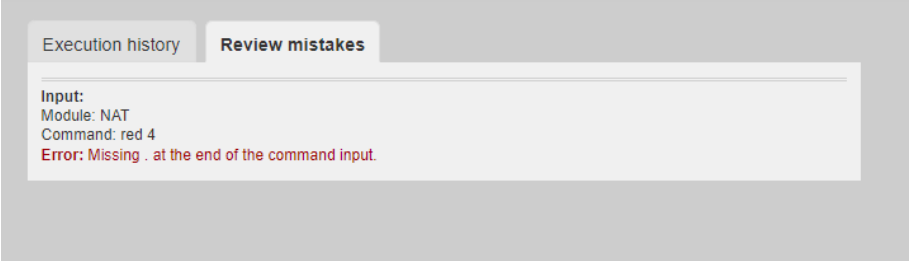

Figure 3.12: Error review tab

To integrate the database, the project had to be restructured. This transformation resulted in a more web application-oriented structure, as the original run_script.php file was replaced with class definitions, forms, and the appropriate supporting files, necessary for database management and design and formatting.

With the database successfully implemented, the focus shifted towards a more didactic approach. Having the stored data could allow as experimental features some concepts like user interactions tracking, monitoring of usage patterns and identification of common errors. This perspective gives this project all the more potential, laying the groundwork for promising development. With this idea in mind, tabs were added to the interface to display the history logs and user error feedback (Figures 3.11 and 3.12).

3.7. Deployment

While the initial development used the local server environment provided by XAMPP, the goal was to deploy the application on a publicly accessible server. Given the application's small scale, several server providers were available without cost. Most platforms, however, either lacked essential features without a paid subscription, did not offer the necessary PHP or Python compatibility, or were simply too limited or not suitable.

PythonAnywhere, an online IDE and web hosting service based on the Python programming language, did not support the PHP code the app had been developed in. Google Cloud Platform, a suite of cloud computing services, also had hosting possibilities with seemingly no compatibility problems with either PHP nor Python, but the database management options were limited. Some other alternatives that were considered were Heroku, Railway, GitHub Pages or InfinityFree, but after thorough testing, all of them were discarded due to different drawbacks.

AWARDSPACE, another free web hosting provider, offered a subdomain which included phpMyAdmin for database management, which would make the transition easier, as it was similar to the development environment offered by XAMPP. However, the python scripts could not be executed, since the shared hosting did not support the Maude Bindings, as it was an external package.

Some of the pages used for deployment testing, although not fully functional, are currently available, like `http://maudeonline.atwebpages.com/` or `http://maudewebinterface.helioho.st/index.php`.

Various additional tools that were employed while testing the different possibilities for deployment included the command line tool Heroku CLI, Composer, a dependency manager for PHP, or Docker, a containerization platform to encapsulate both the application and all its dependencies.

Ultimately the application was successfully deployed in a server managed by the university. This solution offered support for both PHP, database and Python environments, covering all specific demands, ensuring a reliable platform for the project's continued operation.

The web application can be accessed thought the following link:

`https://maude.ucm.es/maudeinterface/`

Chapter 4

Functionality and Usage

This project consists of a web page designed to work in an interactive way with the Maude programming language. The functionality of the Maude Web Interface is designed to replicate the experience of working with Maude in a traditional terminal environment, while adding the convenience of a web-based interface. The application home page, illustrated in the Figure 4.1 allows users to input Maude code immediately, without the requirement of account creation or setup, allowing for immediate access and testing.

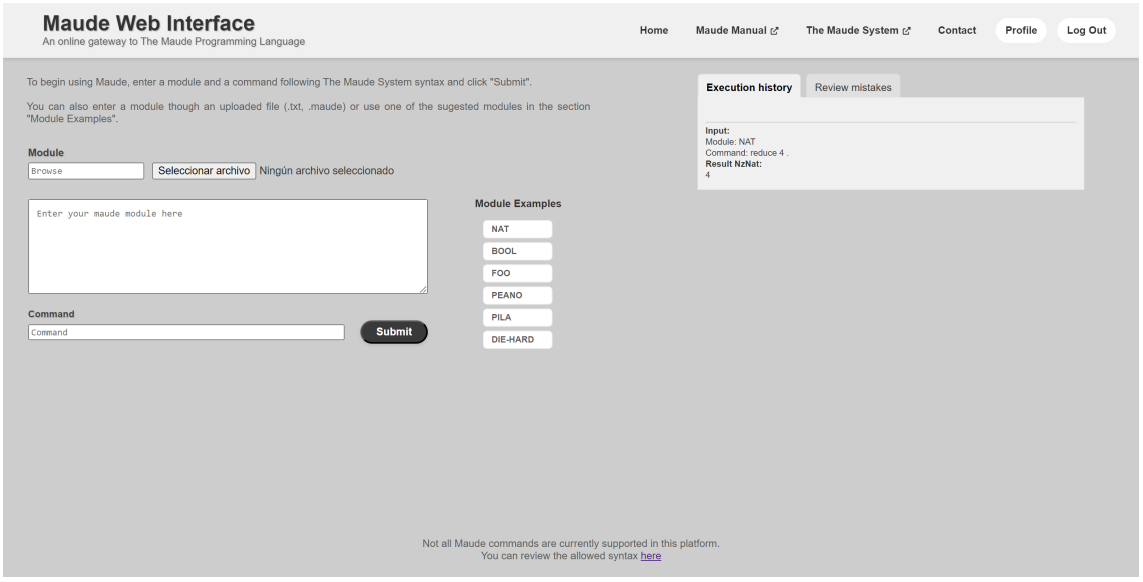

Figure 4.1: Home screen

4.1. User Accounts and Authentication

The Maude Web Interface offers both guest usage and registered accounts. While it's possible to execute commands without logging in, creating an account provides

the added benefit of saving your execution history. This feature is valuable for tracking progress, analyzing results, and debugging issues. Logged-in users can this way continue previous work or review past interactions with the Maude system.

The platform provides an easy login and sign up process, as shown in Figures 4.2 and 4.3. In the event of unsuccessful login or sign up attempts, the system has been designed to displays informative error messages as illustrated in Figures 4.4 and 4.5.

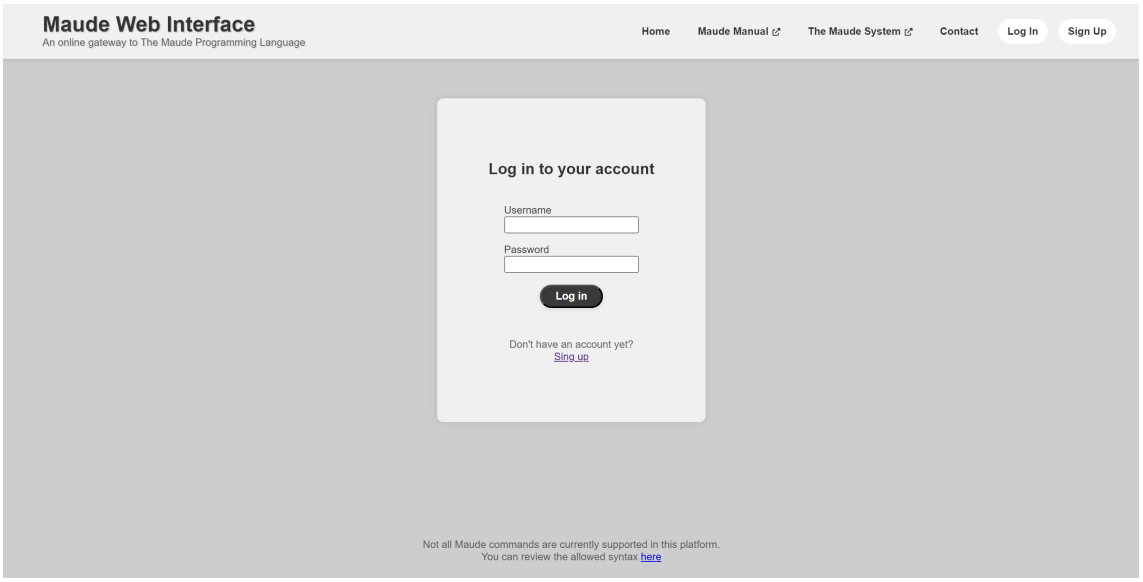

Figure 4.2: Log in screen

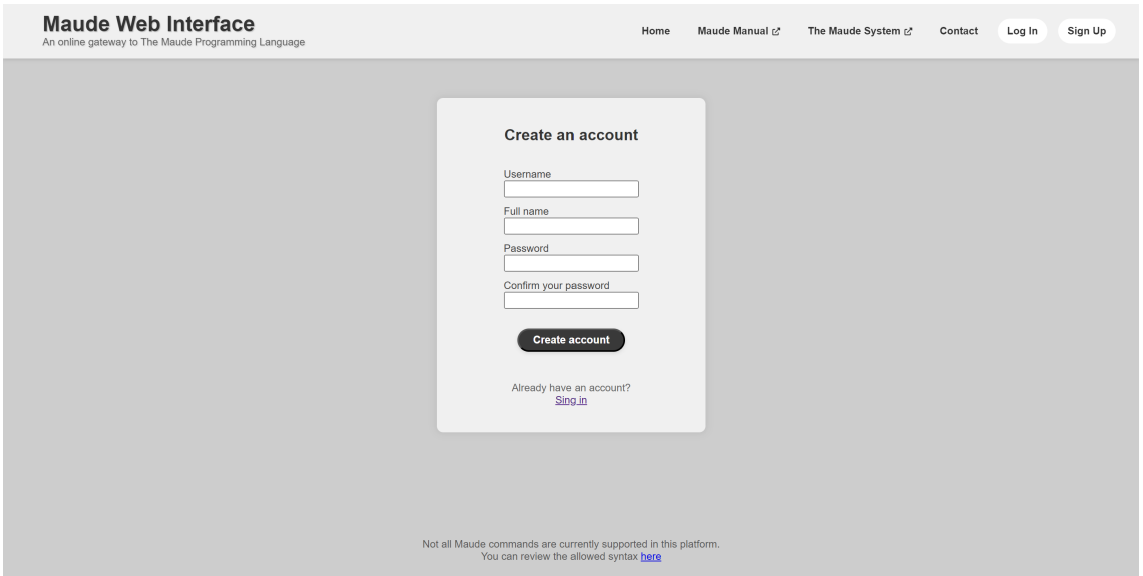

Figure 4.3: Sign up screen

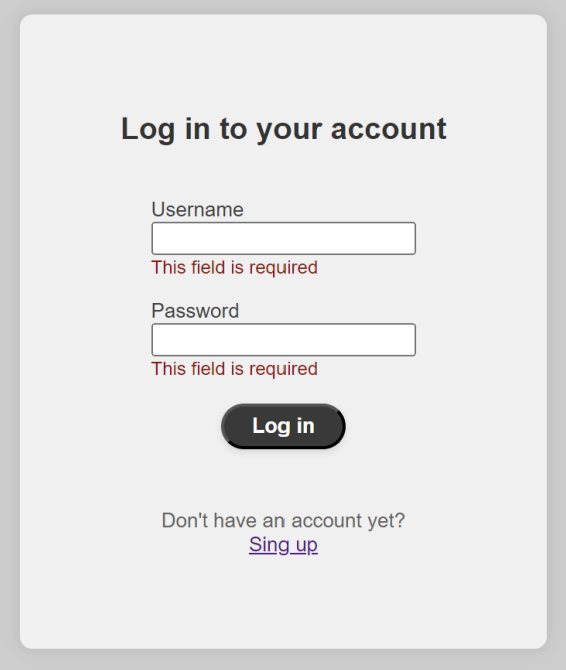

Figure 4.4: Possible login errors

4.2. Supported Syntax Overview

A general summary of the supported syntax can be found in the platform, as shown in Figure 4.6. This overview describes in broad terms the type of prompts that can be used in the platform, noting that certain advanced Maude syntax features may not be fully applicable or supported within the Maude Web Interface. Relevant links referencing extensive documentation of the usage of this programming language is also available for the users.

4.3. Maude Code Input

The primary functionality of the web page is to allow users to input Maude code, which is then processed and executed by the system. The input follows the exact syntax rules defined by Maude, as described in the Maude Manual (Clavel et al. 2007), this allows users to input complex commands without needing to adjust their syntax for the web interface or avoid learning new conventions or methods.

4.3.1. Module Input

The original Maude System's terminal prompt line allows for line-by-line input. This method can be time-consuming, particularity for beginners, who might be more prone to mistakes. As an alternative, this web application offers three possible ways

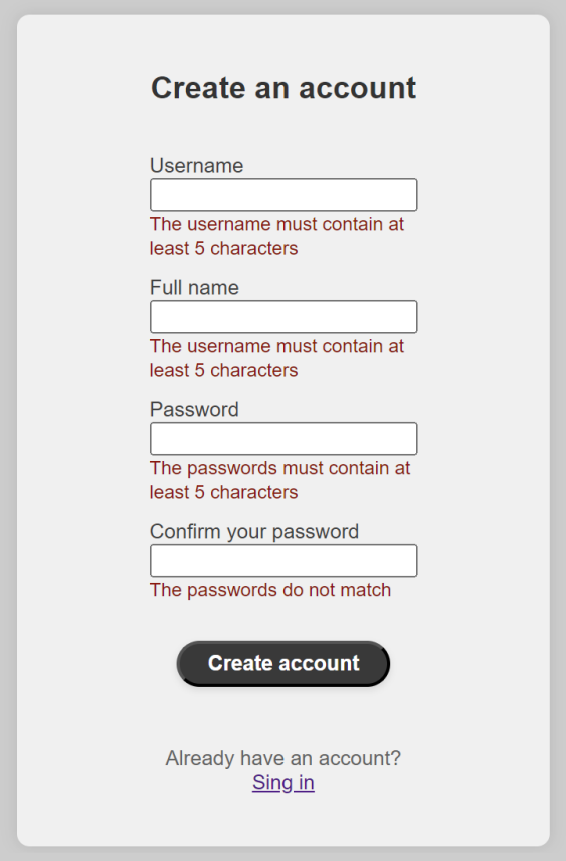

Figure 4.5: Possible sign up errors

to input a module:

- **Text Field**: Users can manually type out a module directly into a text field provided on the page. This allows for real time creation and editing of Maude modules, making it easier to work with compared to the original terminal interface.

- **Predefined Example List**: The web page includes a list of example modules that users can select from. This is particularly useful for beginners or for users who want to quickly test commands without writing out a full module.

- **File Upload**: Users can also upload files containing Maude code. The accepted file extensions are `.txt` and `.maude`, and these files can contain complete modules or any other Maude constructs. This feature allows for the reuse of pre-written code, making the platform convenient for more advanced users or those working with extensive Maude projects.

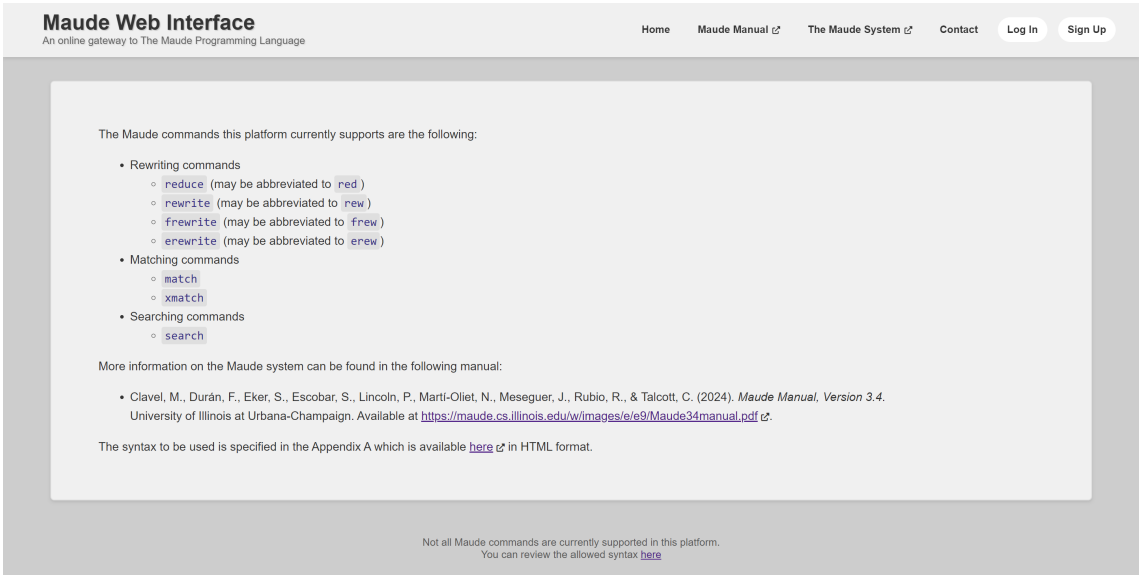

Figure 4.6: Allowed syntax screen

4.3.2. Command Prompt

The command input field functions similarly to the original prompt line in a Linux terminal. This field accepts command with the syntax identical to the required in The Maude System, as described in the manual and official documentation. All parameters are used in the same way. A user can for example input a prompt containing a module (as they would in a Linux terminal: with the `in A :` statement, and that module will be the one used as the basis for the command's logic.

4.3.3. Command and Module Input Specifications

- **Case Sensitivity**: Both the command prompt and the module input fields are case-sensitive, maintaining consistency with the original Maude environment. This ensures that the exact capitalization is preserved, which is crucial for accurate command and module recognition.

- **Line Breaks Handling**: The input fields ignore line breaks. This feature allows users to format their code with line breaks for readability without affecting how the code is processed. The system automatically removes these breaks when parsing the input, ensuring smooth execution.

4.3.4. Error Handling

- **Module Specification**: If a command is entered without specifying a module and no module has been defined through the text field, example list, or file upload, the system will throw an error. This is because the Maude commands

typically require a context (i.e., a module) to execute correctly. Ensuring that a module is always defined before executing a command is critical to avoiding execution errors.

- **Error Messages**: When an error occurs due to the absence of a module or any other syntax-related issue, the system will provide clear and informative error messages. These messages help guide the user towards correcting their input, ensuring a smoother user experience.

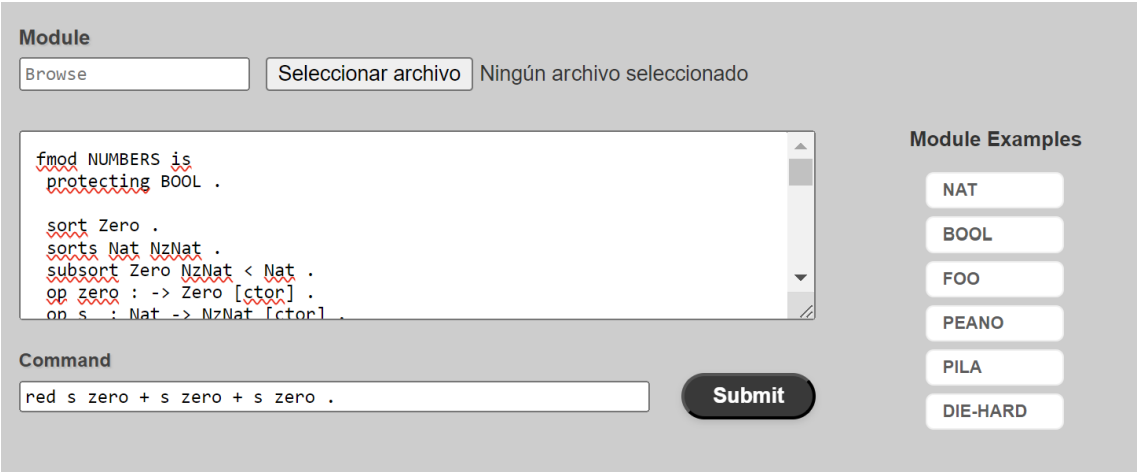

Figure 4.7: Input example

4.4. Application Functionality

The web interface is designed to be intuitive and user-friendly, allowing users to input Maude modules and commands easily. The interface supports both textual input of Maude modules and file uploads. Users can either type their Maude code directly into a text field or upload ‘.txt‘ or ‘.maude‘ files. Once the module is uploaded or inputted, the user can execute various Maude commands.

- User Input Handling: The user inputs a command following the proper Maude syntax. The web interface then parses this input and executes it using the PHP and Python backend as described. Figure 3.2 illustrates the module input process, while Figure 3.3 shows how a command is executed.

- Command Parsing: Parsing the input command is one of the more complex tasks due to Maude's intricate syntax. The input is first cleaned of unnecessary characters introduced by the system. The next step involves identifying

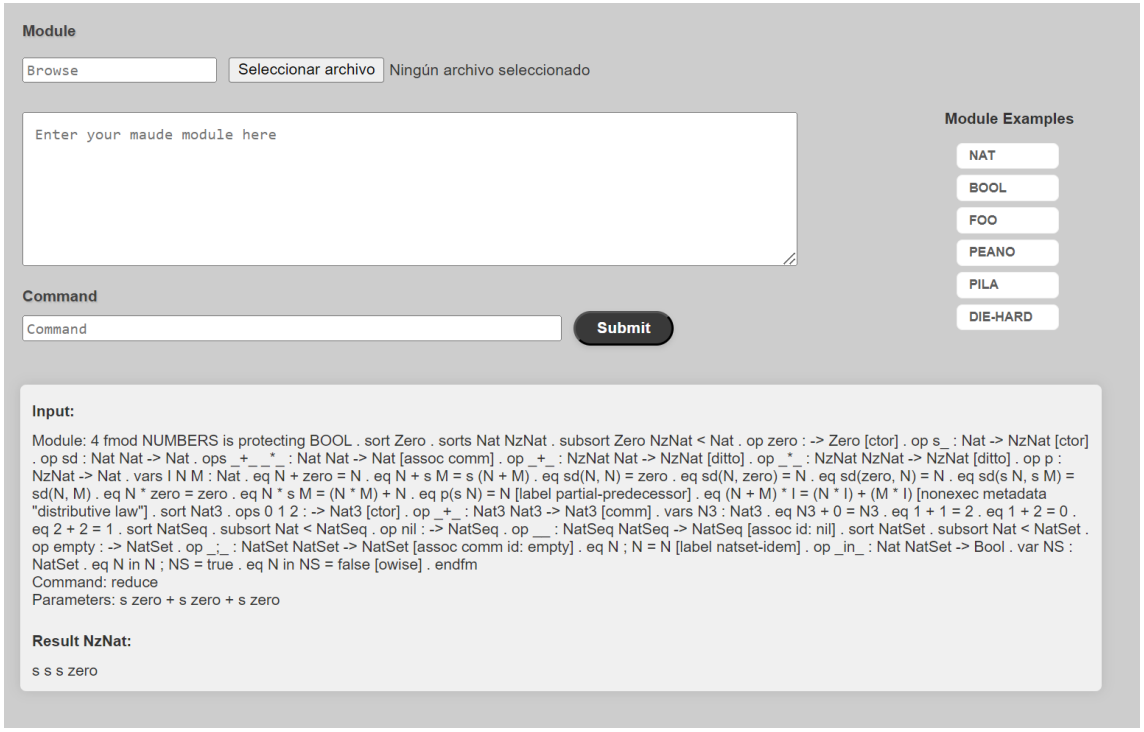

Figure 4.8: Output example

and storing the module using the Maude bindings, without altering the content. For command input, the function ‘getCommand()‘ is called (Figure 3.6), which splits the input into the command keyword and its parameters.

- Module Handling: The function ‘getCommandModule()‘ (Figure 3.7) checks if the input contains a module definition. While specifying a module in the command is optional, if omitted, the current module (from the last input or upload) is used by default.

- Command Execution: Each command is parsed and executed differently depending on its type. Some commands require specific parameters, and others do not. The ‘getBrackets()‘ function (Figure 3.8) is used to handle commands that involve complex structures.

Chapter 5

Comparisons and Critical Analysis

In developing the Maude Web Interface, one of the key objectives was to provide a platform that mirrors the functionality of Maude for Linux while offering the convenience of a web-based environment. Below is a detailed comparison between Maude for Linux and the Maude Web Interface, as well as a comparison with other similar platforms for different languages. This section also includes a critical analysis of the project, highlighting its strengths, limitations, and areas for improvement.

5.1. Comparison with Maude for Linux

5.1.1. Features

- **Command Execution**: The Maude Web Interface faithfully replicates the command execution capabilities of Maude for Linux. Users can input commands in a similar syntax, and the platform supports a wide range of Maude commands, including `reduce`, `rewrite`, `frewrite`, and more. However, while the Linux version allows for more complex interactions and scripting, the web interface is somewhat limited in handling multi-step, interactive sessions that are common in the Linux environment. Execution, results and other available information that can be found in the Maude System's results cannot be found in this platform.

- **Module Management**: Both platforms allow users to define and manipulate modules. However, the Linux version offers more flexibility in managing large projects with multiple interconnected modules. The web interface simplifies module management which can come at the cost of reduced flexibility in larger, more complex projects.

5.1.2. Performance

- **Execution Speed**: The performance of the Maude Web Interface is generally slower than Maude for Linux. This is primarily due to the overhead introduced by the web environment, including the need to transmit data between the client and server, and the additional processing required to parse and execute commands via a web interface. While the web interface is suitable for small to medium-sized tasks, users working on large-scale Maude models may experience significant delays compared to the native Linux version.

- **Resource Utilization**: Maude for Linux is more efficient in terms of resource usage. The web interface, being built on top of a stack that includes Apache, PHP, and Python, inherently consumes more system resources. This difference could be particularly noticeable in environments with limited computational power or network bandwidth.

5.1.3. Usability

- **User Interface**: One of the main innovations of the Maude Web Interface is its user-friendly graphical interface, which makes Maude more accessible to users unfamiliar with command-line environments. The web interface provides a straightforward way to input code, manage modules, and view output, reducing the learning curve for new users. In contrast, Maude for Linux requires familiarity with terminal commands and scripting, which can be a barrier for some users.

- **Error Handling**: The web interface provides more user-friendly error messages and guidance compared to the often cryptic errors encountered in the Linux version. This makes troubleshooting easier for beginners, although advanced users might find the web interface's error handling less detailed and less informative in diagnosing complex issues.

5.2. Comparison with Other Web-Based Platforms

5.2.1. Similar Platforms

- **Online IDEs**: Platforms like **Replit** and **JSFiddle** offer similar web-based environments for coding in various languages. These platforms typically support real-time code execution, version control, and collaboration features.

- **Interactive Theorem Provers**: The Maude Web Interface shares some similarities with web interfaces for interactive theorem provers. These platforms

are designed for formal verification and proof development, much like Maude.

- **UPV's Maude Graphical Tools**: While some of the tools developed for Maude by the Research Group on Extensions of Logic Programming (section 2.3) share a very similar interface, they offer a different concept.

  Most of these platforms provide a Maude input with predefined input suggestions, similar to the Maude Web Interface does, but they delve into more technical aspects. They all have a specialized features that complement the original Maude System, like debugging, automated testing, optimization, verification, analysis or model checking, all of this applied to more complex Maude programs. This is a very useful approach, but one that would reduce beginners from the target audience, since it requires a certain level of understanding on the system.

- **Hets (The Heterogeneous Tool Set)**: Although this platform does address a similar topic, Hets focuses on providing a framework for managing heterogeneous formal specifications rather than executing specific commands. It does include Maude support and offer a web-based interface, but it specializes in formal methods, integrating and translating between different logical systems.

5.2.2. Innovations and Limitations

- **Innovations**: The main innovation of the Maude Web Interface is its ability to bring Maude to users who are not comfortable with a command-line interface. This broadens the accessibility of Maude, potentially attracting a wider audience. Additionally, the interface simplifies module management and offers a more guided experience with its error messages and input options.

- **Limitations**: Despite these innovations, the Maude Web Interface falls short in several areas:

  1. **Limited Advanced Features**: Unlike other platforms, the Maude Web Interface lacks features such as collaborative editing, extensive version control, and live debugging. This limits its usefulness for more complex or collaborative projects.

  2. **Performance Constraints**: As mentioned earlier, the web interface introduces performance constraints that can be a significant drawback for users dealing with large models or needing real-time responsiveness.

  3. **Scalability Issues**: The web interface is less suited for handling large-scale Maude projects due to its simplified module management and the

limitations of the underlying web technologies.

For an overview of the similarities and differences between the mentioned platforms, the Table 5.1 provides a comparative analysis. It outlines some of the most relevant features and characteristics of several online tools, The Maude System, and the Maude Web Interface for contrast.

Feature	**Maude System**	**Replit**	**JSFiddle**	**PRESTO**	**ABETS**	**Narval**	**Hets**	**Maude Web Interface**
Languaje	Maude	Multiple languages (no Maude)	Multiple languages (no Maude)	Maude programs	Maude programs	Maude programs	Multiple languages	Maude commands
Primary Functionality	Formal verification, rewriting logic	Online IDE	Online IDE	Program optimization	Automated error testing	Static analysis	Parsing, static analysis, proof management	Web-based Maude interface
Error Feedback	Technical error messages	Basic error messages, language-specific	Basic error messages, language-specific	Focused on optimization issues	Detailed debugging feedback	Detailed analysis feedback	Comprehensive, multi-logic error feedback	User-friendly, but less detailed for complex issues
Target Users	Advanced users, formal methods experts	Coders, beginners to experts	Web developers, beginners to experts	Advanced Maude users	Advanced Maude users	Advanced Maude users	Experts in formal methods and logic	Maude users, beginners to experts
History	None (without custom implementation)	For logged users	For logged users	None	None	None	Integrated through development graphs	For logged users
Innovations	Rewriting logic, formal verification, etc	Multi-language support, cloud-based	Real-time web development environment	Advanced Maude optimization	Advanced Maude debugging	Advanced Maude analysis	Logic handling, logic translations	Web accessibility Maude commands

Table 5.1: Comparison of the Maude Web Interface with similar systems

5.3. Critical Analysis and Obstacles

5.3.1. Strengths

- **Accessibility**: The primary strength of the Maude Web Interface is its accessibility. By providing a graphical interface, it lowers the barrier to entry for new users and those unfamiliar with command-line tools.

- **User Experience**: The interface is designed with ease of use in mind, offering intuitive input fields, error handling, and module management options

that make it easier for users to work with Maude.

5.3.2. Weaknesses

- **Performance**: One of the most significant drawbacks of the web interface is its performance, particularly when compared to the Linux version. The additional layers of web technologies introduce latency and can make the interface less responsive.

- **Feature Set**: While the web interface is suitable for basic Maude usage, it lacks many of the advanced features that power users might expect. This limits its appeal to more experienced users who require the full capabilities of the Maude system.

- **Scalability**: The platform is not well-suited for handling very large or complex Maude projects, which can be a limitation for users working in more advanced settings.

5.4. Obstacles Faced During Development

- **Integration Challenges**: One of the primary challenges in developing the Maude Web Interface was integrating Maude's command-line capabilities with a web-based environment. This required extensive study of different approaches to try to connect the user interface and the underlying Maude engine. Ensuring that commands executed correctly while maintaining a seamless user experience was a significant technical hurdle.

- **Error Handling**: Implementing robust error handling that was both informative and user-friendly was another challenge. The web interface needed to provide enough information to help users correct their mistakes without overwhelming them with technical details. This was particularly complex as I lacked experience with the Maude programming language. Furthermore, I anticipated that users' expertise levels would vary significantly, making it difficult to determine the optimal level of feedback detail.

- **Resource Management**: Balancing resource utilization to ensure that the interface remained responsive while executing potentially resource-intensive Maude commands was a difficult task. This was particularly challenging given the constraints of running within a web environment, which does not have direct access to system resources in the same way that a native application does, and often lead to performance bottlenecks and unexpected behavior. This

constraint often required manual intervention, such as re configuring settings, terminating infinite loops, or waiting for lengthy execution times, hindering the overall development process.

- **Deployment**: An additional challenge emerged from weak planning for the project's deployment on a public server. This aspect, which should have been addressed in the early stages of the project, was overlooked, leading to unexpected issues related to server compatibility and configuration. The failure to anticipate these factors caused significant delays and complications during the final stages of the development, as resolving these deployment issues required unplanned adjustments and troubleshooting.

Chapter 6

Conclusions and Future Work

This project has been a learning journey in specification languages and web application development. Although versed in application development, The Maude system was completely unknown to me when beginning this project. While studying its characteristics, i found it hard to approach, which gave me insight in what could be useful as an absolute beginner wanting a first contact with the language, and finding several barriers. The result was the Maude Web Interface, which aims to bring the capabilities of Maude to a broader audience through its accessible, web-based platform.

6.1. Conclusions

The writing of this paper has been enlightening as to how little attention the field of declarative programming has gotten, and how useful it would be to further research this field. When I began with this subject my knowledge of the Maude Language were nonexistent and it has been challenging to make this all work. The complexities of the syntax have given me challenge after challenge, trying to reinterpret each command utilizing the python bindings.

One of the most significant challenges encountered was dealing with the intricacies of Maude's syntax and the need to accurately reinterpret each command using Python bindings. This task proved to be more difficult than anticipated, primarily due to the complexities inherent in Maude's language structure and my lack of knowledge on the subject.

Another obstacle was caused by failure to anticipate the potential complications when deploying the project to a public server. This was an oversight in the planning process, and it was an aspect that should have been considered from the beginning of the project. Not taking into account compatibility issues and related factors caused major setbacks in the final stages of the development process.

Nevertheless, the project has met the primary objectives that were originally planned. An online interface has been successfully developed, albeit currently hosted on a local server. This interface supports a range of Maude commands and module operations, providing a useful tool for beginners in declarative programming who might not have access to the full Maude environment.

However, there are areas where the project falls short of its potential. It has notable limitations in terms of performance and security measures, crucial for the intended use of this interface. The implementation of sandbox environments or similar security mechanisms has been postponed, which could limit the interface's broader applicability. Additionally, while the interface does imitate the behavior of the Maude program, it lacks some of the more advanced functionalities and performance efficiencies of the native system.

Overall, while the project has succeeded in creating a functional and educational tool, it remains a work in progress, with significant room for improvement, particularly in terms of security, performance, and feature completeness.

6.2. Future Work

To further improve this project, there are many aspects that could be explored to make the platform more practical.

- Expansion of Supported Maude Commands: A critical next step would be to expand the number of Maude commands supported by the web interface. Currently, only a subset of commands, primarily those that do not affect the underlying system, have been implemented. To make the interface more comprehensive and closer in functionality to the original Maude system, it is essential to develop support for additional commands, particularly those that interact with the system at a deeper level.

- Enhanced Security Measures: With the potential expansion of command support, especially those that interact with the system, it is crucial to implement robust security measures. Future work should focus on developing sandbox environments or other protective mechanisms that can safeguard the system from potential breaches or failures. This will be particularly important if the interface is to be made publicly accessible or used in educational settings.

- Server Maintenance and Sustainability: For the Maude Web Interface to be a viable tool in the long term, proper server maintenance and sustainability are necessary. Currently, the project is hosted on a temporary setup, which limits its accessibility and scalability. Future efforts should include setting

up a dedicated server with appropriate maintenance protocols to ensure the interface's stability and availability. This would involve securing resources for ongoing server management, as well as potentially exploring cloud-based hosting solutions to improve accessibility and performance.

- Performance Optimization: Another area for future work is optimizing the performance of the web interface. While the current implementation is functional, it lags behind the native Maude system in terms of speed and efficiency. Investigating ways to streamline the command execution process and reduce the overhead introduced by the web environment could significantly improve user experience, particularly for more complex or resource-intensive tasks.

- User Interface and Experience Enhancements: Improving the user interface and overall user experience could also be a focus of future development. While the current interface is accessible and user-friendly, there is room for improvement in areas such as real-time feedback, syntax highlighting, and error reporting, as explored in the previous chapter. Enhancing these aspects could make the interface more intuitive and valuable, particularly for beginners in declarative programming.

- Exploring Integration with Other Tools and Platforms: Finally, future work could explore the integration of the Maude Web Interface with other tools and platforms, such as online IDEs or collaborative coding environments. This could open up new possibilities for using Maude in educational settings or collaborative projects, further extending the reach and impact of declarative programming.

In conclusion, while the Maude Web Interface is a promising step forward in making Maude more accessible, there is substantial work to be done to fully realize its potential. This journey could continue by delving deeper into areas that have been overlooked but are fundamental in this the project.

Bibliography

ABETS. Abets. https://safe-tools.dsic.upv.es/abets/, 2024.

Apache HTTP Server. Apache http server project. https://httpd.apache.org/, 2024. Accessed on September 6, 2024.

ATAME. Atame. https://safe-tools.dsic.upv.es/atame/, 2024. Accessed on September 2024.

AWARDSPACE. Awardspace. https://www.awardspace.com/, 2024.

Beaulieu, A. *Learning SQL*. O'Reilly Media, Sebastopol, CA, 2nd edición, 2009.

Clavel, M., Durán, F., Eker, S., Meseguer, J. y Talcott, C. L. The maude system. En *Rewriting Techniques and Applications*. Springer, 1999.

Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J. y Talcott, C. *All About Maude - A High-Performance Logical Framework: How to Specify, Program, and Verify Systems in Rewriting Logic*, vol. 4350 de *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 2007. ISBN 978-3-540-71940-3.

Composer. Composer documentation: Basic usage. https://getcomposer.org/doc/01-basic-usage.md, 2024. Accessed on September 2024.

Docker. Docker. https://www.docker.com/, 2024.

Flanagan, D. *JavaScript: The Definitive Guide*. O'Reilly Media, Sebastopol, CA, 7th edición, 2020.

GitHub. Github. https://github.com/, 2024.

GitHub Desktop. Github desktop. https://github.com/apps/desktop, 2024.

GitHub Packages. Github packages. https://github.com/features/packages, 2024. Accessed on September, 2024.

GitHub Pages. Github pages. https://pages.github.com/, 2024. Accessed on September, 2024.

Goguen, J. A. y Malcolm, G. *Algebraic Semantics of Imperative Programs*. MIT Press, Cambridge, MA, 1996. ISBN 978-0-262-07167-3.

Google Cloud Platform. Google cloud platform. `https://cloud.google.com/`, 2024. Accessed on September, 2024.

Heroku. Heroku. `https://signup.heroku.com/`, 2024. Accessed on September, 2024.

Heroku CLI. Heroku cli documentation. `https://devcenter.heroku.com/articles/heroku-cli`, 2024. Accessed on September 2024.

InfinityFree. Infinityfree. `https://www.infinityfree.com/`, 2024. Accessed on September, 2024.

iPRESTO. ipresto. `https://safe-tools.dsic.upv.es/iPresto/`, 2024. Accessed on September 2024.

JDoodle. Jdoodle. `https://www.jdoodle.com/`, 2024. Accessed on September 2024.

JSFiddle. Jsfiddle. `https://jsfiddle.net/`, 2024. Accessed on September 2024.

Matthes, E. *Python Crash Course: A Hands-On, Project-Based Introduction to Programming*. No Starch Press, 2016.

Maude Bindings Documentation. Maude bindings documentation. `https://fadoss.github.io/maude-bindings/index.html`, 2022. Accessed on May 2024.

Maude Installation Guidelines. Installation guidelines. `https://maude.cs.illinois.edu/w/index.php/Installation_guidelines`, 2022. Accessed on May 2024.

Meseguer, J. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, vol. 96(1), páginas 73–155, 1992.

Meta-Maudest. Meta-maudest. `https://elp.webs.upv.es/soft/maudest/index.php`, 2024. Accessed on September 2024.

Meyer, E. A. *CSS: The Definitive Guide*. O'Reilly Media, Sebastopol, CA, 4th edición, 2018.

Microsoft Visual Studio Code. Microsoft visual studio code. `https://code.visualstudio.com/`, 2024.

Mossakowski, T., Maeder, C. y Lüttich, K. *Hets: The Heterogeneous Tool Set*, vol. 259 de *CEUR Workshop Proceedings*. Springer, 2007. ISBN 978-3-540-71208-4.

MySQL. Mysql. `https://dev.mysql.com/doc/refman/8.4/en/what-is-mysql.html`, 2024. Accessed on September, 2024.

NARVAL. Narval. https://safe-tools.dsic.upv.es/narval/, 2024. Accessed on September 2024.

PRESTO. Presto. https://safe-tools.dsic.upv.es/presto/, 2024. Accessed on September 2024.

PYTHONANYWHERE. Pythonanywhere. https://www.pythonanywhere.com/, 2024. Accessed on September, 2024.

RAILWAY. Railway. https://railway.app/, 2024. Accessed on September, 2024.

REPLIT. Replit. https://replit.com/, 2024. Accessed on September 2024.

RESEARCH GROUP ON EXTENSIONS OF LOGIC PROGRAMMING. Polytechnic university of valencia. https://elp.webs.upv.es/index.html, 2024. Accessed on September 2024.

ROBBINS, J. N. *Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics*. O'Reilly Media, Sebastopol, CA, 5th edición, 2018.

RUBÉN RUBIO. Language bindings for maude. cff-version: 1.2.0, message: If you use this software, please cite it as below., 2022. Version 1.2.0. Retrieved: 2022-08-10.

THE MAUDE SYSTEM. The maude system. https://maude.cs.illinois.edu/w/index.php/The_Maude_System, 2024. Accessed on May 2024.

VICTORIA. Victoria. https://safe-tools.dsic.upv.es/victoria/, 2024. Accessed on September 2024.

WELLING, L. y THOMSON, L. *PHP and MySQL Web Development*. Addison-Wesley, Boston, 5th edición, 2017.

WINDOWS SUBSYSTEM FOR LINUX. Windows subsystem for linux. https://learn.microsoft.com/en-us/windows/wsl/about, 2023. Accessed on September, 2024.

XAMPP. Xampp: The cross-platform web server solution. https://www.apachefriends.org/index.html, 2024. Accessed on August 2024.