# CHAPTER I

# INTRODUCTION

The original λ-calculus was first invented by Alonzo Church ([1]), a recent exposition which covers most of the major developments in the theory is the text by Hindley and Seldin ([2]). The following brief introduction to the original, untyped λ-calculus is derived from this second text.

Given an infinite sequence of distinct symbols called **variables** and a disjoint set of symbols called **constants**, we can define **λ-terms** as follows.

Each variable and constant is a λ-term, called an **atom**, and if M and N are λ-terms and x is a variable, then (MN) and (λx.M) are λ-terms, called an **application** and an **abstraction**, respectively.

An abstraction (λx.M) is used as notation for a function. To be precise, it represents the function f such that $f : x \mapsto M$. Unlike in set theory, functions in λ-calculus are not sets of ordered pairs with domain and range given. Instead, we can think of a function as an operation which may be applied to certain objects to produce other objects. The λ-term M tells how to calculate the new object from the original one.

An occurrence of a variable x in a λ-term L is **bound** if it is in a part of L of the form λx.M; otherwise it is **free**. If x has at least one free occurrence in L, it is called a **free variable** of L; the set of all such variables is denoted by FV(L).

To keep things simple, we will omit the definition of substitution and use it in an intuitive way. We use [N/x]M to denote the result of substituting N for all free occurrences of x in M.

If a λ-term L contains an occurrence of (λx.M) and $y \notin FV(M)$, then the act of replacing (λx.M) by (λy.[y/x]M) is called a **change of bound variable** in L. We say L is **congruent** to a λ-term L′, denoted by $L \equiv_\alpha L'$, if L′ is obtained from L by a finite sequence of changes of bound variables.

As we mentioned above, an abstraction λx.M represents the function

$f : x \mapsto M$, where x is a bound variable in $\lambda x.M$. Since the only differences between congruent terms are bound variables, congruent abstractions represent the same function. So congruent terms have identical interpretations and play identical roles in any application of the λ-calculus.

A **β-redex** is a λ-term of the form $((\lambda x.M)N)$, which represents an operator $(\lambda x.M)$ applied to an argument N. The result of this application is $[N/x]M$, which is called a **contractum**.

If a λ-term L contains an occurrence of a β-redex and we replace that occurrence by its contractum, and the result is M, we say L **β-contracts** to M, denoted by $L \rhd_{1\beta} M$. We say L **β-reduces** to M, denoted by $L \rhd_{\beta} M$ if M is obtained from L by a finite sequence of β-contractions and changes of bound variables.

As mentioned above, each occurrence of a β-redex in a λ-term L represents a function applied to an argument and each β-contraction gives the result of calculating the value of such an application. If a λ-term L reduces to a λ-term M such that M contains no β-redex, then we can think of M as being the result of doing all of the calculations indicated in L. Since we expect the λ-calculus to be a model of doing calculations with functions, this result should be unique, up to changes of bound variables. It is indeed unique because of the Church-Rosser theorem, which states that for any λ-terms L, M and N, if $L \rhd_{\beta} M$ and $L \rhd_{\beta} N$, then there exists a term T such that $M \rhd_{\beta} T$ and $N \rhd_{\beta} T$.

The above is a brief explanation of how the λ-calculus can be used to represent a function and to interpret the result of applying a function to an argument. However, there are limitations in the original λ-calculus, which we can see from the following.

First, consider the successor and predecessor functions. If we let S be a constant representing the successor function, then we can also think of the predecessor function which maps Sx to x. If we apply this function to an argument, the result can be obtained only if that argument is of the form SM and the result of this application is M, which is obtained by removing S from the argument. We can see that the original λ-calculus cannot express such an operation as a λ-term. To

overcome this problem, in this thesis we will define a new class of terms, called patterns, which will be used after the symbol λ in abstractions to specify the form of the argument accepted, and to extract subterms of the argument.

Second, we can see that any function which is defined differently for some arguments cannot be represented by a single λ-term. Consider the above example again. The predecessor function is defined differently for zero and for positive integers, namely, it maps zero to zero and maps positive integers to their predecessors. So we need at least two λ-terms to represent this function. Therefore we will modify the definition of terms to allow a kind of "definition by cases".

So we will construct a new λ-calculus which can describe a larger class of functions, yet still satisfies all of the basic properties of the original λ-calculus, including the Church-Rosser theorem.

We are not the first ones to think about adding patterns and "definition by cases" to the λ-calculus. Most modern functional programming languages are modeled on the λ-calculus and include both of these features (see, for example, [3]). The paper by Breazu-Tannen et al. ([4]) presents a λ-calculus with these features as well. However, the functional programming languages and the λ-calculus of Breazu-Tannen et al. use typed λ-calculi and are oriented towards computer science, whereas the λ-calculus we describe is untyped, and slanted strongly towards mathematical logic.

In Chapter II, we will give the basic definitions of the new λ-calculus and prove some basic properties. Chapter III discusses the Church-Rosser theorem, consisting mainly of preliminary lemmas followed by the proof of the theorem itself. Chapter IV summarizes the results of our work, and suggests possibilities for further research.