Graph Search as a Feature in Imperative/Procedural Programming Languages

by

Christopher Henderson

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2018 by the
Graduate Supervisory Committee:

Ajay Bansal, Chair
Timothy Lindquist
Ruben Acuna

ARIZONA STATE UNIVERSITY

May 2018

ABSTRACT

Graph theory is a critical component of computer science and software engineering, with algorithms concerning graph traversal and comprehension powering much of the largest problems in both industry and research. Engineers and researchers often have an accurate view of their target graph, however they struggle to implement a correct, and efficient, search over that graph.

To facilitate rapid, correct, efficient, and intuitive development of graph based solutions we propose a new programming language construct - the search statement. Given a supra-root node, a procedure which determines the children of a given parent node, and optional definitions of the fail-fast acceptance or rejection of a solution, the search statement can conduct a search over any graph or network. Structurally, this statement is modelled after the common switch statement and is put into a largely imperative/procedural context to allow for immediate and intuitive development by most programmers. The Go programming language has been used as a foundation and proof-of-concept of the search statement. A Go compiler is provided which implements this construct.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

Chapter 1

DECLARATIVE PROGRAMMING

1.1 Section1

Declarative programming languages often ask only of the user that they provide a desire, or goal, that they wish to achieve. The operational semantics of that goal are out of the users hands as the languages runtime environment makes decisions that the user is not a part of.

Listing 1.1: Retrieve for me all employees whose name is Alice. SQL is perhaps the most well known declarative language on the planet. Exactly how this data is stored or how the system retrieves it is usually of little concern to the user.

```
1 SELECT *
2 FROM Employee
3 WHERE name =     A l i c e
```

That is, declarative languages allow for the easy definition of a goal, however defining how that goal is reached is often difficult, if not impossible.

Conversely imperative/procedural programming languages allow for the trivial definition of how to achieve a goal. These languages are stemmed from the tradition of Turing Machines and are the predominant languages used in both industry and academia. Outside of database management systems and artificial intelligence research, imperative/procedural languages dominate much of the world (although functional languages have been making a resurgence). For whatever the reason may be (whether it be psychological, machine performance, sheer cultural momentum, or something else altogether) these languages dominate and nearly every new programmers first language is a member of the imperative/procedural family tree.

However, those who delve into the world of declarative programming often emerge with a great appreciation for the paradigm. They begin to find joy in releasing themselves from the tyranny of manipulating physical memory and relish the transformation from what was a large, and questionable, procedure to a collection of small, and trivially true, facts.

If we are to use imperative/procedural languages, can we not adopt particular features from the declarative paradigm regardless? Afterall, users of imperative/procedural languages rarely argue for the purity of the paradigm. Indeed, the Rust programming language is a systems language which implements a classic declarative feature - pattern matching. [1] What else could the more traditional languages borrow from the world of declarative programming?

Of the long list list of features commonly found in declarative programming languages, we shall focus our discussion on automatic backtracking. Ultimately we shall attempt to port automatic backtracking into an already existing imperative/procedural programming language. Throughout the discussion we will frequently return to the NQueens [2] problem as the canonical example of a backtracking problem.

1.2 Section2

REFERENCES

APPENDIX A

RAW DATA