0.1 Introduction

Functional programming is a programming paradigm in which programs are structured as a composition of pure functions [**Hughes1989WhyMatters**]

Pure functions, by contrast to function constructs of programming languages, refer to the mathematical concept of a function. In mathematics "a function f from A to B, where A and B are non-empty sets, is a rule that associates with each element of A (the domain) a unique element of B (the codomain)" [**NicholsonTheMathematics**].

Figure 1: A visual representation of the function f

Although this is the mathematical definition, in the domain of a programming language it could also be stated that "a pure function has no observable effect on the execution of the program other than to compute a result given its inputs" [**Chiusano2013FunctionalScala**]

However, functions constructs in many programming languages don't have the traits of mathematical functions. As an example let's take this function written in [introduce programming language]. [Explain example].

The elements which make language functions not perform as mathematical functions are called side effects. To make differentiation functions that don't have side effects are referred to as pure functions[**UsingAttributes**] while functions that have side effects are called procedures.

Side effects that can make a function not pure are [**SpulerCompilerEffects**].

- Performing I/O
- Modifying a variable
- Modifying a data structure in place

[[Set examples of broken purity]]

0.2 Referential transparency

When treating with pure functions there is a one to one relation betweeen a function call and the result it produces. For example we can see that the expression $2 + 2$ is the same as 4.