CalcLyst: Computational Mathematics Language

Jiacheng Zhang (Tester), Yizhen Zhang, Zoe Ma (System Architect)
JunBo He, Fang Quan (Language Guru), Lanyue Zhang (Manager)

Feb 9 2024

1 Motivation

The proposed language aims to serve as a bridge between mathematical modeling and functional programming, harnessing the power of OCaml to deliver a robust platform for computational mathematics. By combining a declarative style with strong, static typing and scoping, our language ensures that models are not only written in a form that closely mirrors mathematical notation but also benefit from the safety and predictability of functional programming. The inclusion of higher-order functions and partial application allows for a flexible and expressive syntax that can adapt to complex modeling scenarios. This language is particularly suited for users who require accurate and intricate mathematical models and visual graph representations, such as researchers, data scientists, and engineers in fields like finance, physics, and systems biology.

2 Paradigms and Features

- **Functional Paradigm**: Embraces OCaml's efficient compilation mechanisms.
- **Declarative Programming**: Allows users to articulate computation logic without explicit control flow for enhanced readability and maintainability.
- **Strong, Static Typing and Scoping**: Aims to eliminate runtime errors and accidental variable shadowing.
- **Strict Evaluation**: Guarantees consistent performance outcomes.
- **Higher-Order Functions & Partial Application**: Offers potent tools for abstracting and composing complex functions.
- **Mathematical API-Style Interface**: Facilitates a user-friendly environment, mirroring mathematical conventions.

- **Caching Mechanism**: Optimizes performance for variables that are accessed frequently.
- **Efficient Compilation Process**: Leverages OCaml to translate high-level constructs efficiently.
- **Embedded DSL for Graph Operations**: Enables intuitive model visualization, comparable to GraphCL.

3 "Hello World" Program

```
VOID <- FUNC greet_world:
  Printf.printf("Hello, World!");

VOID <- FUNC main:
    greet_world();
```

This simple program demonstrates the syntax, with a function declaration that prints a greeting to the world. It showcases the use of functions and the language's approach to IO operations.

4 Language in one-slide Program

```
-- Define a function for factorial calculation
INT <- FUNC factorial: INT n:
    IF n == 0 OR n == 1 THEN
        RETURN 1
    ELSE
        RETURN n * factorial(n - 1)
    ENDIF

-- Use the factorial function to calculate the value of an integer
INT n = factorial(k)

-- Define a function for permutations calculation
INT <- FUNC permutations: INT n, INT r:
    RETURN factorial(n) / factorial(n - r)

-- Visualize a simple graph using built-in functions
GRAPH g = Graph.create()
Graph.add_nodes(g, ["A", "B", "C"])
Graph.add_edges(g, [("A", "B"), ("B", "C")])
Graph.draw(g)

-- Main entry point of the program
VOID <- FUNC main:
    INT perm = permutations(5, 3)
    Printf.printf("Permutations of 5 taken 3 at a time: %d\n", perm)
```

This program demonstrates the newly designed language's fundamental functional capabilities, including defining recursive functions for mathematical calculations and utilizing conditionals for logic flow. It showcases a custom-designed Graphical Domain-Specific Language (DSL) for graph construction and visualization, highlighting the language's ability to work with complex data structures visually. Through examples like factorial calculation and permutations, it illustrates how the language simplifies complex programming concepts, making it suitable for computational mathematics and data visualization tasks.