Re-animation of computer programs
Meijer, F.J.

Citation for published version (APA):
Meijer, F. J. (2001). Re-animation of computer programs

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
1 THE APPROACH TO THE CODE TRANSFORMATION PROBLEM

1.1 INTRODUCTION

In the early seventies the computers used by the department of the Economic Faculty of the 'Universiteit van Amsterdam' were housed in large rack filled cabinets. During maintenance, the technician could literally disappear into the computer. Extension of the computer, for example from 8 K of memory to the (at that time) enormous 32 K (its maximum value), was subject to substantial committee deliberations. Now these computers have ended in cabinets of another type: cellars, forgotten corners or even worse: destroyed. With the advent of large-scale integration (LSI) today's computers are much smaller in size, have greater storage capacity and process data faster.

Unless appreciable periods of time have elapsed old things are considered not worthwhile to collect. Examples are old or broken pottery and refuse. Collecting such items, or even showing them proudly to visitors is not considered as generally acceptable. However, if the broken pottery dates back to ancient times it becomes a collector's item. Methods of retaining the past have been developed throughout history. Museums collect what individuals normally cannot. Libraries offer access to historical information considered valuable and applicable to the present.

It is widely accepted that access to the riches of the past is of value. Specific expertise and practical experience have been developed for accessing them. Library science, musicology and archaeology are examples of this. Without such efforts an important loss of knowledge may occur. This thesis offers a possibility to avoid such a loss for what has been developed in the area of programming.

1 The department used a PDP/8 from Digital Equipment, Maynard, U.S.A.
1.2 A PARTICULAR CODE TRANSFORMATION PROBLEM

This thesis discusses the problem of accessing software in machine code written for an obsolete computer. The evolution of computer technology has taken place over a relatively short period. What was considered yesterday as state-of-the-art is now obsolete. But this rapid development also has its drawbacks. Programs have to be accommodated to the new equipment. Usually they cannot be used because of differences in the hardware. Software houses may adapt programs written for superseded computers if they are considered worthwhile. But, if the effort is not considered economically justified, this is not done. So a lot of old computer programs have become riches of the past because access to them requires extraordinary effort.

Not all programs developed in the past may be useful. Sometimes newer versions of old programs have fewer programming errors and provide more functions. To make buyers interested in new versions utility programs are often provided. This enables the old data to be used on the newer equipment. Programs or languages may have been discarded because of a change in interest. However, a number of old programs are still valuable. They offered interesting problem solving methods or are worthwhile being investigated because an ingenuous programming technique was applied. By not having access to old programs a loss of past knowledge that could help solve present problems may occur. Access to them can provide new and unexpected resources.

An example of a loss of past knowledge is the disappearance of the computer language ALGOL.\(^2\) It was a language, constraining its users to certain algorithms\(^3\), useful to solve a large number of problems. Another example is the high-level programming language (HLPL), called FOCAL, a HLPL of the interpreter type. It was specifically developed for Digital Equipment Corporation's PDP/8. FOCAL's operating system commands could be given, not only directly, but also in programs. This language was appropriate for educational purposes because program development could

---

\(^2\) ALGOL is an acronym for "ALGOrithmic Language".

\(^3\) An algorithm is the description of a pattern of behavior, expressed in terms of a well understood, finite repertoire of named ...actions of which it is assumed a priori that they can be done... (Walraet, p. 92)
take place at the terminal. Still another example is the PILOT-language. This language was developed to create teaching material. It constrained its users to a few simple commands for programming computer-assisted learning. Extensive experience with PILOT was acquired because the learning material had to be developed by teachers. This PILOT material is now inaccessible because no provisions were made to use it on new computers.

Obsolete programs can be accessed in different ways. One is emulation: the old program runs on a different platform than originally intended. The program and the operating system are emulated. The other is transformation. The original code is transformed in such a way that it can run on a different platform. The use of emulation is satisfactory for most purposes. The program runs, apart from speed differences, in the same way as on the old computer. Transformation is interesting for those who find it challenging to adapt old ideas to a new environment.

Transformation will make programs written in obsolete computer languages, such as PILOT, available to present users. They can inspect the code to see how past programming problems were solved or have the program function on a recent computer. Previously developed material can be embellished with features which now are considered standard such as: the use of menus, icons, mouse, color monitors, CD rom, etc.

The existence of “The Online Software Museum” demonstrates that there is much interest in conserving obsolete software. It has emulators available to run earlier developed software. It can run from a PC using Telnet. Some examples of software available at the museum are an Altair emulator with Microsoft BASIC and a CP/M system with applications such as WordStar and dBase. It is also possible to run this software if it is downloaded.

4 Some writers detested this method. The argument was based on the philosophy that programming is a mental activity and should be accomplished with pencil and paper, not at a terminal. Similarly, Schumann contended that you should not compose music at the piano keyboard.

5 http://elena.sysun.com/museum.index.html

6 ftp://elena.sysun.com/pub/sim
For the “LINUX” oriented enthusiasts software to emulate the initially developed home computers can be downloaded. One of the most advanced emulators is the Multi Arcade Machine Emulator/Multi Emulator Super System (MAME/MESS). It was created to run arcade games on personal computers. MESS uses the same framework as MAME, but is used to emulate the Commodore 64, BBC’s Color Computer and Texas Instruments’ TI-99/4A. The source code can be downloaded from Internet.\(^7\) It runs under X11 and compiles using LINUX.

In this thesis the difficulties in reconstructing and transforming machine code-programs will be addressed.

1.3 LEVELS OF DIFFICULTY

To make past knowledge accessible is difficult. This is particularly true when reconstructing machine code computer programs. Most literature on this subject discusses how to use old software available in higher-level language.\(^8\) Conversation with the user needs adaptation if there are natural differences. Re-use of software is amply discussed in computer literature. (Cullough, 1992, Feurer, 1984; Walraet, 1989; SCfT, 1990)

Programs may be available in machine, assembly or higher-level language. If an obsolete program is available in assembly or higher level language there is no code reconstruction problem. Transformation to another language is relatively simple. For this reason the more interesting problem of machine code transformation was investigated. According to some writers (Dahlstrand (1984), Horspool and Marovac (1980), Yoo (1985)) the difficulties in this case cannot be overcome. This may be true if a computer solution is only acceptable if the user cannot intervene. However, a computerized solution is feasible if the computer is considered as an agent performing this task together with a user. In defining a solution users are not excluded. They are required to actively participate in each step of the code transformation process. Computer science sometimes considers solu-

\(^7\) http://x.mame.net.

\(^8\) Often referred to as the re-engineering problem. See for example Biggerstaff and Perlis.
tions to programming problems that require intervention inadequate. Often users are even discouraged to intervene. Transformation of software, available only in machine code, can be accomplished when software permits users to intervene. This assumes that the user of such software is capable of intervening, if needed. Necessary skills are: knowledge of the assembly language of the obsolete computer, the target computer and their operating systems. Users can then decide whether or not they possess the necessary expertise to transform software.

When the source code of a program is available in a higher-level language a compiler\(^9\) can be written for a specific computer to transform the source code into machine code. The focus of this thesis is not the transformation of higher-level language programs into machine code, but the reverse: transformation of programs that are only available in a specific machine code. Programs in this language are far more restricted than those written in higher-level language. If a program is available in machine code it can only run on a specific machine. This is because a specific type computer only understands a specific type machine language. Solutions will be researched for the problem of transforming machine code for CP/M machines, using the Z80 processor. To accommodate this a number of programs have been written in assembly and higher-level language. This software will be referred to as the MAchine Language INvestigator (MALIN). It not only emulates but also reconstructs programs written for a specific obsolete computer, that are only available in machine language. It would be preferable to develop software that could transform any old program, irrespective of the type of computer it was written for, regardless of type. The problem with such a program is that its size would be unmanageable. Therefore, the research has been restricted to specific combinations of source and object computers. It is possible to adapt MALIN to other configurations.

A number of constraints have implicitly become apparent. One constraint is that software has been developed to run on a specific computer. Another is that it is designed for use with a specific combination of computers. Furthermore, the user is required to participate in transforming

---

\(^9\) A compiler is a program that translates higher-level language instructions into lower-level language, usually machine code.
old programs. This constraint needs further elucidation. Reconstruction of obsolete programs, only available in machine code, is a highly complex task. Machine code programs may use certain instructions in an unforeseen way. When this occurs users of MALIN must intervene. Intervention is needed if a difficulty is encountered. In such cases the package may need adaptation.

1.4 VALUE OF A SOLUTION TO THE PROBLEM

The proposed solution to the code transformation problem stresses accessing programs in machine language of a particular computer. A number of possible solutions will be discussed, including those that have been tried by others. The package, MALIN, can be seen as an interface to gain access to obsolete programs. When problems are encountered users can overcome them. The solution stresses access to the users ability to make changes to systems that support their activities. Such a solution has certain advantages over others that can be characterized as one directional: from the designer, to the user.

This thesis may have ancillary values. For example, it makes old software accessible to study the history of software development, it recreates useful support systems at a minimal cost and insights may be gained on the epistemological level with regard to problem solving by means of a computer.

The next chapter introduces some concepts in connection with transformation problems. Chapter three describes the history of the problem. Chapter four argues that a number of problems have to be solved if a program provides for code reconstruction and transformation from the old machine language to the language of the new computer. The most important problem to overcome is the interpretation of the meaning of the bit strings: what instructions had to be performed and which were the operands? This and related problems will be discussed in subsequent chapters.
1.5 OVERVIEW

This thesis is organized as follows:

- **Chapter 1** is an introduction to the problem. A particular code reconstruction problem is presented as the main object of research.
- **Chapter 2** introduces a number of useful transformation problem concepts. Depending upon the old program’s language three cases can be discerned: the program is only available in higher-level language, assembly language or machine code. In the latter case the problems encountered in code reconstruction are severe. Its solution is the primary objective of this thesis. Various routes can be taken to achieve this goal. The most promising route will be investigated.
- **Chapter 3** gives a brief history of the problem. Efforts to solve it are also addressed.
- **Chapter 4** examines the main problems encountered in solving the code reconstruction problem. The design of the software package will be discussed. Various possibilities are considered such as emulation and higher-level language transformation.
- **Chapter 5** introduces a number of criteria to investigate the quality of the proposed package, such as: ‘error free’, ‘architecturally correct’ and the notion ‘order of errors’. These distinctions proved helpful in solving problems that were encountered.
- **Chapter 6** discusses the development of the software reconstruction package. Examples of transformation of machine code are given.
- **Chapter 7** summarizes the solution to the code reconstruction problem. The results are discussed in relation to more general questions. User involvement during the design process and its importance is also addressed because designers are often constrained to their own view of the problem.