Explaining a type system in a language understandable by an undergraduate student.

A dissertation submitted in partial fulfillment of the requirements for the degree of Bachelor of Science (Honours) in Software Engineering

By D.W. Evans

Department of Computing & Information Systems
Cardiff School of Management

Cardiff Metropolitan University

April 2017
Declaration

I hereby declare that this dissertation entitled ‘Explaining a type system in a language understandable by an undergraduate student’ is entirely my own work, and it has never been submitted nor is it currently being submitted for any other degree.

Candidate: Dylan W Evans
Ethics Number: 2016D0340

Signature:

Date:

Supervisor: Ana Calderon

Signature:

Date:
Acknowledgements

This research was supervised by Ana Calderon on whom I would like to thank for advising my research project. I thank Cardiff Metropolitan University for providing me with the access to services including the library and desktops that were needed to complete the study.

I would also like to thank other members of staff at Cardiff Metropolitan University along with Stuart McNeil who on a number of occasions provided guidance towards the project. I thank the publishers who gave me the material to read and use in my research project.
Abstract

This study analyses the complexity of a type system and its underlying purpose, an in-depth secondary research from sources such as books, articles, and journals that show the different roles a type system can have in different languages. Demonstrating to the reader a thorough breakdown of type systems, given that the theory of types has changed much over the years be able to interpret different thesis by different theorists and achieving a clear understanding for undergraduate students.

The acknowledgment of the research already published leads to the study being undertaken and scrutinizing type systems thoroughly, then interpreting the broken down type systems to a different audience giving others the chance to use, understand and expand on the principles of a type system. There will be a breakdown of logical understanding of type systems showing assumptions and conclusions that are created before the type systems get implemented into a programming language. Grasping the concepts giving the readers an understanding of where a type systems implementation begins starting from their function in programming languages and the history of type systems.

A clear breakdown of type systems will show the expressions and rules that apply to those type systems, the different possibilities that can occur such as derivation and subtyping. The link between a static and dynamic type system and the benefits each has for a programming language.

The finished project and conclusion will break down a type system and recognize the differences between various type systems converting that area of the research from postgraduate level to undergraduate students to apprehend. In addition, accomplishing my aims and objectives that are met by the completion of this study under the time frame given.
# Contents

1.0 Introduction ........................................................................................................... 1

1.1 Aims and objectives ............................................................................................... 2

1.11 Introduction .......................................................................................................... 2

1.12 Aim ....................................................................................................................... 2

1.13 Objectives ............................................................................................................ 2

2.0 Methodology .......................................................................................................... 3

2.1 Introduction .......................................................................................................... 3

2.2 Research methods ................................................................................................. 3

2.3 Qualitative Research ............................................................................................. 3

3.0 Literature review .................................................................................................. 4

3.1 Introduction .......................................................................................................... 4

3.2 History of programming languages ...................................................................... 4

3.3 History of type systems ....................................................................................... 6

3.4 Fundamentals of type systems ............................................................................ 7

3.5 Type checking combining static and dynamic type checking ......................... 8

3.6 Type safety ........................................................................................................... 9

3.7 Types of programming ....................................................................................... 9

3.8 Data types ............................................................................................................ 9

3.9 Compiler correctness .......................................................................................... 10

3.10 Subtyping and derivation ................................................................................. 10

3.11 Summary ............................................................................................................ 10

4.0 Analysis and discussion of Type Systems ............................................................ 11

4.1 Introduction .......................................................................................................... 11

4.2 Type systems defined .......................................................................................... 11

4.3 Typed and un-typed languages ......................................................................... 12

4.4 Execution errors ................................................................................................. 12
List of Figures

Figure 1.0 Timeline of programming languages ................................................. 5
Figure 2.0 ............................................................................................................ 15

List of Tables
Table 1.1 Logical operators ........................................................................... 18
1.0 Introduction

"A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute." Pierce (2012). Type systems are sets of rules which assign types to many different constructs in a computer program, computer programs consist of variables, expression, and functions. A type of computer programming is a type of data which tells the compiler how the program is being used and what the data is being interpreted for, a types reasoning is to set values for which an expression can use those values.

The significance of this project is that undergraduate students can understand the complexity of type systems undertaken and are able to approach the project and better the study, adding their own knowledge. The relevance is to adapt and create new research material for students with the research already published. Throughout this research project, the purpose is to answer the title, which is to explain a type system in a language understandable to an undergraduate student. To achieve the title there has been an aim set, understanding the complexity of type systems and the project objectives which is first to understand how type systems work and their purpose, then second being able to understand type safety and its objective. The third is analyzing and defining a type system.

The literature review will give the reader a thorough background of type systems and the current literature around that topic area giving the reader a clear understanding of what the research project is targeting. The analysis and discussion section has been split into 3, section 1 includes the considerations taken to implement a successful type system, what the analysis and discussion of a type system will aim to do is examine an in-depth knowledge of the different attributes. Adjacent to this research into how type systems can differ such as the type system being static, un-typed and even the advantages towards using a type system will be analyzed. Section 2 explains how a type system can be implemented into different programming paradigms or into a combination of more than one programming paradigm. Section 2 also aims to educate the reader about type safety and why it is enforced to benefit a programming language. Section 3 breaks down a type system detailing the steps that are considered when different values are applied to the rules that can define and distinguish the assumption and conclusion of that type system.

The amount of data has increased considerably over the last couple of decades and is continuing to do so in predicting world trends in both our personal and work lives. Data can be used to predict simple operations such as shopping trends or more complicated processes such as suitability for job roles. There is a lot of programs written to deal with the increasing and varied demands on data. However, organizing this data and simplifying it into varied types such as strings, integers, Boolean and numerous others is essential when classifying an object.
With regards to data, there are many ways in which outcomes may be controlled, for example, simple processes adding or subtracting numbers or even resizing an image. Not all operations are this simple, however, for example, if someone were to times Boolean by a string this would not work and this would be referred to as a type error. These anomalies are most likely to occur with dynamically typed languages, which has been implemented with a type system.

1.1 Aims and objectives

1.1.1 Introduction

The project will have the following aims and objectives which resorts back to answering the title of the project of the research project, referring back to these aims and objectives throughout the project to assure that they are being answered keeping on track to being achieved.

1.1.2 Aim

Understanding the complexity of type systems.

1.1.3 Objectives

To understand how type systems work and their purpose.

Understanding type safety and its objective.

Analyze and defining a type system.
2.0 Methodology.

2.1 Introduction

The methodology is used to show how different strategies and methods were used to collect the information for the research project. This methodology will analyze the research methods formed to undertake this study along with the qualitative research and secondary research.

2.2 Research Methods

The research methods I have used in this research project has been secondary research rather than collecting questionnaires, surveys etc. which would have been primary research, to prosper from the research that has already been published. Following the research, onion model sees in appendices 1A this gave me the right approach and research methods to use whilst using secondary research. Secondary research includes books, journals, and articles that have already been published to gain a clear understanding.

Starting with pragmatism which is the approach to evaluating theories already created from successful discoveries for my philosophy, by using studies from professors that have already laid the basis for type theory for example. Following the second layer for the approach using inductive to generate a new theory from the data already collected. The strategies that were frequently used in the project was archival research which included the use of articles and journals published by the likes of Cardelli and Pierce but a smaller section known as an ethnography strategy because of the time scale that these theoreticians uncovered the theories. Following the research onion, this project used qualitative research a mono-method approach, as the entire study used exploratory research to find and understanding of the underlying opinions and theories already set in motion. A cross-sectional time frame was used in this study as the research was set over a shorter period of time compared to a longitudinal study, as the data analyzed has already been collected from previous studies. For the data collection and data analysis, I gathered research that has already been published using books, articles, and journals, not collecting any new data to use in my project.

2.3 Qualitative research

“Qualitative methods examine the why and how of decision making, not just what, where, when, or “who” (Alasuutari, 2010). Using different publishings and forms of data that has been gathered through observation, through reports and different studies that have been recorded. Qualitative Research focuses on the information that has already been studied and interpret that into another level of meaning.
3.0 Literature review

3.1 Introduction

Type systems play a big role within programming languages and define how that language can flow and has been the same all the way in history since type systems were first discovered or implemented. Starting back when the first programming language was invented there was talks of using a type system based on type theory to be applied to a language to limit errors and increase the safety within that language whether it be with Fortran or before Fortran’s time. Throughout this literature review, it will illustrate the history and current literature that has been published on the topic of type systems and the views and theory based on the research project.

3.2 History of programming languages

Before a programming language was even thought about there was a technique known as a step to wire method and was first experimented with Charles Babbage’s difference engine in 1822, then further down the line, they knew that a means of commands was needed via a language. So before a language was created technically the first was known as physical motion but manually changes gears and wires between the interface (A history of computer programming languages, 2004). There are many different programming languages but it is really hard to say what language come first as there were few programmable computational machines in history that were created from the 1940s onwards. As quoted by Hagen “s. The MARK computers by Howard Aiken, Konrad Zuse’s Z3, Turing’s COLOSSUS and ACE-machine, and Mauchly-Eckert’s ENIAC participate in the first developments. There is a ubiquity of beginnings, a dissipation of the mechanic start-up of the computer (Hagen, W, 2006).” Hagen is implying that these computers were all created closely together so it is hard to identify which was the first. But by what has been recorded in history that the Konrad Zuse’s Z3 was the world’s first programmable computer created in 1941 with a mixture of binary and floating point arithmetic under the name of Plankalkul. Except Zuse’s Z3 was to only be recognized years later so that argument with Hagen supporting the Z3 and Flanagan saying it was Von Neumann who first created a language.

The first big language that really became anything was known as FORTRAN this was logical and would allow if, then and goTo statements, with basic data types Booleans, integers, real and double precision numbers (A history of computer programming languages, 2004). The other big languages came in 1970 with Pascal created by Niklaus Wirth used in Apple Lisa and Skype, this was set to be a teaching language used across the windows application development. The next major language created in 1972 known as C by a gentleman Dennis Ritchie, C had a big part to play with Unix having been rewritten in 1973. Next followed in 1983 two different extensions of the language C one created by Bjarne Stroustrup known as C++ this was formally C but with the added classes more of an incremental C, this becomes very popular being used for commercial development or client/server applications by the likes of Google
Chrome, Mozilla Firefox, and Adobe. The other by Brad Cox and Tom Love called Objective-C growing on the original C but adding the Object Orientated aspects to it this became Apples primary language for OS X and IOS operating system. Many languages came after that with Pearl being released in 1987 by Guido van Rossum, Ruby in 1993 by Yukihiro Matsumoto and then the world’s most popular language to date released in 1995 Java. The creator James Gosling of sun microsystems developed this language which became purposeful for network programming, software development, and GUI development, Android OS became a recurring user of this language. Additionally, two other languages that were founded in 1995 were PHP by Rasmus Lerdorf for open source development, followed by JavaScript this was created to extend the development of web page functionality and it worked perfectly Brenden Eich created something amazing *(Visually, 2017)*.

**TIMELINE OF PROGRAMMING LANGUAGES**

![Timeline of programming languages](Slide share,2017)

As spoken about previously these languages can be based off just one programming paradigm or multiple paradigms similar to functional, imperative, object orientated etc. The programming languages that are most likely to be used today you can see on figure 1.0 these have been somewhat trial and error over the decades some still being used to this day from its first release into the programming industry. When using one language and tweaking that language to make it compute more efficiently or to suit those programmers that are how one can progress, adjusting languages can come with the advancing technology over the years. An example of this has been seen over the last two decades coming from the first release of the Nokia 3310 in 2000 which at the time was a great success. If we look further on in time to 2009 only 9 years later when the iPhone 3 was placed on the market the difference between these two technologies were incredible. So even taking one language and developing that over
time, for example, Java 7 and Java 8 which originated from Java which was invented back in 1996-1997 it shows us that technology progressing alters the way we program.

3.3 History of type systems

A baseline for type systems history began back in 1902 with Russel’s Paradox known as a very famous paradox in logical mathematics, this was created to set aside any difficulty in the paradoxes preventing the variety of formal logic by Bernard Russel. For those who do not know Russel’s Paradox is the theory that the set of all sets are not members of themselves, a set can be a member of itself if and only if it is not a member of itself (Frege, 1902). After Russel’s discovery of logics, he then ramified his theory of types leading Quine to observe Russel’s work which he called “pointless apart from a certain aspect of the theory of types” Quine’s commentary before (Russell, 1908) disclosing the bound variable of the notion. So Quine came up with something that really brings it together he said “It has been so called because the type of a function depends both on the types of its arguments and on the types of the apparent variables contained in it (or in its expression), in case these exceed the types of the arguments”. Quine explains that the argument cannot be exceeded by its variables and will equally play a part in the function and there given types. But Stephen Kleene has argued that the ramified theory has different aspects and said “The primary objects or individuals (i.e. the given things not being subjected to logical analysis) are assigned to one type (say type 0), the properties of individuals to type 1, properties of properties of individuals to type 2, etc.; and no properties are admitted which do not fall into one of these logical types (e.g. this puts the properties ‘predicable’ and ‘predictable’ ... outside the pale of logic). A more detailed account would describe the admitted types for other objects as relations and classes. Then to exclude impredicative definitions within a type, the types above type 0 are further separated into orders. Thus for type 1, properties defined without mentioning any totality belong to order 0, and properties defined using the totality of properties of a given order belong to the next higher order. ... But this separation into orders makes it impossible to construct the familiar analysis, which we saw above contains impredicative definitions. To escape this outcome” Kleene mentions that you have to be assigned a type i.e. string, Boolean, Int if a value is not passed it has no assigned type. Kleene then goes on to say about Russel’s axiom of reducibility “Russell postulated his axiom of reducibility, which asserts that to any property belonging to an order above the lowest, there is a coextensive property (i.e. one possessed by exactly the same objects) of order 0. If only definable properties are considered to exist, then the axiom means that to every impredicative definition within a given type there is an equivalent predicative one.”

If ramified theory of types could be collapsed which was identified by Chwistek and Ramsey with ruling out the theory of the vicious circle principle which is a notion that no object may be presented by a meaning that depends on that very object. For example, an object can contain S as long as that object is not next to S, once this principle has been ruled out Russel’s paradox can still be applied as these principles contradict each other. This new concept would then be known as the theory of simple
types having been documented in the 1930’s for publishing (Russel, 1908), but formerly we know was further established as the simply typed Lambda-calculus by A. Church in the 1940’s which directed K. Gödel to draw up his own conclusion of the theory of simple types. “By the theory of simple types, I mean the doctrine which says that the objects of thought (or, in another interpretation, the symbolic expressions) are divided into types, namely: individuals, properties of individuals, relations between individuals, properties of such relations” (Gödel, 1944). These discoveries that have been based on the theories of others or their own have been proven and disproven over the years, for instance from type theory many arguments were disputed saying that the theory contradicted itself and if certain aspects were to be ignored it would be more absolute. These theories will still be disputed in the world of mathematicians as more discoveries will be made in the future to redefine what the world know as now.

3.4 Fundamentals of type systems

The fundamentals of a type system are originally studied as type theory which states that programming languages must be able to type check statically or dynamically (compile time or run time). “The fundamental problem addressed by a type theory is to ensure that programs have to mean. The fundamental problem caused by a type theory is that meaningful programs may not have meanings ascribed to them. The quest for richer type systems results from this tension.” In the words of Menasse (Pierce, 2002). So the argument is that the type system was based on the type theory setting and applying rules to a language decreasing the error that may occur within that language, but also Menasse relates that there should be some point in that program applying values to types otherwise what purpose what that language have other than computing random calculations. Applying rules to the language set the baseline for security, it adds that extra layer just so that if any code isn’t intact then an error is thrown to the user assuring that language and its type system do not correspond and need altering.

According to Pierce “In programming languages, a type system is a set of rules that assign a property called type to various constructs a computer program consists of, such as variables, expressions, functions or modules” (Pierce, 2002). This statement can be further agreed by defining that a type systems purpose is to reduce error as follows: “The main purpose of a type system is to reduce possibilities for bugs in computer programs by defining interfaces between different parts of a computer program, and then checking that the parts have been connected in a consistent way.” (Cardelli, 1996). What Cardelli means by the checking process is that the compiler can check for errors within the programmer’s code at compile time known as static checking or at run time which is known as dynamic checking, usually the compiler is enforcing the typing rules set by its type system as the programmer/user is inserting their code. It does this by assigning a type to each value that needs to be processed through, the type system will then analyze the giving values to ensure that there are no errors that will affect the efficiency of the program or code that is being run. If there are no errors then you have success but not all languages have the same type system,
they can be different in how they approach the code, some type systems are stricter and work better than others. If a language can be known to be type safe when the type system does not allow the program to run operations with any errors, but that is not what some programmers want they want to be able to use a language without identifying values to types they want the operation to run whether it have an error or whether it be strongly typed. It has been suggested that a type system should be selected correspondingly with its language being able to select different type systems to any language as an effective plugin when desired (Bracha, 2004). He believes this approach to be beneficial towards programmers, that when a language and a type system are fixed then that language is sensitive and less presumptuous. Expanding the flexibility of that language and by being able to insert a type system into a language is giving the user full control on selecting the best possible combination for the work that is needed to be achieved in a giving timescale for a given price.

3.5 Type checking combining static and dynamic type checking

Static type checking is a procedure of checking type safety within a programming language that analyses the code, if the code passes the static type check then some sort of type safety is guaranteed within that program if it doesn’t then an error is thrown in the compiler. Static type checking allows detection of an error or default in the program earlier on rather than later which without it would delay timing and could potentially affect the costs. But languages can enforce good behavior by performing static checks to eliminate unsafe or bad behaved programs from even getting a sniff of being executed (Cardelli, 1996). The algorithm that checks for this is known as a type checker which shows us that the program is well-typed if it does not get passed then it is known as ill-typed. Dynamic type checking is a procedure that can verify type safety at runtime, it applies a tag to an object at runtime which contains some sort of information about the type. But with a dynamic type when there is an occurrence it can bypass compile-time (static) and check at run time some programmers prefer this because they do not need to have variables before the program is run. It makes sense that some statically typed languages would still have some sort of dynamic checking after the compiler has run the program as it can still run an error so having this implemented as well enhances type safety. Both static and dynamic has a type checker which can be used in both, but with dynamic checking, it may not have static checking before the program is run then this is known as a dynamically typed programming language (Cardelli, 1996). A language that allows both are usually referred to as being soft-typed Java and many other languages support this, being soft-typed has its advantages as the process of eliminating errors becomes more reliant. But for example with Java, it uses both static and dynamic because it supports a technique known as type refinement as it makes a reference to changing one of its data types in the main class to its sub-class, which this will be checked dynamically making it a soft-typed language.
3.6 Type safety

Type safety is how a programming language decreases the number of typing errors occurring in a program, a type error is just an occurrence where two data types cannot process in a function or be declared. Type safety can be used statically at compile time or dynamically at run time or even at both to identify errors with a program, which in a static type system the type safety can guarantee that there will be no errors. Type safety works differently in languages, they can be labeled as strong or weak typed down to the occurrence of type safety. “the idea of strong type checking and the language translator is supposed to enforce strong type checking even across the interface between two separately compiled procedures.” (Liskov and Zilles, 1974). They say that if type safety is strong then in that program when a calling function passes and object to an already called function both types must be compatible with each other. Whereas type safety implemented weak can throw type errors or may perform actions of implicit typing.

3.7 Types of programming

Imperative programming uses statements and commands to change the program’s state. A name can be assigned a value but later on reassigned a different value. With imperative programming, you have a state that is manipulated by the program via a variety of commands providing structure to the code. This is because the state is an association between the memories location and its values.

Declarative programming allows that the user should only state what the user would like to see as a result of the code written and not on how to obtain that. It is usually used in data sets and involves searching through to find what is needed, expressing the logic of computation whilst distinguishing its control flows, to define its purpose of rooting out and minimizing side effects.

Object-orientated is a programming paradigm based on objects (data and methods) inside classes to use four main features inheritance, polymorphism, encapsulation and abstraction between classes that interact with other classes to form a program (tutorialSprint, 2016). It’s good for writing a function in the code for one class but allows for another class to use the same function. So, for example, you may have a parent class such as animal and instances of that class known as sub-classes or child classes such as cat, dog and cow and then a further sub-class such as the breed of dog.

3.8 Data types

A data type is a type of data that is excepted to run in the compiler or determines what the programmer is about to with that data. Compilers accept many different types such as an integer or a string and the type decides what can be done with the data it is representing whether it is stored in an array or used in a while loop (Shaffer, 1997). Data types can be classified and come in many different formats such as primitive
types that the compiler understands. A primitive type can be characters (char), integers (int, short, long and byte), floating point numbers (float and double) and Booleans (true and false), a byte is equal to 8 bits and 2 bytes to 16 bits and so on. But not are all used in a type system but can be implemented to understand and break these down into machine learning language.

3.9 Compiler correctness

Compiler correctness shows that the compiler is behaving to its languages purpose, so through thorough testing and compiler validation shows that it behaves correctly. “Optimizing compilers are so difficult to get right that we dare say that no optimizing compiler is completely error-free!” (Aho, Sethi, and Ullman, 1986). This is the link to a type system as its main purpose is to decrease if not eliminate errors, so that through implementing a type system can change the behavior of the compiler.

3.10 Subtyping and Derivation

Subtyping is the linguistic notion relation between hyponymy and hyperonymy which described in a simpler form that is Ford, Citroen and Volkswagen are all hyponyms of the car which is the hyperonym (Fromkin, Rodman, Hyams, 2010). Subtyping is also a linguistic notion of holonymy this identifies a relationship between the term denoting the term whole and term denoting a part of the whole or member of the whole. Example car is a holonym of wheels, wind mirror and car seat (Cruise, 1986). But not every car has to have a wind mirror it may be required but not needed Cruise has made that clear. This is the relation that subtyping the form of polymorphism has within a type system it allows a data type to be related to another data type and be used in its place, so in notion when the main data type known as a Supertype or Parent type has a certain operation or function for that type the same can be used on what is called its subtype in its place. So if S is a subtype of T then T can be used in any context/ function that S would be required.

The derivation is a type system making a judgment based on the ones just before that “A derivation in a given type system is a tree of judgments with leaves at the top and a root at the bottom” (Cardelli, 1996). A valid judgment can be acquired when properly using typing rules, it can also be achieved as the root of a derivation in a type system.

3.11 Summary

The literature review includes all the current research there is about type systems, covering the history of programming languages and type systems to current type systems and what makes it compatible with its language. There is also literature on type safety being implemented into the programming language and down to the data types the type system recognizes. The theories of type systems and where it began with Russel’s Paradox with the theorist views and arguments against and with.
4.0 Analysis and discussion of a type system

4.1 Introduction

Throughout the analysis and discussion section, it includes a steady breakdown of a type system, sectioning different areas of a type system and the various attributes that a type system can have. A type system can be static or dynamic, type or un-typed these can distinguish how strong or weakly typed that type system is. Also what the advantages and disadvantages of a type system are and what a type system achieves.

4.2 Type systems defined

A compiler of a programming language can identify a type error when it is being made or even before the program has been run. When the operation is selected, that operation may stop the program from working or the program may not even run, to begin with, because of the type error. This is when the compiler will identify an error occurring with the user’s code prompting the user that there has been an error occurring and that it needs to be changed or the program will remain in operational. Either way, there will be some sort of error handling for the user that will occur delaying the program prompting the user to change the error. This then is why a type system is so important, as it is used to stop the type errors occurring later in the program, so avoiding unclear results or execution of a program, both of which are troublesome and time-consuming.

The definition of a type system is then ‘a collection of rules that assign a property called type to various constructs, a computer program consists of variables expressions, functions or modules – (Roundcrisis, 2015). A type systems main purpose is to limit the number of execution errors that occur within a language whether this is at the run time or compile time. A Component that lets the user/programmer distinguish if a language, for instance, Java will have errors that can affect the flow of the program, disrupting the efficiency of the program delaying or causing problems for the current system. It sometimes happens that compilers for the same language can implement different type systems (different rules and assumptions), this allows the program to crash at the time which is handled by the type checker calling upon it to be known type unsound. Although not all languages have type systems and are not represented by the workings for some programming languages, type systems are a helpful way of studying a language when applying rules. As a type system specifies the rules of the programming language with a type checker algorithm. Type rules are applied to the language to determine if the program is well-typed (Cardelli, 2004). Type systems apply the type rules to then determine the type expression of that program.

Type systems are like the fundamentals of the programming languages, without them computing and calculations would not be possible, there would be no structure to the language but the structure is more to protect the safety of the language. Even
though not all languages need type systems it has been proven that a more secure and implicit language can be formed when a language is based off a programming paradigm. It is similar to a type system. As a type system, its’s rules apply to that language which can then be created because the rules and assumptions will make a successful language and can show that the language will work.

Type systems can be described as strongly or weakly typed, a strongly typed language predefines all its variables as one data type. One advantage of this is that it applies a hard set of rules to the programmer, guaranteeing reliable results. One disadvantage of a strong system is that the programmer is limited in what data types he can use by not being able to improvise or create new data types that don't exist in the vocabulary of the language (Whatis, 2012). A weakly type system does not define its data types thus allowing unrelated types implicit for faster development of code cutting corners as they say. One disadvantage of this is that the errors can occur when programs are run.

4.3 Typed and untyped languages

There is a range of variables that can be declared to a language before the program has been run these are defined as types, as variables are types but types are not variables. This is where typed and un-typed languages come in, languages that can be given nontrivial types are called typed languages. For example, if x has type Boolean, the Boolean expression not(x) has a sensible meaning in every run of the program (Cardelli, 1996) Being typed then the languages variables are being tracked but also the expression that is being written within the code. Untyped languages have no restriction on what variables are used as they do not have types they have a single unified type that can contain all values. Untyped languages usually have no errors occurring, since all values are functions their operations never fail. The Lambda calculus is an example of when an error will never occur as it “is a formal system in mathematical logic for expressing computation based on function abstraction and application using variable binding and substitution.” (Cardelli, 1996).

If a system is known to be well behaved, then it is also known to be a safe system. The safer the system the better you would think but that is not always the case as a safer system is usually a more expensive system taking up not just money but time. The more primitive purpose for a type system is the type safety, this is more important than its good behavior. Its other purpose is to distinguish between good and badly typed languages. This is where the decision between static and dynamic languages is chosen; static languages sound good but their safety is their fault.

4.4 Execution Errors

Type systems calculate between different errors and trapped errors that occur before the program is run. They identify problems in advance, such as syntax and language compatibility. This should mean fewer errors occurring when following the advice given. Then there are untrapped errors those that let the program run completing
methods that have been implemented through the code then an error going unnoticed causes a random error costing time and money. This may be to the language you used strongly or weakly typed did not call you out on these errors not clearly underlining or pointing out the very errors that cost companies money. Untyped languages will enforce type safety throughout the running of the program potentially saving time avoiding any possible disrupting errors. To minimize execution errors, statically typed languages introduced a type checker and this algorithm provides safety to a program by hindering an unsafe program from running. If the type checker allows the program to run it is known to be well typed, Pearl, Ruby and Python are examples of a strong, well-behaved language.

Untyped languages can show good behavior and be strongly checked. Even if the variables are not being declared they can still perform run-time checks (checking for forbidden errors) although they have no type system or type checker.

4.5 Static and Dynamic

The classification of a type system aids in enhancing the theoretical understanding of a language and its type (Static or Dynamic etc.) and also illustrates the behavior of the language. The type system can be separated into two known terms, these terms are “static” and “dynamic”. Although both have different meanings, they can be linked with each other and somehow overlap each other.

A Static system is a system/environment that applies named variables also known as “types”. This happens at compile time which is before the code is run. “A static type system is a mechanism by which a compiler examines source code and assigns labels (called “types”) to pieces of the syntax, and then uses them to infer something about the program's behaviour” (Klabnik, 2012). This allows the program to detect any bugs before the code has run making the system reliable and at times can be more efficient with less run time errors. Examples of these are languages such as Java and C++.

A Dynamic system is when the programmer does not need to declare its variables thus allowing code to be written faster. One problem with this is that bugs will only be detected at run-time. Examples of dynamic systems are pearl and python. “A dynamic type system is a mechanism by which a compiler generates code to keep track of the sort of data (coincidentally, also called its “type”) used by the program” (Klabnik, 2012).

4.6 Advantages and disadvantages of type systems

Overloading is a feature that allows you to get rid of useless code that might have been used previously but now has been replaced or extended and is no longer needed for the running of the program, this can be declared by types. Furthermore, a type system requires annotation of the code to clarify documentation which makes it readable to other programmers. Using a type system helps in the clarity of a well-
structured program. Where errors can be found at compile time within a type system, type annotation makes the documentation clearer for other programmers outlining the construct making the aims of what the constructs should be doing. A disadvantage of this is that it makes them less flexible, making the system harder to practice because “not all valid programs can be handled by a given type system. As such programmers often use workarounds, such as type casts, to bypass the limitations of the type system” (Mayer, C., Hanenberg, S., Robbes, R., Tanter, É. and Stefik, A (2012). They also obstruct rapid prototyping because arguments may not obey the given type, but it can work if implemented with the correct subsets of the original protocol. The programmer can be distracted by using annotations and adapting to the given types that they will lose sight of the original task objectives.

4.7 Programming paradigms

A programming paradigm is a way in which a human can model or base a programming language on, it’s a way to classify languages, just like with human languages we can state which comes from Latin, Anglican etc. it’s not that dissimilar, but of course these are all invented as oppose to evolved languages. A language is not tied to just one paradigm a language can be supported by multiple paradigm Smalltalk falls into only one paradigm but C++, for example, overlaps the paradigms (Krishnamurthi, 2008). The features of that language characterize what paradigm they belong to. There are many different programming paradigms and there are many languages based on that of a programming paradigm. There are many programming paradigms like Imperative, declarative or object orientated. Although there are much more, I intend to concentrate on those I have mentioned.

4.8 What is type safety?

Type safety comes before the language is created as mentioned in the previous section, a well-typed system cannot go wrong theoretically this is the reason type safety is enforced to resolve a poor system. But not all type systems implement type safety this doesn’t mean the language is wrong or bad but having type safety has been proven to limit errors and make a language more protected.

There are many models of type safety and are better shown in examples from well typed too poorly typed systems.

There are many advantages of type safety such as allowing the code to be analyzed for errors whilst the program is running giving you pointers in how to clean up the code you are writing also known as linting (Thoughts on web development, 2015). The main advantage is keeping the language safe minimizing the errors that occur keeping the flow of the program for the programmer.

Java uses mostly dynamic type checking it has objects that are stored in the computer’s memory with class tags as labels, as Java’s type safety enforces the
class tags off the current object to be checked before every operation of that very object \textit{(Order book, 1999)}.

Java enforces type safety this prevents the memory being accessed inappropriately \textit{(Order book, 1999)}. In java, everything is assigned to an object stored in its memory so for example if you had an application with an alarm clock with classes such as date, time or day. Each class has defining objects and operations to be performed with those classes. In the application, an operation would be required to turn on the alarm but you'd neither turn on or off the date it would be pointless. That's why type safety only allows the program to perform operations on an object if the operation is valid to that object \textit{(Order book, 1999)}.

\textbf{Figure 2.0}

Figure 1.0 shows the classes that are implemented in a calendar management application, as you can see each of the classes Alarm and Applet has operations, for instance, the turnOn operation would turn the field to true. Another operation in the class Applet is fileAccessAllowed this allows access to the computer's hard drive. Java enforces type safety to stop the operations being mixed up or incorrectly selected causing problems. On an occasion, the program might apply the turnOn operation (this is what we didn’t want to happen) to the Applet class this would otherwise without the type safety enforced select fileAccessAllowed and turn it to true. This would then allow anybody to access any sensitive data that has been stored on the hard drive being tampering with or deleted \textit{(Order book, 1999)}. Another example of type safety occurs in C# this is to help prevent any objects peering into another object's memory location \textit{(Code Project, 2015)}. Even though C# has fewer levels of type safety than Java it still has type safety in place.
4.9 Breakdown of a type system

Types $t ::= \text{Int} | \text{Bool}$

Expressions $e ::= n | \text{true} | \text{false} | \text{skip} | \text{if e then f else g} | \text{while e do f}$

This is a basic type system with rules and expressions set in place to be followed. This would be implemented into a language, breaking down the type system to understand the complexity of it and how it works and what is required to replace the given values or expressions. Further on we will discuss a more complex type system, as well as the grammar in which types are specified, the following line states that in our type system they can be of type integer or Boolean.

Types $t ::= \text{Int} | \text{Bool}$

The following expressions are inputs such as values, constants, variables, operators, and functions that the language can interpret into the type system, as long the language obeys the rules of the type system the compiler can run the code so that no errors will occur statically or if already run dynamically can compute.

Expressions $e ::= n | \text{true} | \text{false} | \text{skip} | \text{if e then f else g} | \text{while e do f}$

Likewise, this states that in our language the expressions that can be accepted being $n$ and true, but $n = \text{integer}$. With expressions they don’t know what the elements are for example in $e_1 + e_2$ the typing rules take care of this so if $e_1$ was of type integer and $e_2$ was of type Boolean it would not compute. But if $e_1$ and $e_2$ were of type integer they follow the rules.

\[
\text{True} : \text{Bool}
\]

The top line is your assumption and the bottom line is our conclusion, “:” just means that “is of type” telling us that true is of type Boolean, this can be written in a different format but can mean the same thing, like removing the top line which is the assumption and just use the conclusion.
In this equation, it is telling us that if $e_1$ is of type integer and $e_2$ is of type integer then in our conclusion $e_1 + e_2$ will be of type integer and the compiler can run this. But for example, if a programmer tries to add $e_1$ as a type Boolean to $e_2$ is of type Boolean then the compiler will return an error. $\text{True + True = Error}$

In this equation, the assumption is $e_1$ is of type integer and $e_2$ is of type integer and the conclusion is when $e_1$ is not equal to $e_2$ which is of type Boolean.

4.10 Derivation

Derivation works by combining multiple rules that are trying to prove something more complicating than the basic type systems shown above, below the assumption is of type integers so if $e_1, e_2$ added together and $f_1, f_2$ added together are both integers then the conclusions can become Boolean which should be true if the assumption meets the requirements.

Type systems for Booleans and numbers can do arithmetic, they can test for equals and also on if then else loops. More interesting examples can be giving when if includes while loops and commands.
4.11 Advanced type system

**Types**

\[ z ::= \text{Int} | \text{Bool} | \text{Comm} \]

**Expressions**

\[ e ::= \text{n} | \text{true} | \text{false} | e1 + e2 | e1 - e2 | e1 \text{ e2} | e1 \text{ e2} | \text{if e1 then e2 else e3} | \text{skip} | \]

\[ \text{While e do } f \mid e1 \mid e2 \]

This is a more complicated type system introducing more rules and more complex expressions that can run in a programming language. So this type system has a variety of rules including the skip command that just passes nothing, also we have added the \( e1 \mid e2 \), on top of that the “ !=" this just means not equal to as previously mentioned. Throughout a type system, we can apply logical operators as you have already seen for example true and true would equal true the following figure shows the rules for these:

<table>
<thead>
<tr>
<th>True</th>
<th>AND</th>
<th>True</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>AND</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>AND</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>AND</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>OR</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>OR</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>OR</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>OR</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

With a skip command, it can also be written in this format.

\[ \text{Skip} : \text{Comm} \]

\[ e : \text{Bool} \quad f : \text{Comm} \]

\[ \text{While e do } f : \text{Comm} \]

While commands are very important as they can re-iterate loop until the expression is made, so the following code will run until met or if the expression isn’t met the compiler will throw an error. While \( x \) is less than 10 keep running through the code until the requirements are no longer met, in this case, the number would be 11 and the compiler would stop printing \( x \).
While \( x \leq 10 \)

\[
Do (\text{print} \ x) \nonumber
\]

\[
End
\]

For \( e_1; e_2 \) this example would compute to “\( e_1 \) and then \( e_2 \)” so apply the rule to \( e_1 \) and then apply to \( e_2 \). Or do command \( e_1 \); then command \( e_2 \).

Another example: \( \text{Print} \ x_1 - e \ do \ x + 1 - e_2 \)

\[
\frac{e_1 : \text{Comm} \ e_2 : \text{Comm}}{e_1; e_2 : \text{Comm}}
\]

4.12 Subtyping

Subtyping is a form of polymorphism so that in any context if \( t \) is being used then it can be replaced by of type \( s \), the type system of a given language can define its own subtyping relations. Subtyping would be used to inherit the characteristics of its superior so \( s \) can inherit the characteristics of \( t \).

\[
s <: t
\]

These equations are used to show how subtyping works. \( s \) is a subtype of \( t \), \( t \) is defined as the supertype.

\[
s < t
\]

In this equation this symbol \(<\) means the same as \(<:\) but can be written as either, \( s \) is also a subtype of \( t \).

\[
s < s
\]

\( s \) is always a subtype of itself, \(<\) can also be replaced with \(<:\) and it would still mean the same thing.

\[
\frac{p < s \ s < t}{p < t}
\]

if \( p \) is of subtype of \( s \), \( s \) is a subtype of \( t \) for \( p \) is a subtype of \( t \), this is a more complicated equation, but where \( t \) was the super type in the previous equations and
Subtyping is a relationship between different types that can show proof that the subsumption principle is correct. For example, if \( s \) is a subtype of \( T \) wherever a value of type is needed then it can be replaced by the type \( T \).

4.13 Summary

Throughout the analysis and discussion, the sections have explained the different types of type systems showing a breakdown of a basic type system and a slightly more complicated type system giving an in-depth knowledge of the types that the compiler will run. There has been a clear understanding of subtyping and derivation that occur, as well as type safety and the importance of using this decrease type errors.
5.0 Conclusion

On completion of the research project, a conclusion has been drawn upon the lifecycle of the project, the findings, problems and how to build on to the project after it has been completed. But also if further develop what impact could it have in the future.

Throughout this research project, an area of research in type systems has been converted from postgraduate level to undergraduate level to be more understandable for students. Using the journals, articles, and books used in this research project to achieve the title to explain a type system in a language understandable to an undergraduate student. Setting aims to achieve this, understanding the complexity of type systems along with objectives to understand how type systems work and their purpose, understanding type safety and its objective and to analyze and defining a type system. The research project has achieved a clear understanding of the complexity of type systems, educating the question why type systems are important arguing that by using them within a language provides a more secure program when being used. As implementing type safety to make a language strongly typed will limit the errors in the program, giving a complete system in the future.

The research that has been proven in the project can be used in the future by others, progressing on to an advanced type system or even be converted to key stage level 3. Step by step concluding that a dismantled type system is understandable and the complexity is made simpler by showing a clear analysis of type systems.

The problems faced along the way is that many different theorists have many views on diverse theories mentioned, gaining an idea of that was shown to be difficult at times but gave a stronger background of information to prove the title.

From the research undertaken in this project to progress from here being able to show students a clear understanding of how to implement a type system into a compiler, being able to write a successful algorithm that works coherently alongside the compiler. Starting from all the basics of how compilers run and work and then being able to create a type system if proven possible that can be implemented in any language, an adjustable type system that can be used in functional programming, imperative programming or even object orientated programming. It would be almost the perfect type system.
6.0 Appendices

Research Onion model (Research gate, 2017)
When undertaking a research or enterprise project, Cardiff Met staff and students are obliged to complete this form in order that the ethics implications of that project may be considered.

If the project requires ethics approval from an external agency (e.g., NHS), you will not need to seek additional ethics approval from Cardiff Met. You should however complete Part One of this form and attach a copy of your ethics letter(s) of approval in order that your School has a record of the project.

The document Ethics application guidance notes will help you complete this form. It is available from the Cardiff Met website. The School or Unit in which you are based may also have produced some guidance documents, please consult your supervisor or School Ethics Coordinator.

Once you have completed the form, sign the declaration and forward to the appropriate person(s) in your School or Unit.

PLEASE NOTE:
Participant recruitment or data collection MUST NOT commence until ethics approval has been obtained.

PART ONE

<table>
<thead>
<tr>
<th>Name of applicant:</th>
<th>Dylan Wayne Evans</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervisor (if student project):</td>
<td>Anna Calderon</td>
</tr>
<tr>
<td>School / Unit:</td>
<td>School of management</td>
</tr>
<tr>
<td>Student number (if applicable):</td>
<td>ST20039680</td>
</tr>
<tr>
<td>Programme enrolled on (if applicable):</td>
<td>Software Engineering</td>
</tr>
<tr>
<td>Project Title:</td>
<td>If using a working title, it should convey what the project is about</td>
</tr>
<tr>
<td>Expected start date of data collection:</td>
<td>31/10/2016</td>
</tr>
<tr>
<td>Approximate duration of data collection:</td>
<td>3 Months</td>
</tr>
<tr>
<td>Funding Body (if applicable):</td>
<td>Click here to enter text.</td>
</tr>
<tr>
<td>Other researcher(s) working on the project:</td>
<td>If your collaborators are external to Cardiff Met, include details of the organisation they represent.</td>
</tr>
<tr>
<td>Will the study involve NHS patients or staff?</td>
<td>No</td>
</tr>
<tr>
<td>Will the study involve human samples and/or human cell lines?</td>
<td>No</td>
</tr>
</tbody>
</table>

Does your project fall entirely within one of the following categories:
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paper based, involving only documents in the public domain</td>
<td>Yes</td>
</tr>
<tr>
<td>Laboratory based, not involving human participants or human samples</td>
<td>No</td>
</tr>
<tr>
<td>Practice based not involving human participants (e.g., curatorial, practice audit)</td>
<td>No</td>
</tr>
<tr>
<td>Compulsory projects in professional practice (e.g., Initial Teacher Education)</td>
<td>No</td>
</tr>
<tr>
<td>A project for which external approval has been obtained (e.g., NHS)</td>
<td>No</td>
</tr>
</tbody>
</table>

If you have answered YES to any of these questions, expand on your answer in the non-technical summary. No further information regarding your project is required. If you have answered NO to all of these questions, you must complete Part 2 of this form.

In no more than 150 words, give a non-technical summary of the project:
I will be designing a tutorial in type systems for undergraduates. This will address the gap at the level of undergraduate education since the literature is typically at postgraduate level. The tutorial will consist of a basic type system and a more advanced type system with an explanation of how they work and what applies when using a type system covering varying aspects of types, type rules, and terms.

DECLARATION:
I confirm that this project conforms with the Cardiff Met Research Governance Framework.

I confirm that I will abide by the Cardiff Met requirements regarding confidentiality and anonymity when conducting this project.

STUDENTS: I confirm that I will not disclose any information about this project without the prior approval of my supervisor.

Signature of the applicant: D W Evans  Date: 08/11/2016

FOR STUDENT PROJECTS ONLY

Name of supervisor:  Date:

Signature of supervisor:

Research Ethics Committee use only

Decision reached:  
- Project approved
- Project approved in principle
- Decision deferred
- Project not approved
PART TWO

A RESEARCH DESIGN

A1 Will you be using an approved protocol in your project?  No

A2 If yes, please state the name and code of the approved protocol to be used.
Click here to enter text.

A3 Describe the research design to be used in your project
In this section, include details (as appropriate) of:
- research method(s);
- sample and sampling;
- recruitment of participants;
- analytical techniques
If your project does involve the use of an approved protocol, much less detail will be required but you should indicate which areas of the project are covered by the protocol.

A4 Will the project involve deceptive or covert research?  No

A5 If yes, give a rationale for the use of deceptive or covert research.
Click here to enter text.

A6 Will the project have security sensitive implications?  No

A7 If yes, please explain what they are and the measures that are proposed to address them.
Click here to enter text.

B PREVIOUS EXPERIENCE

B1 What previous experience of research involving human participants relevant to this project do you have?
none

B2 Student project only

What previous experience of research involving human participants relevant to this project does your supervisor have?
none

C POTENTIAL RISKS

---

1 An Approved Protocol is one which has been approved by Cardiff Met to be used under supervision of designated members of staff; a list of approved protocols can be found on the Cardiff Met website here.
<table>
<thead>
<tr>
<th>C1 What potential risks do you foresee?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include details of risks to the participants, the researcher and the project as a whole.</td>
</tr>
<tr>
<td>C2 How will you deal with the potential risks?</td>
</tr>
<tr>
<td>Click here to enter text.</td>
</tr>
</tbody>
</table>

When submitting your application you MUST attach a copy of the following:
- All information sheets
- Consent/assent form(s)

An exemplar information sheet and participant consent form are available from the Research section of the Cardiff
7.0 References


