LIBERTY UNIVERSITY SCHOOL OF DIVINITY

Solving the Logical Problem of Evil using the Principles of Programming

Submitted to Dr. Edward Martin, Professor

In partial fulfillment of the requirements for the completion of

THES 689 – A09

Thesis Project Proposal and Research

by

Daniel Edmond

April 14, 2023

Contents

Introduction

Perhaps one of the most challenging and recurring philosophical problems is the logical problem of evil. The impact of the argument is catastrophic if the problem holds true because theism, especially Christianity, fails. Alvin Plantinga offered a solution to the logical problem of evil via the Free Will Defense, proclaiming that with a libertarian view of free will there is no problem at all. The goal of this paper is to outline a model for solving the logical problem of evil that encompasses the libertarian, while offering an alternative to Plantinga's *Transworld depravity*. Using the principles of computer programming, a solution to the logical problem of evil may be formulated that demonstrates that there is no inconsistency in the set of statements:

"1. An omnipotent, omnibenevolent God exists

2. Evil exists"[1]

Thus, the logical problem of evil fails.

[1] J.P. Moreland, and William Lane Craig, *Philosophical Foundations for a Christian Worldview*, 2nd ed. (Downers Grove: InterVarsity Press, 2017), 537.

CHAPTER ONE – PRINCIPLES OF PROGRAMMING

Utilizing the principles of computer programming is not actually about drawing a computer model or writing out code to solve the problem. The principles of programming instead focus on high level concepts in computer programming demonstrate the proposed solution. In order to establish the link, a primer on programming concepts is necessary to define the concepts. Two core principles are defined as part of the proposed solution: *inheritance* and *error handling*. It should be noted that the core principles are based off object-oriented programming and therefore might not exist in certain programming languages. However, the solution provided is not dependent ona specific programming language, but simply borrows the concepts that are common within object-oriented programming.

Classes

Prior to diving into the design principles, it is necessary to describe some of the components that will be utilized in the principles. Object-oriented programming is primarily designed around the programming concept of a class. Jehad Al Dallal and Sandro Morasca state, "the central construct of [object-oriented] development is the class, classes are expected to be high-quality units that can be reused."[2] Understanding the class construct, at a high level, is essential to understanding the principles where the construct is applied.

Lorenz Demey writes, "The definition of a class typically involves a number of variables (sometimes called 'fields') and a number of methods, which capture the class's properties and

[2] Jehad Al Dallal and Sandro Morasca, "Predicting Object-Oriented Class Reuse-Proneness using Internal Quality Attributes," *Empirical Software Engineering* 19, no. 4 (August 2014):775-821.

behavior, respectively."[3] Perhaps a simpler definition is a class is a programming construct that defines a group of variables and methods, related to a common purpose acting as template from which objects can be instantiated. There are additional much more technical aspects of class design such as scope, abstraction, virtual methods, and the like, but those need not be discussed beyond here for the purpose of this paper. Classes are important because they are utilized to create objects. Where classes "are usually compile-time constructs (and thus 'eternal' from the program's perspective), whereas objects may be freely created and destroyed during runtime."[4] This is key, because a class is a set of instructionsthat define the construct that can be created during execution of the program.

Class Parts (Variables and Methods)

In programming, classes have two main elements: *variables* (sometimes called *properties* or *attributes*) and *methods*. Jack Purdum provides a succinct description, "A class is a template used to describe an object. As such, a class is an abstraction, or simplification, of some object you observe in the real world. You can break a class down into two basic components: [1] Properties that describe the object [2] Methods, or actions, that you want to associate with the object."[5] Class variables are used to store a value and methods are used to define actions that can be performed. James McDonough describes the class elements as, "object-oriented classes are composed of *attributes* and the *behaviors* defined for those attributes. Attributes are the data

[3] Lorenz Demey, "*The Porphyrian Tree and Multiple Inheritance*," *Foundations of Science* 23, no. 1 (March 2018): 173-80.

[4] Demey, 173-80.

[5] Jack Purdum, *Beginning Object Oriented Programming with C#* (Birmingham: Wrox, 2012), chap. 2.

fields of a class (constants, variables, etc.), while the behaviors, also known as *methods*, are the actions applicable upon those attributes."[6] An example of a class:

```
class shape
{
	var numOfSides;
	var numOfDegrees;
	var segmentLength;

	method findPerimeter(numOfSides, segmentLength)
	{
		return numOfSides * segmentLength;
	}
}
```

The class above defines three attributes that are the variables *numOfSides*, *numOfDegrees*, *segmentLength*. There is also a behavior, which is the method *findPerimeter*. Since the class acts as a template, there are no values assigned to any variables.

Variables can contain numerous types of data of varying complexity: simple *true* or *false*, *integers*, text *characters* or text *strings* as well as complex types such as *arrays* and other *classes*. Therefore, class variables provide a broad range of flexibility and complexity for storing data. Purdum states, "The *class properties* are the data that you want to associate and record with

[6] James E. McDonough, *Object-Oriented Design with ABAP*: *A Practical Approach* (Pennington, 2017), 12.

an object."[7] The type of data that can be associated is largely unconstrained. It is worth noting that not every variable must have a value assigned for the object at creation; it simply needs a value assigned before the variable is used. Some variables might be used only in rare occasions while others are necessary for using the class at all and must be defined before an object can be created from the class.

Class methods are used to define an action that can be performed. Purdum writes, "Methods are used to describe whatever actions you want to associate with the object. Class methods often are used to manipulate the data contained within the object."[8] It is not necessary that a method modifies the data in an object, as some actions might be simple retrieval of the value stored in a class variable. However, other actions might be exceptionally complex calling multiple methods and utilizing additional classes and variables to achieve the work and modify data. Also note what McDonough highlights, "Methods may be defined with a parameter interface, also known as a *signature*, by which information can be exchanged between the method and its caller."[9] This means a method may have variables passed to it and also return a value to the caller of the method, though neither is required. As a simple example, a class could have a method called *sum* and the method could have two parameters, *value1* and *value2*, and the method simply returns the result of *value1* added with *value2*.

[7] Purdum, chap. 2.

[8] Purdum, chap. 2.

[9] McDonough, 12.

Objects

Classes are critical to programming because they are used to create *objects*. Until an object is *instantiated*, or created in memory while the application is running, the class is simply an idea. James McDonough states, "Creating an object of a class is performed by the program at execution time and is known as creating an *instance* of the class, a process also described by the term *object instantiation*."[10] The distinction between a class and object is critical because the difference between the two is a concept of programming that establishes a base for the rest of the argument. Demey highlights the difference stating, "Classes serve as blueprints for the objects."[11] The idea of a blueprint is accurate and provides a mechanism to build on the previous example. Using the same *Shape* class that was defined above which contains variables for defining the number of sides -*numOfSides*, the number of degrees - *numOfDegrees*, and the lengths of each segment - *segmentLength*. Since the class definition allows for multiple values for the number of sides and degrees for the shape the object that is created from the *Shape* class could be a square, a triangle, a pentagon, or a number of other shapes. Each shape created is based on the values set when object is created when the program is executed. So, the *instance* of the *Shape* class defines the type of shape created based on the values assigned when the object is created from the class. Example:

Class: Shape

Object name: Triangle

Variables assigned: numOfSides = 3, numOfDegrees = 180, segmentLength = 10

[10] McDonough, 12.

[11] Demey, 173-80.

Class: Shape

Object name: Square

Variables assigned: numOfSides = 4, numOfDegrees = 360, segmentLength = 5

In the example above, there are two instances created from the *Shape* class. One instance created is a triangle and the other is a square. The *Shape* class could also have a method that determines the degrees of each angle of the shape called *calculateAngle* which takes the *numOfDegrees* and divides it by the *numOfSides*. Note that all the names of the variables and methods are up to the programmer, but they are normally meant to be descriptive and follow a similar naming convention as shown in the example.

A real-world example is the photo gallery on a phone. Each picture is displayed as a thumbnail, the height and width of each thumbnail is the assigned a value from the application settings, but the value for the image (ie. the image for each thumbnail) is different. The gallery application also has a method which allows the photo to display when the picture is touched. So, the photo gallery demonstrates both attributes of a class and behaviors that the class can perform.

Inheritance

With the central construct of object-oriented programming described at a high level, now the first principle of programming may be defined: *inheritance*. Inheritance is the practice of one class expanding the variables and methods of another class that it is linked to, or child of, building a hierarchy of class association. Sunil Sing, Rajnish Kumar, and Kamal Mehta say the following about inheritance, “INHERITANCE is a powerful mechanism in an [object-oriented] programming. This mechanism supports the class hierarchy design and captures the IS-A

relationship between a super class and its subclasses."[12] This relationship provides a basic type and subtype grouping for classes as well as the objects created from them. D.C. Halbert and P.D. O'Brien expand on this point, stating, "One type may be a subtype of another, with the implication that if B is a subtype of A, an object of type B may be used wherever an object of type A may be used. In other words, objects of type B are also of type A."[13] The purpose of inheritance is to allow the derived class to be expanded to contain new attributes and methods for more specific purposes according to Jozef Udvaros and Miklós Gubán.[14] Since inheritance is a hierarchical progression, it naturally follows a narrowing of focus and specificity as additional classes continually inherit from other classes. Such a hierarchical progression of inheritance might be a class called *AutoMobile* which has another class inherit from that called *Car* which has another class inherit from that called *Toyota*: Automobile -> Car -> Toyota. With each new class inheriting from the previous class, the new class if more narrowly focused.

Critical to inheritance is the passing of instance variables and methods to *derived* or *child* (also called *subclasses*) classes. A *parent class* or *superclass* can define a set of variables and methods, known together as *class members*, that will be passed down to any class that inherits from that class. This allows for more generic member sets to be expressed at higher levels in the hierarchy. Harald Wertz states, "This process of going from general to more specific, by adding classes, is at the core of object-oriented programming. The new class inherits the old, it is a

[12] Sunil Kumar Singh, Rajnish Kumar, and Kamal K. Mehta, "*Software Reusability through Object Oriented Inheritance Tree Metric*," I-Manager's Journal on Software Engineering 3, no. 3 (January 2009): 1-5.

[13] D.C. Halbert, and P.D. O'Brien, "Using Types and Inheritance in Object-Oriented Programming," *IEEE Software* 4., no. 5 (September 1987): 71-79.

[14] Jozef Udvaros, and Miklós Gubán, "Demonstration the Class, Object and Inheritance Concepts by Software," *Acta Didactica Napocensia* 9, no. 1 (2016): 23-34.

subclass, while the former is its superclass. This term superclass is justified by the fact that the set of instance variables of the subclass includes the instance variables of its superclasses."[15] Multiple classes can inherit from the same superclass, with each child class inheriting the members from the parent while providing a unique set of members. A basic example of this may be seen by looking a hierarchy such as motor vehicle (superclass) to car or tractor. While both are motor vehicles, a car and tractor have a vastly different set of members due the different purposes they are built for while they are also both motor vehicles.

Taking the *Shape* class that was used previously, rather than creating objects that are simply defined as a triangle or a square, inheritance could be used to create two new classes instead. A new *Triangle* class and *Quadrilateral* class could be created as children of the *Shape* class. Since they inherit from *Shape*, both the *Triangle* and *Quadrilateral* class would already have the variables *numOfSides*, *numOfDegrees*, and *segmentLength* along with the method *calculateAngle*. The *Triangle* class could now add additional variables such as a variable called *typeOfTriangle* and variables for each angle named *angleA*, *angleB*, and *angleC*. In a similar manner, the *Quadrilateral* class could add variables called *top, bottom,* and *height* as well as variables for each angle: *angleA*, *angleB*, *angleC*, *angleD*. In this manner, instances of the *Triangle* class can now be utilized to create different types of triangles, rather than simply equilateral triangles, which instances of triangles created from the *Shape* class would be constrained to. Likewise, squares, rectangles, parallelograms, and other quadrilaterals can now be instantiated rather than only squares. Now, admittedly this setup as is might not be especially ideal and perhaps the parent class should have been better designed to support inheritance more

[15] Harald Wertz, *Object-Oriented Programming with SmallTalk* (Amsterdam, 2015), 59-65.

efficiently. However, the example will suffice and demonstrates the principle of class inheritance.

To summarize, inheritance is a programmatic hierarchy that allows for specificity and uniqueness to be added to classes, while also passing along a common set of class members to be shared by all subclasses. The values of the variables passed down are not necessarily shared, and instead each instantiated object may assign values to the variables that are contained within the class it is instantiated from along with the variables inherited from any superclass the class is derived from. McDonough captures the essence of inheritance as, "Inheritance is what enables a class to have members that are not defined within the class itself. Instead, a class indicates those other classes from which it inherits members. This allows a class to reuse functionality defined in other classes and gives rise to the concept of a class hierarchy."[16] While there are numerous layers of complexity directly associated with this programming practice, the principle of inheritance is all that is critical to understand going forward.

Exception Handling

The second principle of programming is *exception handling* or sometimes referred to as *error handling*. Rebecca Wirfs-Brock defines an exception as, "An exception condition occurs when an object or component, for some reason, can't fulfill a responsibility. Or, in programming parlance, an exception condition occurs when some piece of code can't continue on its expected path."[17] So, exception handling is designing code that responds to exceptions in a manner that allow the program to continue to execute or to exit in a controlled manner or, as Wirfs-Brock

[16] McDonough, 53-84.

[17] Rebecca J. Wirfs-Brock, ""Toward Exception-Handling Best Practices and Patterns," *IEEE Software* 23, no. 5, (September 2006): 11.

explains, “handling an exception and performing some sensible recovery action, you can put your software back into a known, predictable state.”[18] The ability to handle exceptions is critical to smooth program execution and essential to modern programming languages. Julian Oliveira, et. al state, “exception handling is strongly related to program robustness…a program can indicate that an error occurred by throwing an exception and attempt to recover by handling (“catching”) that exception. If an exception is thrown but not handled, the program may crash.”[19]

Exceptions handling then is necessary to maintain program execution but does not necessarily continue the process where the failure occurred because the condition of the failure has to be resolved in order for execution to continue. Felipe Ebert, Fernando Castor, and Alexander Serebrenik analyzed exception handling and found that there are numerous areas where exceptions can occur, “Errors may stem from application logic-related erroneous conditions, e.g., an invalid bank account number, undetected bugs, e.g., null dereferences and arithmetic overflow, or environmentally triggered erroneous conditions, e.g., impossibility to open a file or communicate via network.”[20] Until the conditions for the exceptions are resolved the error will continue to occur and the process will not be able to complete. This results in necessary code branching. If the exception does not occur, the regular process continues and the normal branch of code would execute as expected. Another example is provided by Simon Doherty, Lindsay Groves, and Victor Luchangco, “suppose a program maintains an invariant that two variables x and y are never equal, and then divides by $x - y$ within a transaction. If that

[18] Wirfs-Brock, 12.

[19] Juliana Oliveira et. al., “*Do android developers neglect error handling? a maintenance-Centric study on the relationship between android abstractions and uncaught exceptions*,” Journal of Systems and Software, no. 136 (2018): 1-18.

[20] Felipe Ebert, Fernando Castor, and Alexander Serebrenik, “An exploratory study on exception handling bugs in Java programs,” *Journal of Systems and Software*, no. 106 (2015): 82-101.

transaction reads values from *x* and *y* that are equal, it will cause a divide-by-zero error."[21] If the difference of *x* and *y* is zero, then there is no quotient to the equation and an error is thrown, creating a need for a different branch of code. B.S. Lerner, et. al. describe the process branching of code, "handling these conditions requires expanding the process beyond the 'happy path.' In real processes, the number and complexity of these exceptional situations are typically quite large, and the need to assure that they are all handled as efficiently and correctly as possible may be quite important."[22] If an exception occurs, the code then branches in response to the type of error, possibly recovering if the exception is resolved or continuing down an alternate execution route. Ebert, Castor, and Serebrenik confirm this stating, "Exception handling mechanisms promote separation of concerns between the normal execution flow of an application and the execution flow in which errors are handled."[23] Handling these various exceptions then results in "additional action is required beyond the normative path" according to Lerner, et. al.[24] One way that programming languages handle exceptions is utilizing *try/catch* blocks of code, where the *try* contains an action to execute, and the proceeding *catch* clause or clauses perform actions if an exception is encountered. The following is a pseudo code example of exception handling:

```
try
{
        connectPhoneToCloudStorage();
}
catch exception A
{
```

[21] Simon Doherty, Lindsay Groves, and Victor Luchangco, et al., "Towards formally specifying and verifying transactional memory," *Formal Aspects of Computing*, no. 25 (2013), 770.

[22] B. S. Lerner, et. al., "Exception Handling Patterns for Process Modeling," *IEEE Transactions on Software Engineering* 36, no. 2, (March 2010): 162-183.

[23] Ebert, Castor, Serebrenik, 82-101.

[24] Lerner, et. al., 162-183.

```
        //code path for exception A
}
catch exception B
{
        //code path for exception B
}
catch exception C
{
        //code path for exception C
}
```

As seen in the example, one single process could have a singular attempt to connect to the cloud storage from the phone could result in different types of failure leading to different code paths. Exception handling provides a way for addressing issues that prevent the normal program execution by branching code to address multiple scenarios and avoid crashes. Code branches provide the ability to maintain program stability and execution even though errors occur that prevent normal execution. Deydatta Kulkarni and Anand Tripathi argue that exception handling should be robust and contextually aware to provide the "ability to adapt and perform tasks based on the ambient context conditions."[25] It is this ability to adapt to the situations and conditions that exemplify the second principle of programming, exception handling.

[25] Devdatta Kulkarni, and Anand Tripathi. "A Framework for Programming Robust Context-Aware Applications," *IEEE Transactions on Software Engineering* 36, no. 2 (March 2010): 184-97.

CHAPTER TWO – THE CROSS-OVER OF PHILOSOPHY AND PROGRAMMING

The high-level introduction to computer programming concepts outlines the areas in which programming and philosophy cross-over, and even supplement each other. While the syntax or expression in which these fields of study utilize to document concepts is different, the logical rules they adhere to, the broad concepts they study, and the goals they exhibit all have intersections. Philosophy and programming have been observed to overlap by others, as Wojciech Tylman states, "Interweaving computer science and philosophy is by no means a novel concept—there is a separate branch of philosophy: the philosophy of computer science. Its scope may be still quite vague—which should not be surprising, considering how young it is, especially when compared to such well-established branches as the philosophy of mathematics."[26]

Since the combination of computer programming principles and philosophy is not new and is beneficial, this chapter will focus on three primary areas: formal logic and programming logic, properties and variables, and possible world scenarios and object instantiation. Here is the focus is demonstrating and defining the similarities between the philosophical and programmatical concepts will allow the programmatical concepts to be utilized in solving philosophical problems.

Formal Logic and Programming Logic

Richard Tieszen argues that formal logic and programming logic supplement each other. He states, "When one teaches logic as logic programming, however, it is possible to approach results about decidability, semi-decidability, and efficiency much more naturally and directly

[26] Wojciech Tylman, "Computer Science and Philosophy: Did Plato Foresee Object-Oriented Programming?" *Foundations of Science* 23, no. 1 (March 2018): 159-72.

than is usually possible in formal logic courses…Teaching logic this way also makes it possible to introduce philosophy students to many important concepts of computer science in the context of logic itself."[27] Formal logic works within a given set of rules. Each part can be represented in a formula form with variables as J.P. Moreland demonstrates.

The parts of the logical equation are binary and conditional; therefore, the statements can be represented programmatically. Most of the comparisons in programming logic are represented with conditional statements such as *if/then* statements. A basic *if/then* is a conditional comparison that says *if* the statement is true *then* perform the next section of code before proceeding to additional actions. Additional complexity can be added such as an *else* clause, which is used to direct the sequence if the preceding *if* clause was false. These conditionals can be nested within each other, allowing for multiple tests of conditions and performing complex comparisons of the values and taking appropriate action in code.

Using the conditional statements in programming, one could rewrite some of the formal logic programmatically to understand how the logic works. For example, the following formal logic could be written:

```
if(a && b) //tests whether a and b are both true
{
        c = true;  //if a and b both are true, then c is true
}
else
{
        c = false; //else clause only executes if either a or b is false, which makes c false
}
```

[27] Richard Tieszen, "Teaching Formal Logic as Logic Programming in Philosophy Departments," *Teaching Philosophy* 15, no. 4 (December 1992): 342, 345.

While it is a basic programming example, it demonstrates that formal logic and programming logic are similar. This allows for the programmatic representation of key philosophical logic. It also provides an introduction to the possible complexity and branching available based on the value of the comparisons.

Properties and Variables

In the discussion of programming and philosophy, the terms "properties" and "variables" have distinct and different meanings. Philosophically speaking there is much debate about properties and not a clear consensus on what properties are. However, Sophie Allen notes that regardless of the disagreements on properties, the debate around properties shows that properties are, "qualitative similarity and difference."[28] This means at a most basic sense, a property is a quality of something. David Hommen provides this example, "properties which are common to all electrons:

• M = having an invariant mass of 9.109×10^{-31} kg

• C = having an electric charge of -1.602×10^{-19} C

• S = having spin s = ½." [29]

While David Hommen uses the example to dive into the debate around properties, the basic example provides a philosophical concept to utilize – properties are a combination of a value and attribute.

Computer programming defines property in three ways: value that determines the class accessibility, value that determines method accessibility, or a synonym for a variable that is part

[28] Sophie Allen, *A Critical Introduction to Properties* (London: Bloomsbury Academic, 2016), 1.

[29] David Hommen, "Kinds as Universals: A Neo-Aristotelian Approach," *Erkenntnis* (February 2019): 4.

of a class.[30] The first two uses in programming are not relevant to the discussion at hand as they are used solely for behavior of architecting the software in a certain manner. The third use is the key component because a property or variable (variable will be the term used for the remainder to draw a clearer distinction between programming and philosophical usage) is assigned a value.[31] So, a variable's *value* then is the "qualitative" aspect of a variable. Variables were described in more detail previously; the important piece here is to demonstrate the similarities between properties and variables and how the concepts interact.

Due to the broad debate in the philosophical realm, it is easier to focus on the similarities of thought and function here rather than embracing a specific philosophical position. Using the example of the electron from Hommen, if M , C, and S are a properties of an electron philosophically, programmatically there could be an electron class with a variables named M, C, and S. Each variable of the class then could be assigned a value such as defined by Hommen's example. The electron class could be instantiated many times over, which means those variables would exist in each instantiation and each instance could contain different values. Moreland and Craig describes a similar philosophical behavior of properties stating, "Properties are ones-in-many; they can be possessed by many concrete particulars at the same time."[32] In that sense, each instance of an object is the concrete particular and the properties are the variables that each instance has, and the values assigned in that instance. This provides a high-level equivalence for discussion and provides the limit for which is needed here. While it is possible to analyze the

[30] Purdum, chap. 2.

[31] Ibid., chap 2.

[32] J.P. Moreland, and William Lane Craig, 188.

programmatic principle further and naturally eliminate certain philosophical perspectives and theories that do not align with it, there is no need for the framework presented.

Ontological Actualization and Object Instantiation

The third concept that is similarly found in philosophy and programming is actualization and instantiation. Modality is perhaps one of the easiest representations of the similar concepts of the programming terminology of object instantiation. In modal logic, the idea is that of "possible world." Each possible world has a set of properties that make it unique. Actualization then is when a possible world becomes a real world. All the properties of that possible world are established and the world exists within the framework that it had before it was actualized. If one of the properties of the possible world was that the sky would be green, when the possible world is realized it would have a green sky. It is the idea of taking what could potentially be and bringing it into existence.

As described previously, a class is a group of attributes and methods that is a framework for how the object will be built. It contains no values as a class; only variables that could be given values should it be instantiated. In this way, a class is similar to a possible world. It contains possibility but not reality. Until the class is instantiated and becomes an object, it is only an outline of possibility. To use the same green sky example, the class might be called *World* and it could have a variable called *skyColor*. That class represents an infinite number of possible worlds with skies of any possible color. Unless the programmer has created a limit to which values are valid for *skyColor*, any color value is able to be assigned. When an object is instantiated from the class, the instance of the object will assign a value to the variable *skyColor* at that time. If the object is instantiated with a value of "green" for *skyColor*, then the newly created instance of the World has green for *skyColor*.

In possible world logic, each possible world might have a different set of properties and some might be present in one world but not another. However, in programming a singular class is representing the world. All the variables will be contained within that one class. Each instantiation of the class will assign different values to those variables. An equivalent to not having a given property in a possible world would be assigning a null value to the variable when the object is created. In that manner, even though it has the variable the variable would contain no meaningful value essentially rendering it the same as not being included. Modal logic has many possible worlds, programming logic has one class with many possibilities.

CHAPTER THREE – REFORMING THE LOGICAL PROBLEM OF EVIL

Before the *logical problem of evil* can be reformed into a singular perspective, the problem of evil must first be defined in the classical sense. Using the classical formulation then will allow a more focused look at some of the newer variations of the problem. William Lane Craig states, "The logical version of the internal problem of evil holds that the two statements (1) An omnipotent, omnibenevolent God exists. and (2) Evil exists. are logically incompatible. For centuries this has been the form usually assumed by the problem of evil. Indeed, as late as the mid-twentieth century atheists such as J. L. Mackie propounded the problem in this form."[33] This version of the problem assumes that the statements do not form a logically consistent set and therefore God does not exist. Michael Almeida affirms the inconsistent set stating, "It's not possible that God is omnipotent, omniscient, and wholly good and that evil exists. Obviously the problem cannot be resolved by appeal to the possibility of God's limited power to actualize a morally perfect world that includes no evil states of affairs."[34] Stephen Law declares that the amount of evil is irrelevant to the problem of evil writing, "Under 'evil' I mean to include both suffering and morally blameworthy actions. The argument then proceeds as follows. Clearly, (2) is true. Therefore, (1) is false. Note that the amount of evil is irrelevant to this version of the argument – all it requires is that there is some, no matter how little."[35] The problem of evil then is about the perceived inconsistency of an all-powerful and all-good God allowing any evil in the world; one cannot simply assert that there is only a little evil relatively speaking in comparison to

[33] William Lane Craig, "A Molinist View," in *God and the Problem of Evil: Five Views,* eds. Chad Meister and James K. Dew Jr. (Westmont: InterVarsity Press, 2017), 41.

[34] Michael Almeida, "The Logical Problem of Evil Regained," *Midwest Studies in Philosophy* 36, no. 1 (2012): 168.

[35] Stephen Law, "The Evil-God Challenge," *Religious Studies* 46, no. 3 (2010): 354.

the amount of good in the world as a solution to the problem. At the root of the problem of evil is that any evil exists. Andrew Pavelich summarizes the problem as, "[If] God is all-knowing, all-powerful, and all-good, then God would know when evil was occurring, would be able to do whatever is needed to prevent evil, and would also want to prevent evil. Since evil exists, then either God does not know about it (God is not all-knowing), or God cannot prevent it (God is not all-powerful), or God does not want to prevent it (God is not good)."[36]

Modifying the Logical Problem of Evil

The logical problem of evil has been modified by various philosophers to add more clarity and strength to the argument. J.L. Mackie breaks the first point into two propositions stating, "In its simplest form the problem is this: God is omnipotent; God is wholly good; and yet evil exists. There seems to be some contradiction between these three propositions, so that if any two of them were true the third would be false. But at the same time all three are essential parts of most theological positions: the theologian, it seems, at once must adhere and cannot consistently adhere to all three."[37] However, Mackie recognizes that there needs to be additional assumptions added to the argument to maintain that it is an inconsistent set. To build inconsistency into the argument, Mackie adds, "These additional principles are that good is opposed to evil, in such a way that a good thing always eliminates evil as far as it can, and that there are no limits to what an omnipotent thing can do. From these it follows that a good omnipotent thing eliminates evil completely, and then the propositions that a good omnipotent

[36] Andrew Pavelich, "The Moral Problem with the Free Will Defense Against the Problem of Evil," *The Heythrop Journal* 60, no. 5 (2019): 678.

[37] J.L. Mackie, *The Problem of Evil: Selected Readings*, 2nd ed. (Notre Dame: University of Notre Dame Press, 2016): 82.

thing exists, and that evil exists, are incompatible."[38] By adding the additional principles, Mackie shifts the problem of evil to include an action that God would take. This allows the Mackie to assert that an inconsistency does exist between an all-powerful and all-good God if evil exists, because all-good God would seek to eliminate evil and an omnipotent God would have the power to do so.

Michael Almeida builds on Mackie's argument to further modify the problem prior to offering his own solution. Almeida's argument modifies Mackie's argument stating, "God is essentially perfectly good. Any solution to the logical problem of evil redux must be consistent with God's perfect power to actualize a morally perfect worlds that include no evil states of affairs and God's perfect goodness in actualizing a possible world."[39] The goal of Almeida is to strengthen Mackie's original argument and then to offer a solution using the impossibility argument. However, the additional piece of Almeida's argument adds an explicit inconsistency in the set, making the logical problem of evil more potent. Therefore, given the additional modifications, an all-powerful and all-good God does not exist if evil exists.

The additional modifications to the argument are necessary because without the additional assumptions and propositions the logical problem of evil is not inconsistent. Therefore, to address the logical problem of evil currently, it is necessary to consider the additions and consider their legitimacy as modifications and whether or not they change the outcome. The modified argument may be stated as:

(1) God is omnipotent

[38] Mackie, 82.

[39] Almeida, 169.

(2) God is omnibenevolent

(3) A good God will eliminate evil as far as it is able

(4) An omnipotent God is able to eliminate evil

(5) God is, necessarily, able to actualize a morally perfect world without evil

(6) Necessarily, God can actualize a morally perfect world that includes no evil states of affairs only if God does actualize a morally perfect world that includes no evil states of affairs

(7) An omnibenevolent and omnipotent God necessarily actualizes a world that includes no evil[40]

(8) Evil exists

From this, (1) and (2) are the claims of the Christian theist. Proposition (3) is Mackie's additional component about the nature of good beings. Based on (1), point (4) is Mackie's expansion of the theist's position on God's power. Numbers (5), (6), and (7), are a summary of Almeida's evolution of Mackie's position, asserting the necessary actualization of a perfect world without evil. Given (1) through (7), (8) appears to be inconsistent in the set. The Christian theist should not have any issue with (1), (2), and (8) as they represent the original argument and are not explicitly inconsistent. Solving the problem then must be focused on demonstrating that at least one of the additional propositions is false or probably false, and therefore the set is not inconsistent.

[40] Almeida, 168.

What is Evil?

Since the problem of evil is centered on the existence of evil in the world, then it is necessary to establish what evil is. Failure to understand the term evil opens up doors to arguments that are not valid under a proper definition of evil. In his analysis of Augustinian thought, Phillip Carry defines evil as, "It is like the ruination in a half-ruined house. The house itself, insofar as it still exists, is a good thing, but the disorder and lack of structure in it is bad. So the evil itself, the ruination or corruption, has no being except as something lacking in the house—the structure and form that it is missing. Thus, we can say evil exists, but it exists the way that a lack or an absence exists. It is real, but only in the way something can really be absent."[41] While Carry's insight is valuable about the nature of evil, it lacks a clear definition. Alvin Plantinga makes a point of establishing two categories of evil, *moral* and *natural*, stating, "And finally we must distinguish between *moral evil* and *natural evil*. The former is evil that results from free human activity; natural evil is any other kind of evil."[42] While the categorization helps create a distinction between types, it fails in defining what evil is and leads to the question asked by Angela Blanco, "What makes a value or an action moral or ethical (its social counterpart)?"[43]

Adam Morton adds an interesting perspective on the definition of evil by drawing a distinction between evil and wrong, "We use the word 'evil' to pick out particular kinds of

---

[41] Phillip Carry, "A Classic View," in *God and the Problem of Evil: Five Views*, ed. Chad Meister and James K. Dew Jr. (InterVarsity Press, 2017), 15.

[42] Alvin Plantinga, *God, Freedom, and Evil* (Grand Rapids: William B. Eerdmans Publishing Company, 1977), 26, Kindle.

[43] Angela Blanco, "Values and Socio-Cultural Practices: Pathways to Moral Development," in *The Oxford Handbook of Culture and Psychology* (May 2012): 4.

wrong actions and their bad consequences. But not all very wrong actions are evil. And some acts have the main features of evil acts though they are not very wrong…This is a subtle business; we have to tease out what distinction we are making when we describe an act as evil rather than just wrong."[44] Richard Kraul agrees that there is a difference between evil and wrong, "Not every case in which someone does something right or goes wrong falls within the domain that is studied by moral philosophy. If, for example, someone is baking bread, and it comes out of the oven burned, then he probably did something wrong."[45]

Evil then is not an entity, is ontologically separable based on the nature of the act, and different than a simple wrong action. To add additional specificity to the definition it will be required to define what is meant by *good*.

Defining "good"

Evil is more clearly defined as that which is the opposite of good. However, the definition of *good* must be clarified then so that evil can be ascertained. Therefore, defining *good* is critical to understanding *evil*, and understanding *evil* is critical to the *problem of evil*.

Defining good is often used in arguments against *Euthyphro's Dilemma*. Rather than accepting some possible existence of an external standard for defining goodness, goodness is defined based on the attributes that God possesses. William Alston states, "Lovingness is good (is a good-making feature, that on which goodness is supervenient) not because of the Platonic existence of a general principle or fact to the effect that lovingness is good, but because God, the supreme standard of goodness, is loving. Goodness supervenes on every feature of God, not

[44] Adam Morton, *On Evil* (London: Taylor and Francis Group, 2004), 9.

[45] Richard Kraul, *What Is Good and Why: The Ethics of Well-Being* (Cambridge: Harvard University Press, 2007), 24.

because some general principles are true but just because they are features of God."[46] The position Alston establishes is that goodness is not a quality that exists, but a semantic term used to describe the group of attributes which God possesses. In his critique of William Alston's position, Jeremy Koons summarizes the position in this way,

> "Alston's particularist model of goodness…can also be constructed with respect to God, goodness, and the virtues:
> (3) These particular virtues (lovingness, mercy, etc.) are good because God possesses these particular virtues.
> (4) God is good because God possesses these particular virtues (lovingness, mercy, etc.)
>
> …with particularism, the order of explanation goes in a particular direction: from the exemplar toward the traits the exemplar is established to exemplify. The order of explanation does not reverse; if it does, you are not a particularist. This strongly suggests that if we construe God as a particularist paradigm, as Alston intends, we should likewise find (3) to be true and (4) to be false."[47]

This summation of the particularist position is quite accurate by Koons. The argument is not that God is good because of His attributes, but rather that particular virtues are good because those values are attributes of God. William Lane Craig agrees and outlines the definition of good as, "God's character is definitive of moral goodness; it serves as the paradigm of moral goodness. Thus, the morally good/bad is determined by reference to God's nature; the morally right/wrong is determined by reference to his will."[48] Therefore, to establish what is good is simply to establish the attributes and will of God. Or it could be stated as Koons forms the particularist view, "These traits have no independent power to impart goodness on something. If there were

---

[46] William Alston, "What Euthyphro Should Have Said," in *Philosophy of Religion: A Reader and Guide*, edited by William Lane Craig (Edinburgh: Edinburgh University Press, 2001): 292.

[47] Jeremy Koons, "Can God's Goodness Save the Divine Command Theory from Euthyphro?" *European Journal for Philosophy of Religion* (Spring 2012): 184-85.

[48] William Lane Craig, *Reasonable Faith*, 3rd ed. (Wheaton: Crossway Books, 2008), 181, Kindle.

no God, and someone were loving, merciful, and so forth, then that person (on the Adams/Alston view) would not be good."[49]

Part of the misunderstanding of goodness is treating it as an action instead of a measurement of the action it is associated or related with. J.L. Schellenberg falls into the misunderstanding of goodness and incorrectly states that theist's position as, "Prior to creation there is no evil in God of any kind."[50] In the particularist view evil cannot exist in God because it is not an entity or thing; instead goodness is measured as the standard of values that God possesses and evil is not a value. Good, and by the inverse relationship evil, are therefore qualities of actions applied to a standard. James Crenshaw shows that atheists demonstrate this misunderstanding when they argue, "A convenient answer to the presence of evil in the world was sacrificed for ethical monotheism. As long as there were multiple deities, evil could readily be attributed to one or several of them with little harm to the total worldview. The emergence of belief in only one god who is both good and powerful brought an attendant problem: explaining evil."[51] Having a single God as a standard provides a simpler explanation of good and evil when good is understood to be references to a standard. Koons confirms this stating, "An action is not good *simpliciter*; it is good *because* it represents an act of charity, or a repaying of a debt, or something else. It is good in virtue of something else. Similar comments apply to the goodness of

[49] Koons, 187.

[50] J.L. Schellenberg, "A New Logical Problem of Evil Revisited*," Faith and Philosophy* 35, no. 4 (October 2018): 464 – 72.

[51] James Crenshaw, *Defending God: Biblical Responses to the Problem of Evil,* (Oxford, 2005), 10.

agents."[52] A person cannot simply do *good*; instead a person performs certain actions, and those actions are categorized as good due to a standard applied to the action.

Because God's attributes are the standard which are called *good*, the will of God, or the commands of God, are *good* by extension of God's attributes. Hugo Meynell incorrectly states, "For the theist, to be sure, what is *good* and *right* is ultimately dependent on the will of God, and immediately on nothing which is not ultimately determined by that will."[53] The misunderstanding of good is that it does not understand that God's will is considered good because of nature from which the will manifests. Therefore, God's commands are not arbitrary due to God's will being explicitly tied to God's attributes. As Craig states, God's will or commandments are, "reflections of his own character…As necessary expressions of his nature, God's commands are not arbitrary."[54] In this manner, what makes something good is the alignment of such an action to the will and nature (i.e., attributes) of God.

Here though there needs to be an additional clarification. When the statement, "God is good" is made, it does not reflect a standard applied to God. Instead, when it is stated that God is good, it means God is the standard for goodness. As a standard, it follows that God possesses the perfect amount of each of His attributes otherwise some other entity would necessarily be the standard This perfection then represents the maximum value, such as a meter stick necessarily represents the maximum distance that can be established as one meter. It could be stated that God is maximally good, or that God contains the full value of each attribute. This means that

---

[52] Koons, 181.

[53] Hugo Meynell, "The Euthyphro Dilemma," *Proceedings of the Aristotelian Society, Supplementary Volumes* 46 (1972): 228.

[54] Craig, *Reasonable Faith*, 181 - 182.

something can be good, in that it aligns to God's attributes but that it might not be maximally good. To use a trivial and imperfect example, it could be said that providing a glass with some water is good for a person thirsty in a desert, but a glass filled completely with water is maximally good. So, while there is a binary distinction between something being good or evil, there is a degree which the goodness or evilness of an action can align. From the same example of the glass of water for the thirsty person, goodness could have varying degrees based on the percentage the glass is filled with water. While it is maximally good to provide a full glass of water, it is good to provide some water as opposed to none at all.

The question here is not the size of the glass, but the quantity of the contents inside compared to the limit of the glass. A standard represents the limit of the unit and acts a reference point to measure against. Perhaps somebody would choose to argue that a person could be kinder than God or possess a greater value of kindness. The argument here is not what is possible for an individual, but what is defined as "good." If whatever value God possesses for an attribute is the maximum, that is the limit of one unit of that attribute. Maybe somebody wants to argue that a person could contain multiple units of that attribute. Multiple meters does not change the maximum value for a meter; if multiple units of the attributes of God are possible, it does not change the standard for measuring.

Free will's role

One more critical point for goodness that is used in several defenses and must be mentioned is that moral goodness implies free will. In Alvin Plantinga's *free will defense*, Plantinga defines free will as, "If a person is free with respect to a given action, then he is free to perform that action and free to refrain from performing it; no antecedent conditions and/or causal

laws determine that he will perform the action, or that he won't. It is within his power, at the time in question, to take or perform the action and within his power to refrain from it."[55] Free will is a heavily debated topic among philosophers, but it is critical to morality because it plays into what moral actions are and the possible worlds which God could and would actualize. One example can show the issue by building on R. M. Hare's example that show good is multi-realizable. Hare states, "Suppose that we say 'St. Francis was a good man'. It is logically impossible to say this and to maintain at the same time that there might have been another man placed in precisely the same circumstances as St. Francis, and who behaved in them in exactly the same way, but who differed from St. Francis in this respect only, that he was not a good man."[56] However, Plantinga's definition for free will is critical here because the statement by Hare could not be modified to include a robot. If a robot was programmed to perform all the same actions of St. Francis, would the robot be *morally good*? The answer is *no* within a libertarian free will perspective because the robot was programmed to perform those actions, or in other words, the robot was predetermined to perform those actions. A robot is a machine that executes a computer program, where the computer program is a set of instructions and the machine is the hardware on which the instructions are executed. Federico Gobbo and Marco Benini describe programming as, "Traditionally, programs are thought of as formal description of a coordinated set of algorithms in a specific language, in order to solve a well-defined

[55] Plantinga, 26.

[56] R.M. Hare, *The Language of Morals* (Oxford: Oxford University Press, 1963), 145.

problem."[57] So, a robot that is executing a programming that models the goodness of St. Francis is not morally good because the actions are algorithmically predetermined.

Another high-level and simplified look at the position follows to outline the assumption of free will using modern devices. Amazon's Alexa artificial intelligence (AI) has a command named "Simon says." When a user commands, "Alexa, Simon says," the AI will repeat the statement that follows the command "Simon says." If a user commands, "Alexa, Simon says, I love you" then the Alexa AI will state, "I love you." The AI has processed the command, executed the program, and output the result which is repeating to the user the words that follow the keywords, "Simon says." But even though the Alexa AI has said "I love you" to the user, is it true that the Alexa AI loves the user? Of course not, because the AI responded to commands based on a computer program. The behavior or the words of the AI carry no meaning because of the process that produced them. If one person says to another person, "I love you" and the same response is returned to the person, there is weight to the response. Why? Because the response, presumably, was ascertained freely, without a predetermined action. This view of free will is a libertarian view. Libertarian views of free will have various definitions, but the classical view is described by David Palmer as "the view that people sometimes act freely."[58] Naturally the study of the *philosophy of mind* touches the area of consciousness and will and there are also numerous arguments for free will which expand into many areas. Rather than provide an analysis of the various positions and responses to various philosophical arguments on free will, or argue about

---

[57] Federico Gobbo and Marco Benini, "Why Zombies Cannot Write Significant Source Code: The Knowledge Game and the Art of computer programming," *Journal of Experimental & Theoretical Artificial Intelligence* 27, no. 1 (2015): 41.

[58] David Palmer, editor, *Libertarian Free Will: Contemporary Debates* (Oxford: Oxford Scholarship Online, 2015), 4.

consciousness, the simple example of the AI is to provide the distinction between an act that is *morally good* and an act that is not – the difference is a moral action requires the ability to choose from a libertarianism perspective and not simply in the action itself. The goal here is not to offer a defense of the position, but to provide a basic description of the assumption that is directly implied in the category of *morally good* actions that is used in this paper.

Defining "evil"

Clearly, if good is simply a term assigned to the group of attributes that God possesses then evil is that which opposes the attributes of God. Craig states, "God is essentially compassionate, fair, kind, impartial, and so forth."[59] If those are indeed attributes of God, then to be unkind or unfair would be evil. The actions are evil because they are against the nature of God. Naturally, the conclusion of such a position is that which Augustine makes, summarized by Nonna Harrison and David Hunter as, "God can never do evil in any way."[60] Augustine's explanation of why God cannot do is evil is different than the position here, but the conclusion is exactly right. God cannot perform evil simply because God cannot act against His nature or command an action that He does not will. Morally evil actions require a free choice to perform such an action. Using the robot example, a robot that kills people is not performing evil because the robot is simply executing the program. The programmer that created the program that the robot executed is the agent that performed the evil because the programmer used the robot as a tool to perform murder.

---

[59] Craig, *Reasonable Faith*, 181.

[60] Harrison, Nonna Verna, and Hunter, David G., eds., *Suffering and Evil in Early Christian Thought (Holy Cross Studies in Patristic Theology and History)* (Grand Rapids: Baker Academic, 2016), 119.

Reforming the Logical Problem of Evil

Given the additional clarification about the definition of *good* and *evil*, the logical problem of evil needs to be changed. Since goodness is not a property of God, but rather that term used to describe the set of attributes of God, the propositions need to reflect the position. For example, omnibenevolence implies God is all-good according to a standard that exists external to God, but goodness is an attribute of God and not a qualifier for God's actions. Though the idea of omnibenevolence is not completely lost, since it means God possesses the maximum values of each of His traits, it means that it is not an aspect of God to consider in the actions. Neither of those two propositions reflect what is meant by *good*. Therefore, (2) should be rewritten as "God is the standard of *goodness*." Numbers (3) through (8) all needs to be adjusted to remove references to good and to insert the more detailed definition of evil. The modified argument is:

(1) God is omnipotent

(2) God is the standard of *goodness*

(3) God will eliminate that which is opposed to His will and nature as far as He is able

(4) An omnipotent God is able to eliminate that which is opposed to His will and nature

(5) God is, necessarily, able to actualize a morally perfect world where actions against God's nature and will are not possible

(6) Necessarily, God can actualize a morally perfect world that includes no states of affairs opposed to God's nature or will, only if God does actualize a morally perfect world that includes no states of affairs opposed to His nature or will

(7) An omnipotent God necessarily actualizes a world that includes no actions opposed to His will or nature

(8) Actions opposed to God's will and nature exist

The purpose of the rewrite, while repetitive, demonstrates the slight shift in the arguments. All the propositions have shifted to the reworked definition of evil for additional clarity, to remove the nebulous concept of evil; rather than treating evil like an entity or object, using the definition adds detail to what the argument expresses.

CHAPTER FOUR – SOLVING THE PROBLEM OF EVIL

The reworked problem of evil, given the explanation of goodness, weakens the logical problem. However, the point of the rework is to provide clarity to what the actual claim is, not to argue against a specific position on the argument. Koons suggests that "good" is often used with a "thin" concept and issues this reminder, "We cannot use these thin concepts without remembering that at bottom they are paper currency whose value depends on a reserve of thick virtues."[61] By using a thicker definition in the argument, the solution will target the root issue instead of engaging in solely abstract analysis.

Principles of Programming Model

The principles of programming established some key models which may be utilized to solve the proposed problem. It must be understood that the claim is not that the world is a program, as this argument is not meant to make that claim. Instead, programming is a method of modeling reality in code, and therefore utilizing the techniques of object-oriented programming allow an opportunity to view a model of reality with a set of understandable rules. József Udvaros and Miklós Gubán state that object-oriented programming "is closer to reality" than other programming techniques.[62] Establishing the model requires defining what a class modeled on God might contain and the understanding the impact that inheritance might have on classes that inherit from the God class. Additionally, the model includes error handling to look at how programming principles include mechanisms for addressing possible failures.

---

[61] Koons, 181-182.

[62] József Udvaros and Miklós Gubán, "Demonstration the Class, Object And Inheritance Concepts By Software," *Acta Didactica Napocensia*,9, 1 (November 2016), 23.

Defining the God Class

The heart of the principles of programming solution depends on the class definition of God. Ultimately, this class is based on the assumed instantiated God of this universe in a theistic worldview and creating the class definition from there. Worth reiterating here is that if a different God is instantiated in the universe, then a different class for God must be defined. Two aspects of the class need defined: the variables and the methods.

Variables

For the purpose of the solution here, the focus will be solely on the relevant moral variables. The first set of variables that could be defined are those that are "omni" attributes of God. Each attribute that God contains though can be manifest in other objects, but not the same degree. So, if God is "omniscient" He simply is present with the maximal value for *knowledge*. Since the value assigned to the variables is provided when the object is instantiated, God can simply contain the variable *knowledge*. A second variable of *power* can be designated, since the argument establishes that God is omnipotent in (1).

Another variable assigned to the God class might be *canChange*, which is could be a Boolean (*true* or *false*). Despite there being multiple views on the immutability of God, Richard Swinburne notes, "Theists have normally claimed that God is immutable, that he cannot change."[63] This statement by Swinburne is echoed by theologians of various beliefs and does not consider the nuance of weak immutability versus strong immutability. While Swinburn and others might want to debate the details of God's immutability, the summary that God is

[63] Richard Swinburne, *The Coherence of Theism,* 2nd Edition (Oxford: Oxford Scholarship Online, 2016), 200.

immutable is accepted in some form by theists is what matters – not the form that is accepted. Withholding all the details of the theological debate allows the concept of immutability to be incorporated into the class definition in a basic and ambiguous way. God's omnipresence is reflection of a lack of special restriction –Walter Kaiser defines the theist view of omnipresence as, "God is ontologically present everywhere."[64] A Boolean could also be used to represent omnipresence, *spaciallyRestricted* (the variable names are irrelevant; they are named this way to reflect what they represent and are sufficient for the class definition). This provides a basic class that contains the attributes of God. A pseudocode example of the class would be:

```
class God
{
        var knowledge;
        var power;
        bool canChange;
        bool spaciallyRestricted;
}
```

If the class God was instantiated with maximum values for *knowledge* and *power*, while also setting *canChange* and *spaciallyRestricted* to false, the object would have the maximum value for knowledge and power, be unable to change, and not be restricted to a single location. Therefore, the object instantiated with those values would contain the qualities of the theist's view of God.

Since goodness is measured by God the standard of goodness must be built into the class as well. A set of variables that are used to establish what is good is necessary to the God class. The set does not have to be comprehensive for the model to work; it must only be sufficient and

[64] Walter C. Kaiser, Jr., *The Majesty of God in the Old Testament: A Guide for Preaching and Teaching* (Grand Rapids: Baker Academic, 2007), chap. 7.

therefore if someone wishes to argue that additional variables would need to be included, it does not impact the model.

There are several attributes that might be considered part of the standard of goodness. William Lane Craig states that God's nature is "just and loving."[65] Justice and love could be variable represented as simply *just* and *love*. Mercy and kindness could simply be represented as *mercy* and *kind*. For all of these, it could be assumed there is a large range of values but for the instantiation of the object, all four of these will have the maximum value assigned. These four variables represent the standard of goodness; they are moral attributes with which actions are compared to.

Methods

There are some additional variables that could be added that might be necessary for some of the methods it can perform. At least two actions could be assigned to the God class, since God possesses the ability to give and He has the ability create. Giving requires some object that is given and an object to receive the gift. To create takes knowledge, power, and vision or a goal. This means that additional variables of *objectToGive*, *recipientOfGift*, and *vision* are necessary variables for the methods to work. The class methods have the following pseudocode:

```
object create(object vision)
{
        object Creation newCreation = new Creation(vision, knowledge, power);
        return newCreation;
}

void giveGift(object objectToGive, object recipientOfGift)
{
        var valueOfGift = (power + knowledge + love) / kindness;
```

[65] Craig, *Reasonable Faith*, 181.

```
        objectToGive.value(valueOfGift);
        recipientOfGift.receive(objectToGive);
        return;
}
```

The pseudocode calls additional methods that are not defined here and additional objects. *Creation* would be a separate class with its own variables and methods, in the *create* method it is instantiated using the *vision*, *knowledge*, and *power* that are passed as method parameters using the values that are assigned to those variables in the instantiated *God* object. The create method returns an object, *newCreation*, which is an instance of the *Creation* class which is not defined here.

The *giveGift* method has two parameters passed in. Those objects also represent additional classes that are not defined here. Purposely ambiguous, the pseudocode demonstrates that an object that is the recipient of a gift would have to *receive* the gift for it to be assigned to the object. The value of the gift would be based on the values of *power*, *knowledge*, *love*, and *kindness*. As a void method, it simply executes without returning any value.

These two pseudocode methods demonstrate that the God class has multiple actions to perform, and those methods have additional variables that are necessary for execution. This is purely for simplification to demonstrate the principles (meaning this code is not executable and ignores many complexities of programing) when there are additional methods and actions that could be added.

As such, the God class has attributes, that when instantiated with a maximum value correspond the "omni" attributes of God. The class also contains moral values and a couple of simple methods to perform actions with. Ignoring programming scoping restrictions, this means

that a class that inherits from the God class will also have the same attributes, same moral values, and the same actions that it can perform. The complete class definition is:

```
class God
{
        var knowledge;
        var power;
        bool canChange;
        bool spaciallyRestricted;

        var just;
        var love;
        var mercy;
        var kindness;

        object create(object vision)
        {
                object Creation newCreation = new Creation(vision, knowledge, power);
                return newCreation;
        }

        void giveGift(object objectToGive, object recipientOfGift)
        {
                var valueOfGift = (power + knowledge + love) / kindness;
                objectToGive.value(valueOfGift);
                recipientOfGift.receive(objectToGive);
                return;
        }

}
```

Defining the Human Class

The *Human* class is a child of the *God* class. Using the principal of inheritance, the class will obtain the variables and methods of the *God* class. Without any additional unique qualities added, the *Human* class at a minimum looks the same as the *God* class. Now, as it stands it would not make sense to do this programmatically since both classes would have the same definition and would be identical. Therefore, two additional variables could be assigned to the

Human class, one called *anger* and another called *empathy*. However, a class definition for humans would also likely contain variables to represent physical values, such as hair color. Physical values would not be found in the *God* class, but for the simplicity of demonstration the *Human* class can remain a class that only defers from the *God* class slightly. Due to inheritance, the *Human* class also contains *knowledge*, *power*, *canChange*, *spaciallyRestricted*, *just*, *love*, *mercy*, and *kindness* as variables.

The Human class contains the two methods it inherited – *create* and *giveGift*. An additional method could be added that is called *modifyKindness.* This new method would perform an internal action based on the value of *anger* and *empathy*. If *anger* increases, *empathy* is decreased and if *empathy* is decreased *kindness* is decreased. The inverse operation would also be contained in the method, where if *anger* is decreased then *empathy* is increased and if *empathy* is increased then *kindness* is increased up to but not greater than the maximum value allowed for the *Human* class.

The final form of the human class has only two variables and one method. The additional variables and methods are inherited from the God class and therefore not explicitly defined within the class. A *modifyKindness* method is defined, that accepts a parameter and then assigns that value to a new variable to be used in comparisons. The calculation is the sum of *anger* and *value*, which would be calculated and assigned elsewhere. Based on the result, the method modifies the values of both the human exclusive variables *empathy* and *anger*, and also the value for the inherited variable *kindness*.

```
class Human : God
{
        var anger;
        var empathy;
```

```
        void modifyKindness(var value)
        {
                var newAngerValue = anger + value;

                if(newAngerValue > anger)
                {
                        anger = anger + 1;
                        empathy = empathy – 1;
                        kindness = kindness – 1;
                }
                if(newAngerValue < anger)
                {
                        anger = anger – 1;
                        empathy = empathy + 1;
                        kindness = kindness + 1;
                }

        }

}
```

Instantiated God and Human

Creating the instances based on the class definitions is where the programming principals establish an understanding of the fundamental principles of the moral relationship of God and creation. When God is instantiated with maximum values for *knowledge* and *power*, then God has the attributes of omniscience and omnipotence. Assigning the *canChange* variable a value of false and *spaciallyRestricted* to false allows God to be immutable and omnipresent. The variables that represent the moral values of God can all be assigned the maximum value available. The resulting instantiation of God contains the properties of the theistic God.

Instantiating the Human object means the values from God are inherited and need a value. Due to the physical limitations of humans, the values assigned cannot be maximum. A new human would have extremely limited knowledge and power, so a value of 1 makes sense for as an initial value for both *knowledge* and *power*. Humans are physical beings so *canChange* and

spaciallyRestricted would both be true. The two Human specific variables of *anger* and *empathy* are assigned 0 and 100 respectively.

The assigned values for the moral variables can be complete or the maximum value allowed for a physical being. The new Human object could be instantiated with 100 for *just*, *love*, *mercy*, and *kind* – the moral attributes inherited from the God class. Given these values for the object the human is perfect. It contains a maximum value capable of a physical being for each moral variable associated with the God class and an extremely limited knowledge and power value.

Assessing the Truth of (3) and (4)

(3) God will eliminate that which is opposed to His will and nature as far as He is able

(4) An omnipotent God is able to eliminate that which is opposed to His will and nature

(3) is an interesting point because on the surface the statement does not necessarily appear to be true. Perhaps this is because Mackie's original argument uses "good" as an attribute rather than as a standard that points to other attributes. However, assuming that this statement is true as is, then it appears the point of contention is "as far as He is able." (4) builds on (3) noting that omnipotence grants the power to be able to eliminate evil. The assumption then of (3) and (4) is that omniscience grants a knowledge of, and omnipotence grants full power to prevent evil. While (3) is questionable at face value, the addition of (4) becomes problematic due the omnipotence assigned to God.

As some philosophers have noted previously, omnipotence does not mean God can perform every action; there are limitations. William Lane Craig states, "God's being all-

powerful does not mean that He can bring about the logically impossible."[66] The libertarian view of free will has a point of contention there where free will is of supreme value to God. Alvin Plantinga argues that free will is something that God values, "A world containing creatures who are significantly free (and freely perform more good than evil actions) is more valuable, all else being equal, than a world containing no free creatures."[67] Therefore, even an omnipotent God might not be able to prevent evil because God could value free will to point of refusing to intervene.

Exception handling provides another type of explanation. This allows the program to execute given certain situations happen during the execution of the program. These could be explicit exceptions that would cause failures or conditions that would lead to failures elsewhere if not properly addressed. Simon Doherty, Lindsay Groves, and Victor Luchangco provide the divide by zero example, where a program simply divides x by y as long as the values are different.[68] It is possible that y could be 0 and therefore the equation cannot be evaluated. Programming principles suggest that exception handling be written to address those types of situations either through a conditional check of the y value prior to performing the operation or an exception code branch based on the resulting divide by zero error. Since 0 could be a perfectly valid value for y, it means that it encountering the condition does not mean a mistake happened anywhere else. If y was based on the amount of money remaining in the bank account, then the value is derived from another source. A value of 0 would be precisely the correct value if the

---

[66] William Lane Craig, *On Guard: Defending Your Faith with Reason and Precision* (Colorado Springs, 2010), chap. 6, Kindle.

[67] Alvin Plantinga, *Good, Freedom, and Evil* (Grand Rapids, 1974), 335, Kindle.

[68] Simon Doherty, Lindsay Groves, and Victor Luchangco, et al., "Towards formally specifying and verifying transactional memory," *Formal Aspects of Computing*, no. 25 (2013): 770.

bank account had a balance of 0 as well. The dependency is valid, but due to the dependency there is an opportunity for the necessary operations to encounter errors, in this case the divide by zero error.

It could be stated then that if any values assigned to the *Human* class have dependencies that are used to derive their value, then an exception could be encountered, and a different code path taken. Using the same primitive Human class, if the human stubbed his toe on the log 100 times, then the value for *anger* would be 100, *empathy* would be 0, and *kind* would be 0. Though clumsy, the human has committed no evil and had no evil performed against him. The values have been modified and the probability of evil increased without evil being committed.

The impact of the value change can be demonstrated in the methods that can be executed from the *Human* class. Inherited from the God class was the method *giveGift* which calculates the value of the gift with *kind* as denominator in the equation. Since the method as written has no exception handling, an error would be encountered if the human stubbed his toe 100 times and then tried to give a gift. Exception handling would be required to allow the program to function properly and continue forward. It would be necessary to include exception handling, not due to bad design, but due to dependencies on dynamic values.

It would be entirely possible that there are situations where intervening without overriding the free will of an individual performing an act evil against an innocent victim has too many variables and dependencies to intervene without breaking the system. This type of perspective might appear similar to the greater good idea that Schellenberg states as, "any good for which evil is required or must be permitted an evil-involving good."[69] However, the argument here is not that God has a plan which evil becomes a necessary part to fulfill some

[69] Schellenberg, 465.

greater good. The argument here is that God could necessarily impact other areas through intervention, due to dependencies, which would result in more issues. Creating a banking program that never allowed an account to have a balance of zero would prove unsustainable as the bank would allow spending of money the customers did not have. God too, must allow certain values and actions to occur even though the impact or potential impact might result in negative outcomes which might lead to evil. (3) is likely false, since there exists the possibility that God is not able to eliminate evil due to the dynamic system that exists. If God secretly modified the value of a zero balance to have funds, the change would need to also have a corresponding transaction in the system. That transaction would have to have a corresponding source. The source of funds would have to have some physical reality (a business, a person, another bank account) and some funds. Essentially, a single value change creates a necessary stream of subsequent changes in order for the change to not have an impact on the system. A trivial change of the value of balance leads to a complex set of additional changes. Given the human system and interactions with others, the complexity of such interventions would require additional necessary modifications too. It would not be sufficient to only modify a single value, it would require a system of changes that might ultimately lead to values that are unchangeable without a substantial impact on the whole system. Exception handling instead is a pro-active approach to addressing such situations and ensuring the system progresses in a proper and controlled manner, while handling those scenarios. The knowledge and power of God then would be completely compatible with (3) while still having (4) as false.

In this example, the inability to prevent a bad outcome is not the result of a lack of power. The power to address such outcomes and foresee the possible failures results in exception handling to allow continual execution. If God, even with omnipotence and omniscience, has

created a system with dependencies and consequences are derived from activities outside of the autonomous human objects, then it is entirely possible that there are scenarios that God is not able to prevent. In a dynamic system, where objects interact and those interactions modify the mutable values of the objects, it is entirely possible that God would be restricted to handling the possible outcomes rather than preventing the possible outcomes.

Assessing the Truth of (5)

(5) God is, necessarily, able to actualize a morally perfect world where actions against God's nature and will are not possible

Given the state of affairs as defined in the two objects that were instantiated, (5) is true. However, this appears to be only true given a condition that the world is continually a static state of affairs. The values assigned to variables for these objects are not required to be static and can be changed based on various code paths. Since humans are able to learn and acquire new knowledge, the value for *knowledge* could increase. Humans can also acquire new skills, naturally gain more strength as they age, therefore the value assigned to *power* could change.

Since the values are dynamic, it also means there could be factors that reduce the values of the moral variables. The instance of Human has *canChange* set to *true*. A mutable object then could be instantiated perfect and through the vicissitudes of the program result in an object with completely different values than it was instantiated with. This means that given the ability to change it stands that the moral values could be modified, and the result could be less than perfect. As values are reduced, the probability of committing evil increases; the more a moral value depletes the less impact a moral value has.

Then the question must shift to, "What could reduce the moral value?" Here the answer is two primary grouping of sources: external and internal. Natural experience could modify a value,

for example tripping over a log could result in an increase in the *anger* value, moving the value from 0 to 1. Since the human has a *modifyKindness* method which ultimately decreases the value of *kind* if the value of *anger* is increased, the external and non-evil event had an impact on the human object. It does not make the human evil nor does it mean the natural event was evil, but the impact of the event changes the values of the object allowing for the possibility of choosing evil.

Obviously, the argument cannot intend for the static condition to be applied to (5). The question of (5) is can it be true given a dynamic state of affairs. This seems less obviously true. If having values of 100 for all the moral values results in always choosing "good" and having 0 for the values results in always choosing "evil", then if the value is reduced by even 1, there is the possibility of actions against God's nature and will. The point here that is that the ability to perform evil is directly correlated with the ability to change. A world that allows dynamic moral agents to interact with each other and the environment introduces the opportunity for evil.

Assessing the Validity of (6) and (7)

(6) Necessarily, God can actualize a morally perfect world that includes no states of affairs opposed to God's nature or will, only if God does actualize a morally perfect world that includes no states of affairs opposed to His nature or will

(7) An omnipotent God necessarily actualizes a world that includes no actions opposed to His will or nature

(6) and (7) are connected. The summary of the two points is that if God has the power to create a world without evil then God would necessarily create that world; thus, if there is evil in the world the God must not be omnipotent. Again, (6) and (7) must be referencing a dynamic

world because the premises are targeted at the actualized world. Since (5) is false, it immediately calls into question the validity of the assumption that God could "actualize a morally perfect world that includes not states of affairs opposed to His nature or will." If (5) is false, then (6) and (7) are also false. -

The weakness of (6) and (7) is that they do not progress the argument on their own. A dependency on (3), (4), and (5) exists in order to force an action of God "necessarily." Without (3), (4), and (5) there is no need to accept (6) and (7). It has already been demonstrated that (3) could be assumed true and still (4) is false, as well as (5) being false. This leaves (6) and (7) without the necessary support to stand.

In each premise, the underlying and unstated belief appears to be that God is able to and willing to intervene in each and every moment. That the theistic God is compelled to micromanage the universe at extraordinary levels. This however is not explicitly stated in the premises of the argument but given a universe in constant flux with numerous entities interacting there are too many dynamic elements for the execution to occur without issue.

Assume that a program is perfectly designed and has no user interaction; all elements of the program simply interact within predefined boundaries. The program still has a dependency on external sources such as the CPU and memory of the hardware it is executing on. A perfect program then is capable of failure because there are external dependencies. Likewise, any actualized world would necessarily have a set of rules, such as the laws of physics even if they are completely distinct from the known laws of the current universe. It is wholly plausible then that due to the interaction of external elements, such as the laws of an actualized universe, and the individuals in the universe there are possible events that could occur that are unintended. A tree falling on a person could still be possible in such a universe.

The argument then has to be that either God would create the universe in such a manner that such events did not impact the state of the people or that God would micromanage the events and immediately intervene to stop the tree from falling on the individual. If the universe was created in such manner as to allow no impact from interactions, then it seems the agents would be static. There is no question that God could create a universe of where all values are static, and no evil occurs and therefore this explanation does not suffice. It then must be assumed that God would be willing to intervene in each moment. Such a position implies that God did not have enough foresight to implement exception handling and proactively address the possible issues. This position then assumes that God is not omniscient. The other assumption of implying God must micromanage the universe is that God was careless or reckless in the design. Meaning, the argument present from the atheist has additional assumptions about the character of God built into it. It does not assume just (1) and (2) but also other conditions about the nature of God. This is why Mackie's addition of (3) is added to the argument; the additional assumption about the nature of God is an effort to compel the theist to accept that God must act in certain manner. Perhaps (3) is intended to establish the necessity of God to micromanage, but that changes the God the argument is about, modifying the attributes and no longer addressing the theistic God.

(6) and (7) then are false because they modify God beyond the theist's position. The dependence on (3) is claiming more than the theist must claim and (6) and (7) only hold true if the world actualized is static. If the agents in the world are mutable then there are possible outcomes from interactions that could lead to undesirable results.

Evaluating Mackie's Good eliminate Evil Principle Separately

Utilizing the reworked problem, (3) does not immediately appear to be true. By removing the property of goodness from God, the necessity to eliminate that which is opposed to

His will is not based on the Christian theist's position. However, J.L. Mackie affirms that the logical problem needs the additional principles, stating:

> "However, the contradiction does not arise immediately; to show it we need some additional premises, or perhaps some quasi-logical rules connecting the terms 'good,' 'evil,' and 'omnipotent.' These additional principles are that good is opposed to evil, in such a way that a good thing always eliminates evil as far as it can, and that there are no limits to what an omnipotent thing can do."[70]

Goodness is not a property of God, rather God is the standard and goodness is measured in relation to God. This means that for Mackie to continue with this claim he must find an attribute of God that necessitates the opposition to evil and not the nebulous "good." What specific moral value compels the elimination of evil? Mackie's weakness here is to set good and evil and forces and not moral standards. By stating the principle as "good is opposed to evil" the argument has established an ambiguous and isolated view of good and evil. This shift allows the atheist to impose a premise on the theist's position that is opposed to the theist's view.

As a moral standard, goodness must be in relation to something. A person that performs acts of kindness is a good person. It cannot be stated that the inverse is *necessarily* true. There is nothing obligatory of a good person to perform acts of kindness. Perhaps the good person is simply an honest person or a just person. A non-good or neutral entity might be opposed to evil which makes this point all the more puzzling. Given the additional principle, there could not be a "good" object instantiated without also instantiating a "bad" object. In the programming example then, God would be required to instantiate humans with values that allowed evil to maintain goodness, because there is inherent aspect of the quality of goodness to be opposed to evil. If the opposition of evil is not a requirement of goodness, then Mackie's principle fails because a good thing does not have to oppose evil. Either there is a contingent relationship

[70] J.L. Mackie, 82.

between good and evil, or there is not. If goodness is contingent on opposing evil, then goodness is contingent on evil existing. That means an instance of the God class could not be instantiated without instantiating another object of some sort. Mackie cannot be claiming a contingent relationship though, for that would be demanding that there are always two objects instantiated, God and something else that is evil.

The intention of Mackie's premises is likely to attribute good and evil as qualities, but then why would a quality of goodness eliminate a quality of evilness? Beauty does not seek to destroy that which is ugly, and kindness does not seek to eliminate rudeness. As a quality, it simply exists as an aspect of an entity, not a causal force. Mackie's principle that "a good thing always eliminates evil" does not follow the aspects of normal attributes. The principle is simply an additional overreach due to not defining good and evil in the normal manner. Does a "good" human object necessarily also have to eliminate moral evil? If the quality of goodness demands that, then a human might be "good" simply because it opposes evil with harsh preprogrammed sayings, as that it "as far as it can" eliminate evil, assuming that it was instantiated with those responses and additional values not listed in the human class. Modifying the argument, as Mackie did, does not add additional weight. It reduces the argument to a notion of God that the theist is not defending and should not attempt to defend.

CHAPTER FIVE – ADDRESSING OBJECTIONS AND GAPS

Any objections to the argument must consider what is not claimed. First, it is not a claim that the world is a program or that the world operates as a program. The principles of programming are ways of modeling the world and in the argument the principles are applied to analyze philosophical challenges in the world. Second, the argument greatly simplifies true programming requirements and syntax to create the model and therefore programming complexities are not a suitable counter to what was presented. Last, the assumption of a libertarian view of free will is applied due to the presumption of accountability being a requirement for an action to be moral; it is not a requirement for the model to be successful and therefore attacking libertarian free will does not address the argument. With those three points of consideration each anticipated objection is considered below.

Objection: Change could be controlled by restricting variables

If change is capable of causing evil, restricting the type of change or the variables available to create change could have been done initially to prevent evil. The argument states that God could create a world witch static agents with no evil, but not a world with dynamic moral agents. However, reducing the variables that apply changes could prevent evil while still allowing dynamic moral agents to exist. Given the example of the log, simply create the world where trees do not fall or where emotions cannot modify values and then evil would not exist.

There are multiple considerations in response to this objection. When the God class was defined it had two methods: *create* and *giveGift*. This means that if God creates multiple worlds or an infinite number of worlds, all those worlds are instantiated from the same instance of God. Therefore, all the humans in all those worlds would inherit from the same instance of God. Amplifying the number of worlds does not change human design. The objection then must

assume that humans could be instantiated with different values in one those worlds; principally that humans could be instantiated with *canChange* set to a value of *false*. By modifying *canChange*, that means humans would not function the way humans are known to function. If *canChange* is *false*, then the instance of the human would either never age or remain the same age it was instantiated at. Then knowledge could not be acquired and instead would remain static as well. Whatever the values were at instantiation, those values would remain static, and it is already accepted that a static agent could be created without evil. This means that simply changing the value of *canChange* to *false* is not enough.

Now, the objection does not limit the argument to only a single variable having a different value. Perhaps new variables could be added to achieve a world without evil. If the Human class had a variable of *allowEmotionsToModifyValues* and that was set to *false* it would allow *canChange* to be *true* but limit the sources that can generate change. Limiting the impact of emotional change would not suffice because external factors could generate change without emotional impact. A broken body could impact power or brain injury might impact knowledge. An additional variable would have to be added to prevent physical forces from modifying values. The Human class could add *allowPhysicalForcesToModifyValues* and instantiate humans with that value set to *false*. But humans still have a conscious mind that thinks, and processes information and the internal thoughts and activity could also impact the values – modifying knowledge or empathy through thoughtful consideration. Another variable could be added to the human class to limit the impact of internal workings, *allowInternalProcessesToModifyValues*. The more variables are added to restrict change from various sources the closer to a static object the instance becomes. Restricting the variables is only effective in managing the granularity in which restriction occurs, but if value modification is possible then evil remains an option.

Preventing trees from falling results in the same result for humans. It is not sufficient to prevent external events from occurring. The point here is that there are additional factors that impact the values in humans and not all changes are initiated by emotions or external experiences. Simply restricting one or some allows modification to still occur and evil to remain a possibility, restricting all results in a static agent.

Objection: Restricting choices available could prevent evil

The argument of restricting choices aligns with a belief in closed limited libertarian free will. Gellman defines closed limited libertarian freedom as, "freedom to choose between good options, without freedom to choose evil."[71] Simply put, only provide a range of options of good choices in which evil choices are not presented. James Cain summarizes the argument as, "A well-known objection to the free will defense holds that the exercise of free will is compatible with determinism. Therefore, it is argued, if God exists, God could have allowed the exercise of free will by creatures, and yet predetermined exactly how the will would be exercised. Thus, God could have prevented the existence of evil that arises through the misuse of the will without having to exclude the existence of free will in creatures."[72] Reducing this to a more programming related scenario might be stated as, if the movement of human is restricted to forward because that is "good," then the human can never go backward which is "evil." In this manner, the human chooses "good" freely and evil does not exist.

The notion of restricting operation to only moving forward provides a starting point to addressing this argument. To meet the criteria of having only good choices, humans are also

---

[71] Jerome Gellman, "On a New Logical Problem of Evil," *Faith and Philosophy: Journal of the Society of Christian Philosophers* 32, no. 4 (2015): 443.

[72] James Cain, "Free will and the problem of evil," *Religious Studies* 40 (2004): 440.

then granted the choice to go left because both left is also "good." Given this simple movement restriction then, the program executes, humans cannot go backward because that is evil and cannot travel right because that is not an option. The choices are limited and predetermined to be "good" in the hopes that it will prevent the human from moving where to an evil place. The result is that by only making "good" choices the human can traverse backward or even to the right (which is not "evil" but also not a defined choice). Restricting the movement leads to the possibility of "evil" when greater than a single option is presented.

The other issue with this objection is the way in which choices are reduced to a set of predefined paths. Modern programming is event driven, responding to the inputs and acting on them. Aleksi Lukkarinen, Lauri Malmi, and Lassi Haaranen confirm this approach to programming stating, "During the past two decades, event-driven programming (EDP) has emerged as a central and almost ubiquitous concept in modern software development: Graphical user interfaces are self-evident in most mobile and web based applications, as well as in many embedded systems, and they are most often based on reacting to events."[73] Restricting input to valid values controls program flow, but since invalid values are a possibility in a system driven by events, programming uses error handling to address invalid values. Restricting the choices means the system is generating the choices and not the dynamic agents. The choices are predetermined based on events, rather than choices being generated due to events. If choices are generated by events and not the system, then it is possible to have "evil" choices in the system because those would be based on the values of variables when the event occurs. In the God class was the method *giveGift,* and that method was inherited in the Human class. The method allows

[73] Aleksi Lukkarinen, Lauri Malmi, and Lassi Haaranen, "Event-driven Programming in Programming Education: A Mapping Review," *ACM Transactions on Computing Education 21, no. 1* (March 2021): 2.

a gift to be given to a target but the determining factor for what is capable of being given is based on the values of the variables that are used in the method: *power*, *knowledge*, *love*, and *kindness*. Possible gifts are restricted based on the values then and not predetermined before the instance is ever created. The possible gifts of a human at one moment in time then will not necessarily be the same possibilities. Also, the possible gifts available to give at that exact moment for all humans instantiated would be different because the values determine the choices available. Events drive the system and the options available are determined by the values of the variables at the time. To argue that only "good" choices could presented is to argue that the system itself designates the choices available for each interaction and therefore the system is not actually interactive, but interactions are predetermined. A non-interactive system could be created without evil because the agents are not able to act or engage outside of scripted scenarios and those scenarios would have controlled outcomes.

Objection: Machine learning chooses without preset choices

As programming continues to advance, one area of growth in artificial intelligence comes through machine learning. Machine learning is an attempt to create a type of reasoning to allow for deeper problem-solving skills. Léon Bottou defines reasoning as, "algebraically manipulating previously acquired knowledge in order to answer a new question."[74] Instead of just restricting the choices, machine learning allows new responses to be generated as the program executes. God could therefore create a system that adapts and grows over time while only allowing "good" choices to be available.

[74] Léon Bottou, "From Machine Learning to Machine Reasoning: An Essay," *Machine Learning* vo. 94, no. 2 (February 2014): 133.

The problem with machine learning as a solution to restriction are the initial requirements of building the model. Machine learning requires an initial set of data with relevant tags and then it is necessary to train the machine learning models from that data. Bottou states that training is to, "algebraically enrich the set of manipulations applicable."[75] This means that God would have to create the data before the system was instantiated and tune the algorithms before ever instantiating an object. First, this automatically implies deeper assumptions on the nature of the knowledge of God that are not implied by the argument itself. It might target specific views of the knowledge of God but cannot be effectively applied to theism broadly. Since the nature of God's knowledge is not specified in the logical problem of evil this cannot be applied as a broad attack on theism. Second, machine learning builds the choices, as the machine learns the resulting set of choices is not predetermined in the algorithm. Simply, generically applying machine learning in a broad assault does not advance a true objection. The objection would have to demonstrate a model that grows continually and only produces desired results. This objection is simply too generic to provide weight and ignores the complexities of machine learning, and machine learning is trying to mimic human reasoning. As Bottou states, "Human reasoning displays neither the limitations of logical inference nor those of probabilistic inference."[76]

Summary of Objections

The objections presented reduce humans to robots. Instead of being moral agents, humans are implied to be robots with skin. "A robot is an automatic or semiautomatic machine

[75] Bottou, 133.

[76] Bottou, 134.

capable of purposeful motion in response to its surroundings in an unstructured environment."[77] David Gunkel notes that robots and animals are similar in Descartes' view, because Descartes argues animals, "simply followed predetermined instructions programmed in the disposition of their various parts."[78] If Gunkel is right, then the more a human is reduced into the same category as other agents, the less morally responsible humans are, not the other way around. This reduction does not promote the animal or robot to a moral agent, it reduces the human to a mechanism that is not responsible for any actions it takes. Michael Tooley reduces the human to deterministic causes from the brain stating, "In short, many phenomena – some very familiar, others much less so – would be not only surprising and mysterious but also hard to explain if the mind were an immaterial substance. By contrast, however, those phenomena are perfectly explainable if the mind is, instead, the brain. The conclusion, accordingly, is that there is massive evidence against the view that the mind is an immaterial substance."[79] This reduction removes uniqueness from the human agent and then creates equality between animals, humans, and robots. A robot is not a moral agent though, so reducing humans to robots in function and execution, but human only by name is not addressing the argument. Gunkel analyzes many different views and confirms the non-moral agency of robots stating, "Understood as an extension or enhancement of human faculties, sophisticated technical devices like robots, AIs, and other computer systems are not considered the responsible agent of actions that are

[77] Eugene Kagan, Nir Shvalb, and Irad Ben-Gal, eds *Autonomous Mobile Robots and Multi-Robot Systems : Motion-Planning, Communication, and Swarming* (Newark, 2019), 6.

[78] David J. Gunkel, *The Machine Question: Critical Perspectives on AI, Robots, and Ethics* (Cambridge, 2012), 3.

[79] Michael Tooley, *The Problem of Evil* (Cambridge, 2019), 15.

performed with or through them."[80] Non-moral agents cannot perform evil acts, only moral agents are capable of evil; moral agents might cause a non-moral agent to perform an act of evil on behalf of a moral agent, but it is evil only in the context because the moral agent was the cause. So, the objections to the argument are not an attempt to demonstrate that God does not exist, they are meant to demonstrate humans cannot be held culpable for the actions taken.

Conclusion

The logical problem of evil argues that the existence of evil results in the denial of the theistic God because an omnipotent and omniscient being have the power to prevent evil. Since evil undeniably exists, the set appears inconsistent with the theist's notion of God. However, the argument does not take into consideration that evil and good are not simply properties, but they correspond to a standard – something is only good in so far as there is a standard to compare against. Misunderstanding "good" has led to an argument that takes on anthropomorphic behaviors, such as Mackie's addition of good "opposing" evil – as if to apply a conscious force to "good" and "evil." This confusion makes addressing the argument take unnecessary routes and forces multiple hidden presuppositions on the theist, forcing a specific notion of God that only conforms to the atheist's view.

Addressing this problem then starts with defining "good" and "evil" in terms of the standard they represent. Moral goodness is simply a reference to the moral values that God contains. Kindness is good because God possesses the value of kindness. By removing the abstract notions of "good" and "evil" from the argument the argument can then be solved with some simple programming principles.

[80] Gunkel, 27.

Inheritance provides a mechanism for moral agents to know what good and evil are because the values are inherited from God. Each human instance contains the moral values that God contains. Sam Harris attempted to develop a moral picture absent any objective source, but failed because his entire argument was built on, "Once we see that a concern for well-being (defined as deeply and as inclusively as possible) is the only intelligible basis for morality and values, we will see that there must be a science of morality, whether or not we ever succeed in developing it: because the well-being of conscious creatures depends upon how the universe is, altogether."[81] The initial argument clearly represents a universal value of inheritance and therefore provides reference to a standard that is not self-generated. Inheritance provides a way for everybody to have a similar sense of moral values, but programming also teaches that those values are able to be modified which can explain why Harris and others see values as subjective. Michael J. Murray writes, "There is indeed a good deal of evidence that lots of human behavior we would describe as 'moral' arises from innate or hardwired dispositions."[82] Those "hardwired dispositions" are the inherited values. The distinction between philosophy and programming provides the final simplified piece of utilizing inheritance to help solve the problem. Values in the philosophical sense are represented by variables in the programming sense, and the moral variables can be modified based on numerous factors which could result in distinct value differences among different human instances based on nurturing, culture, life events and other factors. However, the underlying moral variables remain present in each instance and the values could be perceived as subjective.

---

[81] Sam Harris, *The Moral Landscape: How Science can Determine Human Values* (New York, 2010), 28, Kindle.

[82] Michael J. Murray, *Contending with Christianity's Critics: Answering New Atheists and Other Objectors* by William Lane Craig (Nashville, 2009), Chapter 4, "Belief in God: A Trick of our Brain?" para. 7, Kindle.

Another aspect of programming is exception handling, which allows for various errors to be handled in code to prevent the execution of the program from failing. One example is preventing a divide by zero error and branching the code to address the issue. By branching the code, the code takes a different path that it would under other conditions but allows the program to continue to execute. An iPhone does not fail to work simply because it cannot connect to WiFi and bank software does not fail because the balance went to zero or below. This type of pro-active error handling is useful in any program with multiple dependencies and variables. It means if a program does encounter an issue it can continue. This then is extrapolated to God to argue that God would be able to pro-actively address failures in a more sufficient manner and with less intrusion. So, rather than prevent evil from occurring, God is able to plan ahead to handle the outcomes in the event that evil happens. Since the variables are inherited from God, humans cannot contain the same maximum values of the infinite God. This means the values of those variables that are inherited are limited and as such any increment or decrement of the values provides a potential point of failure. WiFi connections would not fail for an iPhone if WiFi did not exist, but since WiFi does exist and can be of multiple connective states, there is a point of failure that is dependent on something external to the phone. Similarly, since there are external dependencies that exist for humans that can modify moral values, there are multiple points of moral failure. God could be prevented from actively intervening in evil situations, in which case exception programming demonstrates that God is ultimately providing a method for humanity to continue.

The result is that inheritance explains both the reason for moral values and the reason for changes in those values. Exception handling demonstrates that failures can happen, but error handling is there to prevent catastrophic failure of the system. Having the option to modify the

values, along with multiple dependencies, creates the opportunity for evil to occur. It is therefore not inconsistent that God can be all powerful and have all knowledge and yet have evil exist in the world. God could simply have already planned for such failure a provided a way for the world to continue. In fact, that is the very message of redemption in Christianity.

References

Al Dallal, Jehad, and Sandro Morasca. "Predicting Object-Oriented Class Reuse-Proneness Using Internal Quality Attributes." *Empirical Software Engineering* 19, no. 4 (August 2014):775-821.

Allen, Sophie. *A Critical Introduction to Properties*. London: Bloomsbury Academic, 2016.

Almeida, Michael J. "The Logical Problem of Evil Regained." *Midwest Studies in Philosophy* 36, no. 1 (2012):163-76. https://doi.org/10.1111/j.1475-4975.2012.00240.

Alston, William. "What Euthyphro Should Have Said," in *Philosophy of Religion: A Reader and Guide*, edited by William Lane Craig. Edinburgh: Edinburgh University Press, 2001.

Blanco, Angela. "Values and Socio-Cultural Practices: Pathways to Moral Development." *The Oxford Handbook of Culture and Psychology* (May 2012): 1-31. https://doi.org/10.1093/oxfordhb/9780195396430.013.0036.

Bottou, Léon. "From Machine Learning to Machine Reasoning: An Essay." *Machine Learning* 94, no. 2. (February 2014): 133.

Cain, James. "Free will and the problem of evil." *Religious Studies* 40 (2004): 437-56. https://doi.org/10.1017/S0034412504007231.

Carry, Phillip. "A Classic View." In *God and the Problem of Evil: Five Views*, edited by Chad Meister and James K. Dew Jr., 15. Westmont: InterVarsity Press, 2017.

Craig, William Lane. "A Molinist View." In *God and the Problem of Evil: Five Views*, edited by Chad Meister and James K. Dew Jr., 41. Westmont: InterVarsity Press, 2017.

Craig, William Lane. *Contending with Christianity's Critics: Answering New Atheists and Other Objectors.* Nashville: B&H Publishing Group, 2009, Kindle.

Craig, William Lane. *On Guard: Defending Your Faith with Reason and Precision.* Colorado Springs: David C. Cook, 2010, Kindle.

Craig, William Lane. *Reasonable Faith,* 3rd ed. Wheaton: Crossway Books, 2008, Kindle.

Crenshaw, James. *Defending God: Biblical Responses to the Problem of Evil*. Oxford: Oxford University Press, 2005.

Demey, Lorenz. "The Porphyrian Tree and Multiple Inheritance." *Foundations of Science* 23, no. 1 (March 2018): 173-80.

Doherty, Simon, Groves, Lindsay, and Victor Luchangco, et al. “Towards Formally Specifying and Verifying Transactional Memory.” *Formal Aspects of Computing*, no. 25 (2013): 770.

Ebert, Felipe, Castor, Fernando, and Alexander Serebrenik. “An Exploratory Study on Exception Handling bugs in Java programs.” *Journal of Systems and Software*, no. 106 (2015): 82-101.

Gellman, Jerome. "On a New Logical Problem of Evil." *Faith and Philosophy: Journal of the Society of Christian Philosophers* 32, no. 4 (2015). https://doi.org/10.5840/faithphil201592144.

Gobbo, Federico, and Marco Benini. “Why Zombies cannot write significant source code: The Knowledge Game and the art of computer programming.” *Journal of Experimental & Theoretical Artificial Intelligence* 27, no. 1 (2015): 37-50. https://doi.org/10.1080/0952813X.2014.940142.

Gunkel, David J. *The Machine Question: Critical Perspectives on AI, Robots, and Ethics.* Cambridge: MIT Press, 2012.

Halbert, D.C., and P.D. O’Brien. “Using Types and Inheritance in Object-Oriented Programming.” *IEEE Software* 4, no. 5 (Sept. 1987): 71-79.

Hare, R.M. “’Good’ In Moral Contexts.” *The Language of Morals*. Oxford: Oxford University Press, 1963.

Harris, Sam. *The Moral Landscape: How Science can Determine Human Values*. New York, 2010, Kindle.

Harrison, Nonna Verna, and David G. Hunter, eds. *Suffering and Evil in Early Christian Thought (Holy Cross Studies in Patristic Theology and History)*. Grand Rapids: Baker Academic, 2016.

Hommen, David. “Kinds as Universals: A Neo-Aristotelian Approach.” *Erkenntnis* 86, no. 2 (April 2021): 295-323. https://doi.org/10.1007/s10670-019-00105-6.

Kagan, Eugene, Shvalb, Nir, and Irad Ben-Gal, eds. *Autonomous Mobile Robots and Multi-Robot Systems: Motion-Planning, Communication, and Swarming.* Newark: John Wiley & Sons, 2019.

Kaiser, Walter C., Jr. *The Majesty of God in the Old Testament: A Guide for Preaching and Teaching*. Grand Rapids: Baker Academic, 2007.

Koons, Jeremy. “Can God’s Goodness Save the Divine Command Theory from Euthyphro?” *European Journal for Philosophy of Religion* (Spring 2012): 177-95.

Kraul, Richard. *What Is Good and Why: The Ethics of Well-Being.* Cambridge: Harvard University Press, 2007.

Kulkarni, Devdatta, and Anand Tripathi. "A Framework for Programming Robust Context-Aware Applications." *IEEE Transactions on Software Engineering* 36, no. 2 (March 2010): 184-97.

Law, Stephen. “The Evil-God Challenge.” *Religious Studies* 46, no. 3 (2010): 353–73. https://doi.org/10.1017/S0034412509990369.

Lerner, B. S., et. al. "Exception Handling Patterns for Process Modeling." *IEEE Transactions on Software Engineering* 36, no. 2, (March-April 2010): 162-183.

Lukkarinen, Aleksi, Malmi, Lauri, and Lassi Haaranen. “Event-driven Programming in Programming Education: A Mapping Review.” *ACM Transactions on Computing Education* 21, no. 1 (March 2021). https://doi.org/10.1145/3423956

Mackie, J.L. *The Problem of Evil: Selected Readings,* 2nd ed. Notre Dame: University of Notre Dame Press, 2016.

McDonough, James E. *Object-Oriented Design with ABAP: A Practical Approach*. Pennington: Apress, 2017.

Meynell, Hugo. “The Euthyphro Dilemma.” *Proceedings of the Aristotelian Society, Supplementary Volumes* 46 (1972):223-34.

Moreland, J.P., and William Lane Craig. *Philosophical Foundations for a Christian Worldview*, 2nd ed. Downers Grove: InterVarsity Press, 2017.

Morton, Adam. *On Evil*. London: Taylor and Francis Group, 2004.

Oliveira, Juliana, et. al. “Do Android Developers Neglect Error Handling? A maintenance-Centric Study on the Relationship between Android Abstractions and Uncaught Exceptions.” *Journal of Systems and Software*, no. 136 (2018): 1-18.

Palmer, David, editor. *Libertarian Free Will: Contemporary Debates*. Oxford: Oxford Scholarship Online, 2015. https:// 10.1093/acprof:oso/9780199860081.001.0001

Pavelich, Andrew. “The Moral Problem with the Free Will Defense Against the Problem of Evil.” *The Heythrop Journal* 60, no. 5 (2019):678-688. https://doi.org/10.1111/heyj.12654.

Plantinga, Alvin. *God, Freedom, and Evil*. Grand Rapids: William B. Eerdmans Publishing Company, 1977. Kindle.

Purdum, Jack. *Beginning Object Oriented Programming with C#*. Birmingham: Wrox, 2012.

Schellenberg, J. L. “A New Logical Problem of Evil Revisited.” *Faith and Philosophy* 35, no. 4 (October 2018):464-472. https://doi.org/10.5840/faithphil20181016112.

Singh, Sunil Kumar, Kumar, Rajnish, and Kamal K. Mehta. “Software Reusability through Object Oriented Inheritance Tree Metric.” *I-Manager’s Journal on Software Engineering* 3, no. 3 (January 2009): 1-5.

Swinburne, Richard. *The Coherence of Theism,* 2nd ed. Oxford: Oxford Scholarship Online, 2016. https://doi.org/10.1093/acprof:oso/9780198779698.001.000.

Tieszen, Richard. “Teaching Formal Logic as Logic Programming in Philosophy Departments.” *Teaching Philosophy* 15, no. 4 (December 1992): 337-47. https://doi.org/10.5840/teachphil199215454.

Tooley, Michael. *The Problem of Evil*. Cambridge: Cambridge University Press 2019.

Tylman, Wojciech. "Computer Science and Philosophy: Did Plato Foresee Object-Oriented Programming?" *Foundations of Science* 23, no. 1 (March 2018): 159-72.

Udvaros, József, and Miklós Gubán. “Demonstration the Class, Object and Inheritance Concepts by Software.” *Acta Didactica Napocensia* 9, no. 1 (2016): 23-34.

Wertz, Harald. *Object-Oriented Programming with SmallTalk*. Amsterdam: Elsevier, 2015.

Wirfs-Brock, Rebecca J. “Toward Exception-Handling Best Practices and Patterns." *IEEE Software* 23, no. 5, (Sept.-Oct. 2006): 11-13.