Project Specifications

Tashrique Ahmed, Elyes Laalai

0.1 Introduction

Halal Gamble: The Stock Simulation Game Halal Gamble is a domain-specific language (DSL) that emerges as a strategic tool designed to unravel the complexities of stock market investing through simulation with historical data. It is expressly engineered to serve as a user-friendly and engaging platform, targeting novice investors seeking a risk-free environment for learning and experimenting with stock investments. As old tools grew more complex, beginner-friendly platforms and programs have turned vary rare which led to the idea of this programming language. This project targets people with different trading experiences and, most importantly, is open to all.

The Halal Gamble provides the player with historical stock prices from 2010-2020. The player starts in 2015 and reviews graphs and data about gold, silver, and Tesla, and decides whether they decided to pursue a stock purchase or sale. Afterwards, this program provides them with the new prices in 2016, and the same process continues up to 2020. The player has both the option to buy or sell stocks. The player might also choose to move through a a year without starting a transaction. At the end of the game, the player is informed about their gains or losses. The results are a useful metric to estimate how well the trader would have done a few years in the past with real capital.

0.2 Design Principles

Halal Gamble places a paramount emphasis on simplicity, clarity, and user engagement. Its design incorporates a straightforward syntax and semantics for easy comprehension, coupled with efficient data structures and memory management to optimize performance. The language offers flexibility in stock and date selections, ensuring robust type-checking and error-handling for user safety. Committed to user accessibility, Halal Gamble provides clear documentation and fosters a supportive community, enhancing the overall user experience. Tailored for practical application, the DSL adeptly handles historical stock market data, featuring an intuitive, natural-language-like syntax suitable for all skill levels. Beyond syntax, it includes investment-specific functionalities and visualizations crucial for demystifying complex financial concepts, while allowing for flexible and customizable portfolio management. In essence, Halal Gamble seamlessly combines technical sophistication with user-friendly features, catering to a diverse audience and facilitating a deep understanding of stock market dynamics. This design make this language easy and educational.

0.3 Examples

Sample Program 1

```
dotnet run
start

/* Open a PDF file containing stock market data and news from 2010 to 2015*/
InitialCapital(100)
Buy(GOLD, 25)
Next

/*Open a PDF file containing stock market data and news from 2015 to 2016*/
Sell(GOLD, 25)
Buy(SLVR, 70)
Next

/*Open a PDF file containing stock market data and news from 2016 to 2017*/
Sell(SLVR, 70)
Next

/*Open a PDF file containing stock market data and news from 2017 to 2018*/
```

```
Buy(SLVR, 50)
Next

/*Open a PDF file containing stock market data and news from 2018 to 2019*/
Sell(SLVR,50)
Buy(GOLD,50)
Next

/*Open a PDF file containing stock market data and news from 2019 to 2020*/
Buy(GOLD, 50)
Graph(BarGraph)
Exit
```

Sample Program 2

```
dotnet run
start

/* Open a PDF file containing stock market data and news from 2010 to 2015*/
InitialCapital(100)
Buy(TSLA, 25)
Next

/*Open a PDF file containing stock market data and news from 2015 to 2016*/
Sell(TSLA, 25)
Next

/*Open a PDF file containing stock market data and news from 2016 to 2017*/
Sell(TSLA, 100)
Next

/*Open a PDF file containing stock market data and news from 2017 to 2018*/
Sell(TSLA, 100)
Next

/*Open a PDF file containing stock market data and news from 2018 to 2019*/
Buy(SLVR, 100)
Next

/*Open a PDF file containing stock market data and news from 2019 to 2020*/
Sell(SLVR, 100)
Graph(TimeSeries)
Result(PortfolioStatement)
Exit
```

Sample Program 3

```
dotnet run

/* Open a PDF file containing stock market data and news from 2010 to 2015 */
InitialCapital(150)
Buy(GOLD, 50)
Buy(TSLA, 50)
Next
```

```
/* Open a PDF file containing stock market data and news from 2015 to 2016 */
Sell(GOLD, 50)
Sell(TSLA, 50)
Next

/* Open a PDF file containing stock market data and news from 2016 to 2017 */
Buy(SLVER, 50)
Buy(TSLA, 50)
Next

/* Open a PDF file containing stock market data and news from 2017 to 2018 */
Next

/* Open a PDF file containing stock market data and news from 2018 to 2019 */

Next

/* Open a PDF file containing stock market data and news from 2019 to 2020 */
Sell(SLVER, 50)
Sell(TSLA, 50)
Graph(TimeSeries)
Result(PortfolioStatement)
Exit
```

0.4 Language Concepts

The core concepts in this domain-specific language (DSL) are centered around stock market investment operations. Users need to understand basic investment terms and actions, as well as how to represent these in the DSL's syntax.

Primitives:

- **Stock Names:** These are primitives in the DSL, representing distinct investment assets like GOLD or SILVER.
- **Numerical Values:** These include figures like initial capital and investment amounts. Action Keywords: Words like 'Buy' and 'Sell' are primitives that represent fundamental operations in the language.

Combining Forms:

- **Sequential Commands:** The combination of various commands (like buying and selling stocks, setting initial capital) in a sequence to form a coherent investment strategy.
- **Investment Operations:** Constructs that combine primitives like stock names and numerical values to define a particular operation, e.g., Buy(GOLD, 50).
- **Control Structures:** Commands like Next or Exit, which control the flow of the simulation, acting like control flow statements in conventional programming languages.
- **Data Visualization Commands:** These combine various data points (such as percentages of investment in different assets) into a cohesive graphical representation. They are similar to functions that take multiple inputs and produce a composite output.
- **Portfolio Statements:** Combining different elements of the portfolio (like different stocks and their respective quantities) to generate a comprehensive statement or analysis.

0.5 Formal Syntax

Detailed Syntax of Halal Gamble:

- **Initial Capital:** Defined using ‘InitialCapital(amount)‘. This command sets the starting budget for the simulation.
  - Example: ‘InitialCapital(100)‘ allocates a budget of 100 dollars.
- **Stock Transactions:** Executed with ‘Buy(stock, amount)‘ or ‘Sell(stock, amount)‘, allowing users to purchase or sell stocks.
  - Example: ‘Sell(GOLD, 20)‘ indicates selling 20 dollars of GOLD stock.
- **Control Commands:** ‘Next‘ advances the simulation to the next timeframe, while ‘Exit‘ terminates the simulation.
- **Data Visualization:** Utilize ‘Graph(Type)‘ to display investment data graphically. Supported types include BarGraph, PieChart, and TimeSeries.
  - Example: ‘Graph(TimeSeries)‘ creates a time series plot of the investment data.
- **Result Reporting:** ‘Result(Type)‘ provides a summary or detailed report of the investment performance. Types can include PortfolioStatement, TransactionLog, etc.
  - Example: ‘Result(PortfolioStatement)‘ generates a detailed statement of the portfolio.

0.6 Semantics

Primitive Kinds of Values:

- **Integers:** These are used for specifying quantities like the number of stocks to buy or sell, and for defining initial capital.
- **Strings:** Used for stock names (e.g., GOLD, SILVER) and for potential date inputs.
- **Graphs:** To present visual data, rendered using python matplotlib

Actions and Compositional Elements:

- **Buy/Sell Operations:** These actions combine stock names (String) and quantities (Integer) to execute transactions.
- **Next/Exit Controls:** ’Next’ progresses the simulation, while ’Exit’ terminates it. These control the flow of the program.
- **Visualization Commands:** Commands like Graph(BarGraph) combine historical data to generate visual representations.

The AST (Abstract Syntax Tree) for this DSL will have several key components represented as algebraic data types:

- **TransactionNode:** Represents stock operations, with types like BuyNode and SellNode, each holding stock name (String) and quantity (Integer).
- **CapitalNode:** Represents initial capital setting, holding the capital amount (Integer).
- **ControlNode:** Represents control commands like Next and Exit.
- **VisualizationNode:** For graphical commands, with types like BarGraphNode and TimeSeriesNode.

The AST elements fit together as follows:
The *root* node represents the entire program. Child nodes of the root are operation nodes, capital nodes, control nodes, and visualization nodes, reflecting the sequence of commands in the program. For example, the AST for Sample Program 1 might look like:

```
Root
|-- InitialCapitalNode(100)
|-- TransactionNode
|   |-- Buy
|   |   |-- Stock: GOLD
|   |   |-- Quantity: 25
|-- ControlNode(Next)
|-- TransactionNode
|   |-- Sell
|   |   |-- Stock: GOLD
|   |   |-- Quantity: 25
|   ...
|-- VisualizationNode
|   |-- Type: BarGraph
|-- ControlNode(Exit)
```

Program Evaluation (Input) Programs in this DSL read textual input commands, which include stock names (e.g. GOLD, SILVER), numerical values (e.g. quantities for buying or selling, initial capital), and control instructions (e.g. Next, Exit). Each line of input corresponds to an operation or control command that affects the simulation of the stock market investment.

Program Evaluaiton (Output) The output of evaluating a program in this DSL includes:
A comprehensive summary of the investment portfolio's performance, detailing gains or losses. Graphical representations of the investment portfolio, showing the distribution and performance of different stocks over time.

Post-Order Traversal for Evaluation: In the post-order traversal of an AST:
Each node represents an operation or control command. The traversal order ensures that each operation's effect on the portfolio is calculated before moving on to subsequent operations.

0.7 Remaining Work

At this stage, we have added the three main components of our programming language to our repository. First, we coded Combinator.fs, AST.fs, Combinator.fs, Evaluator.fs, Library.fs, Math.fs, Parser.fs, Program.fs, constructResult.py. Second, we included csv data files of three stock prices: gold, silver, and Tesla from 2010 to 2020. Last, we added PDFs which include graphics of the evolution of prices over time. We tested these components separately so that it's easier later on to spot problems with our code and be able to fix them. The next, and probably ultimate, step of the process is to connect all these pieces and make sure that our project compiles and runs. We tested all the different pieces separately and made sure they worked. Now, we have to combine them all, and see how they interact and produce (or not) the correct output that we are expecting. Doing so, we can also pick up on some inefficiencies that we can address. Currently, the data types we started with, which we described thoroughly in this project specification, seem enough to achieve the goal of our programming language, so we are not considering adding new ones. The next steps of this project are outlined as follows:

- ☐ Finish debugging the code files
- ☐ Make sure the output is correct
- ☐ Optimize code (if possible)

- ☐ Add documentation (when needed)
- ☐ Review submission, and submit!