

CMPSC **460**

PoPL - principles of programming languages (Spring 2023)

**Multiple Dispatch from Julia in Racket**

Hansi Seitaj

Dr. Nate Wagenhoffer

Submitted On: March 12, 2023

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| 1 | INTRODUCTION | 2 |
| 2 | BACKGROUND (optional) | 2 |
| 3 | DESIGN & IMPLEMENTATION | 5 |
| 4 | RESULTS / SAMPLE OUTPUTS | 6 |
| 5 | CONCLUSION | 6 |
| 6 | REFERENCES | 6 |

1. **INTRODUCTION**

Multiple dispatch is a programming paradigm that allows a function to have different behaviors depending on the types of its arguments. Julia, a high-performance dynamic programming language, has been praised for its efficient implementation of multiple dispatch, which has enabled developers to write code that is both flexible and performant.

There is a library implementation of multiple dispatch in Racket has been heavily influenced by Julia, and the two communities have collaborated on several occasions to share ideas and code. The result is a powerful set of tools that allow developers to write code that is both elegant and efficient, and which can be easily extended and customized as needed. However, in this final project I decided to implement this amazing feature in Racket from scratch.

1. **BACKGROUND**

In most programming languages the feature of multiple dispatch for the method structure is not present. Furthermore, these programming languages use a feature that is called ‘parameter overload’ to facilitate the abstract notion of making the same method accept different parameters. Moreover, since most programmers are familiar with parameter overload, I believe that is important to emphasize the difference between these two features to get a better understanding of ‘multiple dispatch’.

* The difference between multiple dispatch and parameter overload.

Multiple dispatch and parameter overload are two approaches to function dispatch, but they differ in how they determine which method to call when a function is invoked with arguments.

Parameter overload (or ad hoc polymorphism) is a dispatch mechanism based on the number or type of function arguments. In parameter overload, the behavior of the function is determined by the number and types of its arguments, and each method is defined with a specific set of parameters. For example, in Java, method overloading allows you to define methods with the same name but different parameter types.

On the other hand, multiple dispatch is a dispatch mechanism based on the types of arguments involved in the function call. In multiple dispatch, the behavior of the function is determined by the types of all its arguments, and each method is defined with a specific combination of argument types. For example, in Julia, functions can be defined with multiple methods that differ in the types of arguments.

**The key difference between the two approaches is that parameter overload is determined by the number or types of arguments, while multiple dispatch is determined by the combination of argument types.** Multiple dispatch allows for more flexibility in function definition, as it can handle more complex scenarios where the behavior depends on the interaction of multiple arguments. It also allows for easy extensibility, as new methods can be defined to handle new combinations of argument types without modifying existing code.

In summary, parameter overload is based on the number or types of function arguments, while multiple dispatch is based on the types of all arguments involved in the function call.

1. **DESIGN & IMPLEMENTATION**

**Code Structure**

The code consists of a custom implementation of type-of procedure that returns the type of the input argument. A hash table method-table that stores methods and their corresponding argument types. A butlast procedure that takes a list as input and returns a new list containing all the elements of the original list except the last element. A find-method procedure that finds the best matching method for a given set of arguments. Finally, the define-generic procedure that implements a generic function, which is used to call the corresponding method with the input arguments.

**Functions and Methods**

* type-of

The type-of procedure is defined as a custom function that takes an argument x and returns its type. The implementation is done using the cond construct to check the type of the argument and return a corresponding symbol. This procedure is used in the find-method procedure to get the type of arguments.

* method-table

The method-table is a hash table that stores methods and their corresponding argument types. It is implemented using the make-hash function.

* add-method

The add-method procedure adds a method to the method-table. It takes three arguments: name, types, and f. name is the name of the method, types is a list of argument types, and f is the procedure that implements the method.

* butlast

The butlast procedure takes a list as input and returns a new list that contains all the elements of the original list except for the last element. It is used in the find-method procedure to generate new sets of argument types.

* find-method

The find-method procedure finds the best matching method for a given set of arguments. It takes two arguments: name is the name of the method, and args is a list of arguments. The procedure first gets the type of arguments using the type-of procedure and generates a list of types. It then uses a loop to iterate over the list of types and tries to find the corresponding method from the method-table. If a matching method is found, it returns the method. Otherwise, it removes the last type from the list and repeats the process. If no matching method is found, an error is raised.

* define-generic

The define-generic procedure implements a generic function. It takes a name argument and returns a lambda that takes a list of arguments. It calls the find-method procedure to get the best matching method for the input arguments and applies the method using the input arguments.

* Add

The add procedure is an example of a method with multiple dispatch. It is implemented using the define-generic procedure. Two methods are added to the method-table, one for adding two integers and one for adding two floats. The add procedure calls the define-generic procedure with the name add.

* add-str

The add-str procedure is another example of a method with multiple dispatch. It is implemented in a similar way as the add procedure.

1. **RESULTS / SAMPLE OUTPUTS**

The code also includes several test cases to ensure that the procedures and functions work as expected. These test cases include testing add and add-str with integer and string arguments, respectively, and testing type-of with various argument types.

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

All these test cases are made to provide complete test coverage and the last two test cases reveal the scenario of when the program will fail and return error based on the type mismatch.

1. **CONCLUSION**

All in all, the benefit of the multiple dispatch implementation is that the program takes advantage of type restriction in method signature and can improve the sorting operation performance significantly. It can help programmers test more efficiently and save a significant amount of time. Moreover, it speeds up the process for algorithms to work the data structures.

1. **REFERENCES**

<https://freecontent.manning.com/using-multiple-dispatch-in-julia/>

<https://docs.racket-lang.org/>

<https://docs.julialang.org/en/v1/>