**Automated Code Generation through**

**Analysis of Inputs and Outputs**

MASTER THESIS

at the University of Applied Sciences Hof

Faculty IT

M.Eng. Software Engineering for Industrial Applications

Presented to

Prof. Dr. Ralph Lano

Alfons-Goppel-Platz 1

95028 Hof

Germany

Presented by

Gokul Rajan

Wartbergstrasse 45

74076 Heilbronn

Germany

October 2017

# Declaration of Authorship

I, Gokul Rajan, declare that this thesis titled, ‘Automated Code Generation through Analysis of Inputs and Outputs’ and the work presented in it are my own. I confirm that:

* This work was done wholly or mainly while in candidature for a Master’s degree at this University.
* Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
* Where I have consulted the published work of others, this is always clearly attributed.
* Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
* I have acknowledged all main sources of help.
* Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed.

Signed:

Date:

# Abstract

The aim is to develop a system that can generate a program from the set of inputs and outputs that the program is supposed to take in and produce. The system was made to analyze these inputs and outputs and predict the properties of the program to generate the code at the end. Due to syntactical complexity and vast domain size, it was decided to reduce the domain size of program to be generated programming languages like Java, C, C++ to a much simpler yet powerful educational programming language, Karel (developed by Richard E. Pattis, 1981). As Karel can perform all the operations of a Turing machine, proving that this system can generate all sorts of programs for Karel, can prove that the system can generate almost any complex program.

Two approaches were followed for achieving this task. First approach was to develop a Genetic Algorithm, which will generate a random population of solution programs based on the input at the beginning and eventually will generate the program through numerous iterations to reach the output. Second approach was to train a general purpose deep learning framework with sets of inputs and outputs, and a set of valid programs for the same. The system was able to predict a program afterwards, for a new set of test inputs and outputs.

Among the two approaches the first was able to generate efficient results for programs involving loops and recursive actions. The latter was more time efficient than the former one. In the paper the possibility of a combination of these two algorithms to generate a genetic algorithm with efficient fitness functions and generate better population with lesser number of iterations is also discussed.

# Acknowledgement

I would first like to thank my thesis advisor Prof. Dr. Lano Ralphat **Hof University of Applied Sciences**. Prof. Ralph was always available whenever I ran into a troublesome spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right direction whenever he thought I needed it.

Finally, I must express my very profound gratitude to my parents and to my brother for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing the thesis. This accomplishment would not have been possible without them. Thank you.

Author

Gokul Rajan  
24 October 2017

# Contents

Declaration of Authorship 2

Abstract 3

Acknowledgement 4

Contents 5

List of figures 7

List of Abbreviations 8

1 Introduction 9

1.1 About Karel the Robot 10

1.2 Organization 12

1.3 Internship Focus 12

2 Need for Continuous Testing 13

2.1 Problem Statement 13

2.2 Current Scenario 13

2.3 Possible Solution 14

3 Continuous Testing System 16

3.1 About the CT system 16

3.2 Static Code Analysis 16

3.2.1 Workflow 16

3.2.2 Implementation 17

3.3 Standardized procedure for continuous testing 17

3.3.1 Unit Testing Strategies 19

3.3.2 Integration Testing Strategies 21

3.3.3 Scenario Testing Strategy 23

3.3.4 Overall Structure 23

3.3.5 Overall CIDM Use Case 24

3.3.6 Integrating Cocoapods 27

3.3.7 Integrating CIDM server 34

3.3.8 Abnahmekriterien 40

3.3.9 Durchgeführte Arbeiten 40

3.3.10 Ergebnis 40

3.3.11 Gestaltungsqualität 40

3.3.12 Zusammenfassung 40

Summary 41

Future Work 42

Bibiliography 43

List of tools considered 44

Eidesstattliche Versicherung 45

# List of figures

1.1 CIDM System Architecture 5

2.2 Current structure of dependency management 3

2.3 Proposed structure of project integration 2

2.4 Proposed structure of dependency management 3

3.1 CIDM System Architecture 2

3.2 CIDM System Infrastructure 2

# Abbreviations

**GA…………...…………...Genetic Algorithm**It is the name of the system developed for handling dependency management (in Java and iOS/ OSX platforms) and auto  
mating continuous integration using Jenkins.

**DSL…………………………...Domain Specific Language**It is the practice of merging all developer working copies to a  
shared mainline several times a day. Grady Booch first named and proposed CI in his 1991 method.

**CD……………………….….Continuous Delivery**It is the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments into production, or into the hands of users, *safely* and *quickly* in a *sustainable* way.

**SDLC………………….……Software Development Life Cycle**It adheres to important phases that are essential for developing a software, such as planning, analysis, design, implementation, testing and deployment.

# Introduction

AI is defined as the study of agents that receive precepts from the environment and perform actions (Stuart Russell, 2010). This means introduction of a perfect AI can easily automate a lot of tasks which require cognition. Though many people hate the evolution in this field due to factors like unemployment or something bad happening, evolution in this field is inarguably, inevitable. Almost all technological giants around the world including Google, Microsoft, Facebook, Amazon and IBM are actively working in this domain. The use of machine learning (a subdomain of AI) to extract non-obvious information from huge amounts of data (also known as Data Mining) is a common thing right now.

The ultimate goal for creating an artificially intelligent system is to solve problems and achieve goals just like humans, so that the idea of solving a problem itself can be automated. A small step in this direction would be to build systems that can write programs by analyzing ‘what it has’ and ‘what it wants to achieve’. Several recent successful attempts have been done in this field. Google DeepMind’s victory against the world champion in the game Go, was followed by the software’s capability to play, learn and master Atari games with pixel data and score information. Microsoft DeepCoder’s ability to write beginner level coding competition programs happened recently. All these were possible using evolutionary algorithms which can learn from experience and re-adjust themselves until they reach their goal.

In this thesis document, two ideas are proposed using the following learning algorithms for solving the problem.

1. Genetic Algorithm
2. Deep Reinforced Learning

The techniques were used to automatically generate computer program by analyzing the set of inputs it needs and the set of outputs that it will produce.

## Goal

The only thing that stands between an idea and its realization is its implementation. If a software can perform this automatically by analyzing your expected inputs and results, that can drastically change the perspective of people about realizing their ideas. If the software cannot automate the development process completely, it would still be very much relevant if it can increase the development speed by developing the components along with the developers during the software development process.

In the thesis, the idea to automate computer programming is implemented need to be achieved through the analysis of initial and final states of the Karel map. And the software output is the set of instructions that Karel would need to perform for generating the output state of the map. The end goal of the thesis is to solve the problems in the world of Karel and there by proving that such a software can be implemented and to show the application of such a software in a much larger domain is possible.

## Motivation

The inspiration for creating this work came from my Bachelor Project to create source code in ‘C’ language from user specification in English language. The compiler to process this specification was made in a way to grow its dictionary on its own by constantly interacting with the user by providing suggestions. The aim of the project was to make a software which can generate source code for those people who can clearly specify what they require but were not developers. But the development of the project resulted in its evolution from requirement specification to a Domain Specific Language (DSL).

To overcome this, two main changes were made to the project.

1. The program domain was changed from ‘C’ programming language to a much simpler introductory language, Karel.
2. Instead of compiling the software specification, the input and output sets were analyzed to generate the code.

By doing the first change, the instruction sets were drastically reduced. Also as Karel can do all the operations done by the classical Turing machine, it can represent complex problems with these limited instructions. With the second change, the complexity of language processing was removed from the problem. It also moves the responsibility of generating the logic for deriving the output from the input (solving the problem) from the user to the computer.

## Organization

The contents of the report are organized as follows:

This chapter introduces the topic, a brief description about why this topic is relevant and finally, specifies the goal and motivation for writing this thesis.

Chapter 2 introduces about machine learning and both the strategies that are used to crack the thesis problem, genetic algorithm and reinforced learning. It is also explained what problems are in general handled with these algorithms and they were chosen for this particular problem.

Chapter 3 explains the thesis problem in detail. It is also explained how the two strategies were applied for solving the problem and what were the results.

Finally, the future applications of the software are explained.

# About Karel Code

The program generator is designed to generate instruction sets for the Karel code. Karel being a simplistic domain, resembles the turing machine in solving complex logical operations. The reason for reflecting the bigger problem of automated program generation to the domain of Karel is because of the same.

## Problem Statement

Ultimately, we are trying to prove that programming as a process can be automated and work in this direction can simplify the software development process and increase development speed. Through the Karel prototype, in a smaller domain, this has to be proven.

In the world of Karel, through the analysis of the input and output states of the Karel map, the program should generate the instruction sets for Karel to run through the map to achieve the output state. To be precise, the software should generate the core logic of a Karel problem by analyzing the input and output of the problem.

## Current Scenario

Regardless of the size of the project or domain, in general computer programming works in the following fashion

* The developer will get the set of clearly defined requirements and specifications. This includes clearly defining the what will be the input and what we are expecting from the software.
* The specification for the software is defined as per the collected requirements for the software.
* The software developers write the program based on the defined specification. The development process is done again after testing and validation until the desired software has been implemented.

This process of programming is clearly defined. In a larger domain, this may seem complicated with all the shared dependencies, build systems and dev-op solutions. But in a smaller scale like the world of Karel, the problem can be reduced to a much smaller one. In the world of Karel, considering only the development of core logic of the program, this. Moreover, the process with humans has various drawbacks like the following:

* From a layman‘s point of view, this is a time consuming process and is prone to human errors.
* From an organization‘s point of view, this costs a lot of money.
* From a much larger perspective, slowing down the process of solving problems and realising intelligent ideas can slow down the pace of technology as it is.

## Possible Solution

For solving the problem, we need to break down the problem of software development. Defining the requirements and generating software specification involves defining what the problem is and what is supposed to be the input and what is the output. This cannot be automated, as it requires intelligence and mostly creativity. What we can automate is the implementation as the process itself is defined under specification. Also, there are proper guidelines and methods for writing a program and algorithms which are already defined for various challenges.

The idea is not to immediately automatize programming as it is completely, but to clearly define its application in software development and automate the possible steps in it.

## State of the art

Though there were many attempts at creating an automatized code generation tool, most of them are implemented using metaprogramming techniques, reducing code size and increasing productivity (SK-logic, 2011). The concept became limited to programming languages in with dynamic programming language features. Like , Lisp and other scripting languages.

Once machine learning gained popularity, recently Microsoft came up with DeepCoder. They have made a working prototype, which uses a combination of two different strategies for generating code from requirements. The prototype was able to solve the intermediate level problems in coding contests (Matej Balog, 2017). The system has basically two parts, a part which writes algorithms and a part which is trained through a potential database of input, output and code set. DeepCoder was trained with a sets of input, output and code for very small programs (Gershgorn, 2017). The second part uses GA for generating the code which it cannot generate from its training. A domain specific language was defined for DeepCoder and through processing the set of inputs and outputs for the program, produces the actual program, in the DSL.

# About Machine learning

The Machine Learning field evolved from a much broader field of Artificial Intelligence, aiming to mimic intelligence in humans (Rätsch, 2004). Intelligence as it is cannot be mimicked if it cannot be defined. In the history of artificial intelligence, the algorithms defined were confined to a specific purpose rather than general purpose. And the algorithms were designed in a way, so that it knows all the set of outcomes for that specific purpose. A classic example would be the min-max algorithms used in games like chess, tic-tac-toe and checkers in the 1950s. But since 2015, the applications of Artificial Neural Networks in areas like computer vision became quite common as they surpassed the traditional hand-written classifiers in accuracy and also were more generic. i.e. One algorithm can be used for identifying various pictures and objects (Copeland, 2016). This was ground breaking and became popular fast as human did not tell the algorithm how to classify. The algorithm could generate rules on its own based on the training data provided. Google Deep Mind’s ability to master Atari games from pixel data and score information and, its victory against the Go champion are the recent examples for the same.

In our project the two machine learning strategies were used to crack the problem, of which the former is falls under the category of evolutionary algorithm and the latter falls under reinforced learning. The key similarity between the two algorithms is that it works on a feedback system with or without supervised learning. You are telling the algorithm whether the solution created was correct or not and based on how close is the generated output to the expected output, the algorithm modifies the generated solution.

## Genetic Algorithm

### Origin

In the late 1960s Rechenberg introduced “evolution strategies”, an idea that evolution could be used as an optimization tool for engineering problems. The concept of Genetic Algorithms was invented by John Holland. Hollands intention was to formally study the phenomenon of adaptation as it occurs in nature and to develop ways in which the mechanisms of natural adaptation can be imported into computer systems. Holland was the first to attempt to put computational evolution on a firm theoretical footing (Mitchell, 1999).

### Why Genetic Algorithm?

To define clearly, Genetic Algorithm is basically a method for solving constrained and unconstrained optimization problem. You can consider the process as an optimized brute force strategy.

A classic example is the Shakespear Monkey example which states that if a monkey randomly keeps on typing in a typewriter for an infinite amount of time, eventually the monkey will type out the complete works of William Shakespear (Shiffman, 2016). To see the practicality of this statement, we can consider the statement, “to be or not to be“. Suppose the typewriter had only 26 lowercase letters and space in it, the chances for the monkey to type that statement would be,

(1/27)^18 = 1/ 58,149,737,000,000,008,096,024,448

Suppose the monkey was typing at a rate of 1 million phrase combinations per second, still this would take 1,843,916,064,180 years in the worst case.

GA can reduce the solving time of this problem. In this case, by defining a fitness function to check the similarity between the generated phrase and the desired result, each time when a right character at the right location in the phrase is generated, the possible set of remaining combinations are reduced drastically. This pattern follows until the desired result is derived. (Shiffman, 2016)

Another example is the maze problem. Suppose you have a set of mazes whose source and destination are fixed, but the walls and the openings in it are different. If we follow the traditional maze algorithms, we are basically brute-forcing our way to the exit. We will be considering all the possible set of paths and trying each and every one of them. Using an optimization algorithm like GA, we can tell the computer that the paths leading towards the exit, have more chances of being the solution. There could be cases were the path is leading away from the exit at the beginning and then eventually leading towards the exit. With GA, just like in normal, natural evolution, the bad results should also be considered along with the best ones. This is done specifically to avoid scenarios like these,

### Implementation

For a GA to exhibit proper evolution behaviour, it has to exhibit the following properties as in Darwinian evolution .

Variation: The population must contain a vareity of traits for evolution to happen. If all the elements in the population are the same, evolution can never occur.

Heredity: The offsprings or population generated from an existing population should exhibit some properties of its parent

Selection: There must be a process through which some parents can pass down their properties to the next generation and some cannot. This process of passing down the desirable properties to the next generation is to mimic the natural evolution process of “survival of the fittest“.

1. **Create a population of solutions**

The population created should exhibit the first property, variation. There should not be 2 similar elements in a population. They should be completely random.

1. **Define fitness function and calculate the fitness for the elements in the population**

It represents the selection part of the evolution. The fitness function is the core part of the GA. It decides whether the generated element exhibits the desired properties and ranks the elements in the population so that the element with the highest rank will have higher influence will be more likely to pass their behavior of the next generation.

1. **Generate a new population based on the feedback from the fitness function**

*“Holland's GA is a method for moving from one population of "chromosomes" (e.g., strings of ones and zeros, or "bits") to a new population by using a kind of "natural selection" together with the genetics−inspired operators of crossover, mutation, and inversion*.” (Mitchell, 1999)

As the original GA implemented by John Holland, this step represents the selection process as well as the hereditary in the Darwinian evolution. This involves selecting the parent elements and making sure the child elements inherit some characteristics from the parent elements. The generation of a set of elements involves selecting the parents and making a new element. The number of selected parents can be two or more. The parents are selected based on the ranks or selection criteria as defined by the fitness function. Once this is done, the next step is to create a new element from the selected parents. The genetical operations are defined by considering the element as a chromosome and its properties as its genes. For creating a child element, the genes of the chromosome are altered in a way to ensure heredity. The following genetical operations are performed to achieve this,

* Crossover
* Inversion
* Mutation

The most common strategy to generate a new element from the selected parents is crossover. In crossover the subparts of the parent elements are exchanged, mimicking the biological recombination between two single chromosome organisms. In inversion, the random continuous part with random length, of the element, is reversed. With the above two genetical operations alone we can make sure that the newly created element exhibits heredity. The relevance of the mutation operation comes with the fact that sometimes if the initial random population or population generated at a later point of time, does not exhibit variation, the GA can end up in a cycle, were it is creating same sets of elements repeatedly. Mutation is done to avoid this by altering a small random portion of the generated element. Usually mutation is performed on 1 -5% of the total generated elements to ensure good balance between heredity and variety.

1. **Replace and Repeat**

Finally, the generated population is replaced with the newly created one and steps 2 and 3 are repeated to generate the next generation of elements.

## Reinforced Machine Learning

### Origin

In the late 1960s Rechenberg introduced “evolution strategies”, an idea that evolution could be used as an optimization tool for engineering problems. The concept of Genetic Algorithms was invented by John Holland. Hollands intention was to formally study the phenomenon of adaptation as it occurs in nature and to develop ways in which the mechanisms of natural adaptation can be imported into computer systems. Holland was the first to attempt to put computational evolution on a firm theoretical footing (Mitchell, 1999).

### Relevance of the Algorithm

To define clearly, Genetic Algorithm is basically a method for solving constrained and unconstrained optimization problem. You can consider the process as an optimized brute force strategy.

A classic example is the Shakespear Monkey example which states that if a monkey randomly keeps on typing in a typewriter for an infinite amount of time, eventually the monkey will type out the complete works of William Shakespear (Shiffman, 2016). To see the practicality of this statement, we can consider the statement, “to be or not to be“. Suppose the typewriter had only 26 lowercase letters and space in it, the chances for the monkey to type that statement would be,

(1/27)^18 = 1/ 58,149,737,000,000,008,096,024,448

Suppose the monkey was typing at a rate of 1 million phrase combinations per second, still this would take 1,843,916,064,180 years in the worst case.

GA can reduce the solving time of this problem. In this case, by defining a fitness function to check the similarity between the generated phrase and the desired result, each time when a right character at the right location in the phrase is generated, the possible set of remaining combinations are reduced drastically. This pattern follows until the desired result is derived. (Shiffman, 2016)

Another example is the maze problem. Suppose you have a set of mazes whose source and destination are fixed, but the walls and the openings in it are different. If we follow the traditional maze algorithms, we are basically brute-forcing our way to the exit. We will be considering all the possible set of paths and trying each and every one of them. Using an optimization algorithm like GA, we can tell the computer that the paths leading towards the exit, have more chances of being the solution. There could be cases were the path is leading away from the exit at the beginning and then eventually leading towards the exit. With GA, just like in normal, natural evolution, the bad results should also be considered along with the best ones. This is done specifically to avoid scenarios like these,

### Implementation

For a GA to exhibit proper evolution behaviour, it has to exhibit the following properties as in Darwinian evolution .

**Variation**: The population must contain a vareity of traits for evolution to happen. If all the elements in the population are the same, evolution can never occur.

**Heredity**: The offsprings or population generated from an existing population should exhibit some properties of its parent

**Selection**: There must be a process through which some parents can pass down their properties to the next generation and some cannot. This process of passing down the desirable properties to the next generation is to mimic the natural evolution process of “survival of the fittest“.

1. **Create a population of solutions**

The population created should exhibit the first property, variation. There should not be 2 similar elements in a population. They should be completely random.

1. **Define fitness function and calculate the fitness for the elements in the population**

It represents the selection part of the evolution. The fitness function is the core part of the GA. It decides whether the generated element exhibits the desired properties and ranks the elements in the population so that the element with the highest rank will have higher influence will be more likely to pass their behavior of the next generation.

1. **Generate a new population based on the feedback from the fitness function**

*“Holland's GA is a method for moving from one population of "chromosomes" (e.g., strings of ones and zeros, or "bits") to a new population by using a kind of "natural selection" together with the genetics−inspired operators of crossover, mutation, and inversion*.” (Mitchell, 1999)

As the original GA implemented by John Holland, this step represents the selection process as well as the hereditary in the Darwinian evolution. This involves selecting the parent elements and making sure the child elements inherit some characteristics from the parent elements. The generation of a set of elements involves selecting the parents and making a new element. The number of selected parents can be two or more. The parents are selected based on the ranks or selection criteria as defined by the fitness function. Once this is done, the next step is to create a new element from the selected parents. The genetical operations are defined by considering the element as a chromosome and its properties as its genes. For creating a child element, the genes of the chromosome are altered in a way to ensure heredity. The following genetical operations are performed to achieve this,

* Crossover
* Inversion
* Mutation

The most common strategy to generate a new element from the selected parents is crossover. In crossover the subparts of the parent elements are exchanged, mimicking the biological recombination between two single chromosome organisms. In inversion, the random continuous part with random length, of the element, is reversed. With the above two genetical operations alone we can make sure that the newly created element exhibits heredity. The relevance of the mutation operation comes with the fact that sometimes if the initial random population or population generated at a later point of time, does not exhibit variation, the GA can end up in a cycle, were it is creating same sets of elements repeatedly. Mutation is done to avoid this by altering a small random portion of the generated element. Usually mutation is performed on 1 -5% of the total generated elements to ensure good balance between heredity and variety.

1. **Replace and Repeat**

Finally, the generated population is replaced with the newly created one and steps 2 and 3 are repeated to generate the next generation of elements.

# Implementation

The Karel code generator is aimed to provide instruction sets for Karel to walk across the map. Even though this resembles to solving a maze problem, the fact that Karel can represent almost all computational problems just like a Turing machine justifies the use of evolutionary algorithms and the machine learning strategies over the traditional maze solving algorithms like DFS, BFS and A\*.

The implementation is done in two phases. The first phase is to define exactly what was the input/output format, specifics and how is the environment defined. This is important as this stands as the base set of information for both the algorithms and has to be the same for both the properly evaluation the applications of both these algorithms. Second is to define the strategy and implementation to generate the set of instructions for generating the input from the output. This was done using GA and Reinforced learning.

## Input and Output

The inputs and outputs processed are karel maps representing the initial state of the map with the start position of Karel in it and the final state of the map with the end position of Karel. A sample 3 x 3 input and output states of the Karel map will look like the following

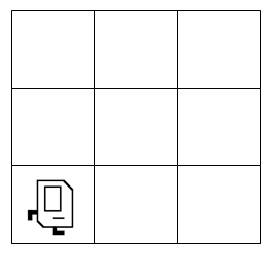


Figure 1: Initial state of Karel map

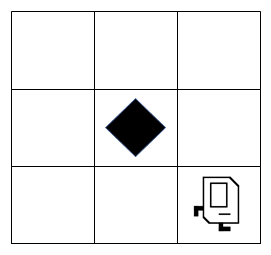


Figure 2: Final state of Karel map

The processing of the inputs and outputs are handled as a form of array. Empty spaces in the array are represented with zeros and the spaces occupied by beepers are represented with the number of beepers in that space. The format used for representing the Karel map as is as [0, 0, 0, 0, 0, 0, 0, 0]. Each element in the map is represented in this 6 digit array format for representing the following details

**X and Y Coord** The first two digits represent the x and y coordinate positions of the cell in the map. For an ‘n’ rowed and ‘m’ columned map, the cells are labelled as (0,0) from top left corner to (n,m) for the bottom right corner.

**Wall/ Obstacle** The first four digits represent the walls on left, down, right and up. ‘0’ in any of the first four digits means the wall is not there in the corresponding side of the cell and ‘1’ means it is there.

**Karel** The fifth element in the array represents whether Karel is there in that cell or not. The presence of Karel is represented using the letters, ‘L’/ ‘R’/ ‘U’/ ‘D’ (representing left, right, up and down) describing the direction in which Karel is pointing. 0 means Karel is not there in that cell.

**Beeper** The last element represents the presence of Beeper in the cell. ‘0’ means the beeper is absent and ‘1’ means it is there.

## Environment

The environment does the following

* Translates the instruction to map representation for initialing the map from the input Karel map state
* Executes the instruction set generated by the algorithm
* Translates the final state of the map to the Karel output map state

The environment of Karel plays a vital role in defining the map and testing the generated instruction set by executing the same. Once the input map array is received, the environment creates the actual map, with the set boundaries and position of Karel. The environment sends back the final version of the map, post executing the instructions.

## Implementation with Genetic Algorithm

Once the GA concept was clear, implementation with GA was straight forward. A random population of solutions need to be created at first, a fitness function should be defined for checking the accuracy of the solutions and a new set of population needs to created from the parents exhibiting desirable properties. The idea was to repeat the process until an actual result was created by the algorithm.

### Strategy

Initially a 5x5 Karel map was considered for running the algorithm. The strategy followed was to solve simple problems like making Karel move from one corner to the other, initially and eventually to move Karel across obstacles to reach the designated end point. To make things simple, the concept of gravity was removed from the world of Karel, hence the robot can be given instructions for free movement across the map.

By proving that, through GA, it is possible to generate programs in the Karel world, we are eventually proving that real world progrsams are also more or less completely generatable.

For defining the instructions a constraint was introduced on the length of the generated instruction set. This was introduced to avoid the issue of generation of large solutions in the population.

### Implementation

The algorithm was implemented through the following steps:

1. Initial population generation: The initial population generated was a set of random instruction solutions. This is purely random and sometimes include instructions like asking Karel to turn left and right alone without moving.
2. Fitness function: The fitness function implemented initially was ranking solutions based on how close is the final position of Karel after the execution of the solution to the expected destination. This was done by translating the input and output instructions arrays to Karel map and executing the instruction set in the population, to the Karel environment.

The fitness function was ranking the tiles based on proximity to the actual destination. Post ranking, the parents with desirable properties were selected and stored in an array in the order of selection. Solutions with undesirable properties were avoided during this process. This includes solutions which make Karel turn alone and does not move Karel.

1. Generation of new population: The new population was generated from the selected parent array from step 2. The first step was to convert the rank to probability. The parents with higher rank was selected more likely to be the next parent. The generation of solutions was done using the following genetical operations

* Crossover: Two selected solutions are broken into combinations of 2 and swapped and passed on as the child solution
* Inversion: A random length of the selected solution is inverted (last instruction will become first and so on) and passed on as the child solution.
* Direct-passdown: A randomly selected solution is passed down directly if it has a fitness value more than a certain threshold.
* Random-mutation: During the implementation process, the following issues were encountered.
* One entire generation had same set of elements
* Multiple generation of population got cyclic

To avoid this a randomly selected set of elements in a solution is mutated. The instructions in these elements are randomly chosen and replaced with another instruction. Mutation was implemented

These processes were repeated over and over again until the fitness function declared one solution as matching the required result.

## Using Reinforced Machine Learning

The static code analysis was implemented using the SonarQube server as it is supported by a wide group of developers and has support for around 20 languages including Java, Objective C and Swift.

### Strategy

### Implementation

## Standardized procedure for continuous testing

For implementing a standardized procedure for testing continuously, at first it is necessary

# Summary

*Ziehen Sie ein persönliches Fazit zu Ihrem Praxissemester sowie über die Tätigkeit im Unternehmen als solche. Gehen Sie auf die positiven und negativen Aspekte Ihrer Tätigkeit ein. Stellen Sie außerdem dem Mehrwert heraus, der sich für Ihre weitere berufliche Zukunft aus der Durchführung des Praxissemesters ergibt. Gehen Sie insbesondere auf die Sicherung der Nachhaltigkeit ein und erläutern Sie das Lessons Learned. Reflektieren Sie insbesondere das Praxissemester hinsichtlich der bisherigen Studieninhalte!*

**Umfang:** ca. 1 – 2 Seiten

# Future Work

Kurzweil, R. (2006). *The Singularity is Near: When Humans Transcend Biology.* Penguin.

Matej Balog, A. L. (2017). *DeepCoder: Learning to Write Programs.* University of Cambridge, Department of Engineering. ICLR. Retrieved from https://openreview.net/pdf?id=ByldLrqlx

Roberts, P. E. (September 2005). *Karel the Robot.* Von cs.stanford.edu: cs.stanford.edu/people/eroberts/karel-the-robot-learns-java.pdf abgerufen

Stuart Russell, P. N. (2010). *Artifical Intelligence: A Modern Approach (Third Edition).* Pearson.

Untch, R. H. (2015, 08 29). *Karel the Robot.* Retrieved from Computer Science, Middle Tennessee State University: https://cs.mtsu.edu/~untch/karel/

# Bibliography

# List of tools considered

**Xcode** The web development kit was developed from the Bill’s Kitchen  
(https://github.com/tknerr/bills-kitchen) software from git. It is a  
vagrant VM with docker installed in it. This was developed by Bosch and is used for web development in Windows OS.

**Cocoapods** It is a dependency manager for Swift and Objective-C Cocoa projects. It has over 28 thousand libraries and is used in over 1.7 million apps.

(*https://cocoapods.org/)*

**Pycharm** It is a software containerization platform. The containers wrap a  
piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries  
– anything that can be installed on a server

(*https://www.docker.com/).*

# Eidesstattliche Versicherung

**Name:** Rajan  
**Vorname:** Gokul  
**Matrikel-Nr.:** 01151615  
**Studiengang:** M.E Software Engineering for Industrial Applications

**gemäß §31 Abs. 7 RaPO**

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde nach meiner besten Kenntnis bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Hof, den Nov 10, 2017