Options Trading and Plotting DSL

This document outlines the design of our DSL regarding option trading. It describes how we will handle modeling and visualizing option trading strategies. This DSL will also be able to detect and name certain strategies that the user inputs, allowing it to function as a teaching tool as well.

Purpose and Concepts

This DSL will be used to allow traders to easily test and examine option strategies and portfolios that they come up with, as well as plot them in an easy-to-understand way. Additionally, this DSL will detect option strategies (if they are pre-existing) and name them accordingly, allowing users to experiment and learn existing strategies. This DSL can be used by option traders or financial students.

Option

A contract giving the right to buy (**call**) or sell (**put**) at a fixed price.

Mapped in:

```
racket (define-option <option-name> #:type <call | put> #:action <buy | sell> #:strike <Number> #:expiration <Number> #:premium <Number> #:quantity <Number>)
```

These variables represent:

- **Type (`#:type <call | put>`)** → Whether the option is a **call** or **put**.
- **Action (`#:action <buy | sell>`)** → Whether the option is being **bought or sold**.
- **Strike Price (`#:strike <Number>`)** → The price at which an option is exercised.
- **Expiration (`#:expiration <Number>`)** → How long the option has until expiration.
- **Premium (`#:premium <Number>`)** → Cost of the option.
- **Quantity (`#:quantity <Number>`)** → The number of options purchased.

Strategy

A combination of options that are part of the same stock.

Mapped in:
`racket (define-option-strategy <strategy-name> #:ticker <String> #:current-price <Number> (<option1> <option2> ...))`

These variables represent:

- **Ticker (`#:ticker <String>`)** → The name of the stock (e.g., "AAPL").
- **Current-price (`#:current-price <Number>`)** → The current price of the stock.
- **Option list (`<option1> <option2> ...`)** → List of options that make up the strategy.

Portfolio

A combination of strategies, making up a portfolio. Each strategy represents an individual stock, making a portfolio a collection of options for different stocks.

Mapped in:
`racket (define-portfolio <portfolio-name> (<strategy1> <strategy2> ...))`

List of Strategies (`<strategy1> <strategy2> ...`)

- A portfolio consists of multiple strategies, each containing different options.

Other Planned Functions

```racket option-payoff: (HashTable Number) -> Number
; Computes the profit or loss for a single option at a given stock price.

strategy-payoff: (Listof HashTable Number) -> Number
; Computes the total profit/loss for an option strategy.

plot-strategy: (Listof HashTable [#:x-min Number] [#:x-max Number]) -> Void
; Plots the combined payoff function of one or more options.
```

compute-breakeven: (Listof HashTable) -> (Listof Number)
; Finds breakeven prices where net payoff is zero.

compute-max-loss: (Listof HashTable) -> Number
; Determines the maximum loss for a strategy. ```

Example Usages

Defining an Option

```racket (define-option call1 #:type 'call #:action 'buy #:strike 100 #:expiration 30 #:premium 5 #:quantity 1)

(define-option call2 #:type 'call #:action 'sell #:strike 110 #:expiration 30 #:premium 3 #:quantity 1) ```

## Defining a Strategy (e.g., Bull Call Spread)

`racket (define-option-strategy bull-call-spread #:ticker "AAPL" #:current-price 105 (call1 call2))`

## Defining a Portfolio

`racket (define-portfolio my-portfolio (bull-call-spread bear-put-spread long-straddle))`

## Example Usage with Payoff Functions

```racket (strategy-payoff bull-call-spread 110) ;; Evaluate strategy at $110

(strategy-payoff bear-put-spread 85) ;; Evaluate strategy at $85

(portfolio-payoff my-portfolio 100) ;; Evaluate entire portfolio at $100 ```

# Implementation Milestones

## Phase 1: Core DSL Syntax

**Implement basic option payoff functions**
- Create `option-payoff` to compute the profit/loss for a single option at a given

stock price.
- Example: Calculate how much a trader gains or loses for a **single call or put option**.

**Implement `define-option` macro**
- Introduce a macro to define **option contracts** in a structured way.
- Ensures each option has the correct attributes (strike price, expiration, premium, etc.).

## Phase 2: Multi-Option Strategies

**Implement `define-option-strategy` macro for multiple-leg strategies**
- Allow users to define **strategies** that contain multiple options.
- Example: A **bull call spread** (buying one call, selling another).

**Extend `option-payoff` to handle single-leg & multi-leg strategies**
- Modify `option-payoff` so it can compute payoffs for **an entire strategy**, not just single options.
- Example: If a strategy has two options, their payoffs should be **added together**.

**Implement `compute-breakeven`**
- Calculate the **breakeven point** where the strategy makes $0 profit/loss.
- Helps traders understand at what stock price they stop losing money.

**Implement `plot-strategy`**
- Create a function to **visualize** option payoffs using Racket's `plot` library.
- This allows traders to see **profit/loss across different stock prices**.

## Phase 3: Advanced Features & Optimization

**Add error handling and static validation**
- Ensure that users **cannot define invalid options** (e.g., missing required fields).
- Catch errors **before runtime** for better user experience.

**Improve performance of payoff calculations**
- Optimize how strategies are evaluated, especially for **large portfolios**.
- Reduce unnecessary calculations to improve efficiency.

**Implement portfolio-level analysis (future feature)**
- Allow users to define a **portfolio** of strategies across different stocks.

- Example: Compare the total profit/loss of **AAPL, TSLA, and GOOG options together**.