Chapter XXXV

Text Mining in Program Code

Alexander Dreweke
Friedrich-Alexander University Erlangen-Nuremberg, Germany

Ingrid Fischer
University of Konstanz, Germany

Tobias Werth
Friedrich-Alexander University Erlangen-Nuremberg, Germany

Marc Wörlein
Friedrich-Alexander University Erlangen-Nuremberg, Germany

ABSTRACT

Searching for frequent pieces in a database with some sort of text is a well-known problem. A special sort of text is program code as e.g. C++ or machine code for embedded systems. Filtering out duplicates in large software projects leads to more understandable programs and helps avoiding mistakes when reengineering the program. On embedded systems the size of the machine code is an important issue. To ensure small programs, duplicates must be avoided. Several different approaches for finding code duplicates based on the text representation of the code or on graphs representing the data and control flow of the program and graph mining algorithms.

INTRODUCTION

Computer programs are a special form of text. Words of a programming languages are combined to form correct sentences in this programming language. There exists a wide variety of programming languages, ranging from high-level object-oriented languages like Java or C++ to machine code, the language a processor can actually “understand”. Programming languages are usually translated with the help of compilers from high- to low-level. To produce this kind of “text” - the computer programs
Text Mining in Program Code

- is the daily work of many programmers; billions of lines of code have been written. Mostly, this code is not well documented and not really understood by anybody after the original programmer stopped working. Typically, many programmers are working on one project and often old code from former versions or other projects is used.

Duplicated code fragments are a special problem in big amounts of program code. These duplicated fragments can occur because of excessive use of “copy & paste”, because something was simply re-programmed or also because of the compiler. When translating from the high-level to intermediate or low-level languages, new duplicates can be introduced, e.g. by using code templates for instructions and instruction sequences.

Finding these duplicates has been in the focus of interest for many years. Code duplicates are called clones and clone detection has produced many different algorithms. If program code is simply viewed as text, clone detection is nothing else than mining in this text with the goal of finding the duplicate or similar code. Merging application areas and algorithms from the data mining community on the one hand and clone detection leads to fruitful new insights and results.

Finding duplicated code in programs can have different goals. First these duplicates can be visualized as a hint for programmers that something has to be done about this specific piece of code. Second, the redundant code can be replaced automatically by subroutine calls, in-lined procedure calls, and macros etc. that produce the same result. This leads to smaller code that is easier to understand or to maintain. Third, methods to detect and replace duplicated code can be integrated into compilers. Finally, finding duplicated code can lead to special hardware for the duplicates in the area of embedded systems.

In the case of program code, duplicates are not always “totally equivalent”. It is not only the one-to-one duplicate from a piece of code that is interesting. Also near duplicates or even pieces of code, that are syntactically different, but semantically equivalent must be found. E.g. in two fragments only two independent pieces of code having no side effect onto each other can be exchanged. Variable names can be different or registers in machine code can vary.

The application of clone detection ranges from high-level languages to machine code for embedded systems. The latter is the main topic in this chapter. The clone detection algorithms especially for embedded systems are described in detail.

Clone Detection and Embedded Systems

In recent years, many small programmable devices appeared on the market. PDAs, mobile phones, and navigation systems are examples for these systems. Typical software on these gadgets deals with speech or image processing, encryption, calendars, or address books. Therefore, the devices contain small processors, handling the different tasks.

But also in larger technical devices, such as cars, the number of small processors exploded in the last years. They offer e.g. efficient engine control, drivers assistance functions such as the “Electronic Stability Program” or “Line Departure Warning”, that warns the driver when he/she is leaving the right lane or more comfort functions such storing the different seat features for different persons. In modern high-end cars a network of up to sixty communicating processors are used.

These processors, also called embedded systems, are much more specialized in nature than general computing systems (e.g. desktop computers or laptops). Typically, they are produced at low cost, have low energy consumption, are small, and meet rigid time constraints for their tasks. Normally, they are manufactured in high quantities. These requirements place constraints on the underlying architecture. An embedded system consists of a processor core, a Read Only Memory (ROM), Random Access Memory...
Related Content

A Taxonomy for Distance-Based Spatial Join Queries
[www.igi-global.com/article/a-taxonomy-for-distance-based-spatial-join-queries/185656?camid=4v1a](www.igi-global.com/article/a-taxonomy-for-distance-based-spatial-join-queries/185656?camid=4v1a)

Lossless Reduction of Datacubes using Partitions
[www.igi-global.com/article/lossless-reduction-datacubes-using-partitions/1821?camid=4v1a](www.igi-global.com/article/lossless-reduction-datacubes-using-partitions/1821?camid=4v1a)

Networking Serendipitous Social Encounters in Urban Neighbourhoods
Marcus Foth (2009). *Social Implications of Data Mining and Information Privacy: Interdisciplinary Frameworks and Solutions* (pp. 71-87).
[www.igi-global.com/chapter/networking-serendipitous-social-encounters-urban/29145?camid=4v1a](www.igi-global.com/chapter/networking-serendipitous-social-encounters-urban/29145?camid=4v1a)

An Ant Colony Algorithm for Classification Rule Discovery
[www.igi-global.com/chapter/ant-colony-algorithm-classification-rule/7590?camid=4v1a](www.igi-global.com/chapter/ant-colony-algorithm-classification-rule/7590?camid=4v1a)