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.
2 THE CODE RECONSTRUCTION PROBLEM

2.1 INTRODUCTION

This chapter introduces some concepts necessary to understand code transformation. During the research process it became apparent that some definitions in computer literature were inadequate. In general, definitions are formulated to help understand a problem. On its own a definition is neither good nor bad. It has to be viewed in its context. Often it is more fruitful to provide for essential characteristics instead of giving definitions. For example, one important characteristic of a computer is that it generates event sequences. By stressing this characteristic of the computer the notion of transformation is easier to understand.

The language of the program strongly determines the severity of the code transformation problem. Its impact will be discussed. The various routes that may be taken to achieve transformation will be evaluated.

2.2 THE DEFINITION OF COMPUTER AND INFORMATION

There is no basic agreement in computer science literature about concepts such as information and computer. Liebenau and Backhouse (1990) enumerate in their Understanding information ten different definitions of the term information. Instead of defining they proposed identifying the elements that characterize it. Lavington (1980) points out that the notion computer has different meanings. Prior to 1940 the term computer applied to a manually operated calculating machine. Around 1940 this term was often preceded by 'automatic' to identify a fast calculating

1 The same applies to the notion 'software': As technology changes it becomes increasingly difficult to define the term 'software'.
2 Liebenau & Backhouse p 2.
device. Later, the term universal calculating machine was used. According to Lavington “these universal computers are the ones we would recognize today”. Kochenburger and Turcio (1974) state that the term computer can refer to different types of machines\(^3\). They state that “the computer is what its name implies: it works with digits just as we do when we solve arithmetic problems”\(^4\). These definitions of the computer do not help to understand the transformation problem. As Liebenau and Backhouse proposed, instead of trying to give a definition of the computer the elements that characterize it should be defined.

### 2.2.1 Main computer characteristic

The main characteristic of a computer is its ability to generate events. This thesis capitalizes on this characteristic when discussing transformation. Computers and humans generate event sequences as well. Both their human and mechanical generators are called *agents*. They can produce a multitude of events. By issuing commands, also called instructions, agents can generate specific series of events. Whereas it is very difficult for the human agent to repeat such series in exactly the same way, it is one of the main characteristics of the mechanical one. Without being able to exactly repeat event sequences the computer would be useless. A computer program consists of commands. It defines event sequences and the order in which they are executed. In view of this a program is considered faithfully transformed if the result of the event sequences executed on the object computer is the same as on the source. Therefore, in this thesis the main objective is to define those event sequences that are congruent for both computers. For every user the result of his actions should have the same effect.

Former attempts to solve the transformation problem considered it as primarily a translation problem. If the source code of the program was available there was basically no problem. If not, attempts to reconstruct the source code and to translate this into the language of the object computer had to be made. Another approach, circumventing the transformation problem, emulation, emerged in the accelerated environment. By

---

\(^3\) These types are: the analogue, the digital and the hybrid computer. In this thesis the notion ‘computer’ always applies to the digital one.

\(^4\) Kochenburger & Turcio, p 4.
using emulator software on the object computer it is possible to run programs developed for another, usually obsolete, system. The computer establishment considered the problem of how to use obsolete software on new equipment solved through use of an emulator.\(^5\)

### 2.2.2 The time aspect

A characteristic of the technological development of computers is that successive systems are faster and have more storage capacity. The accompanying software uses these properties. Software became more complex. A distinction must be made between commands (operators) and data (operands) when discussing the transformation problem. The computer's inability to make this distinction was identified in literature as a major obstacle (Dahlstrand, 1984, Horspool and Marovac, 1980). This task requires the invocation of many event sequences that would be difficult to accomplish without high-speed computers. Given sufficient time any sequence of events can be produced. Time constraints no longer apply to state-of-the-art computers. This made the solution to the code reconstruction problem feasible.

### 2.2.3 The language aspect

The notion transformation is related to language. Some writers (Catell, 1979 and Claus, 1987) see the transformation problem as one of translation. They believe the solution can be achieved by translating the machine code of the source program to an equivalent code for the object computer. Why direct translation does not provide a solution to the code reconstruction problem will be discussed in the next chapter. Instead, a one-to-one mapping between the series of events that are evoked in the source computer (M) to those of the object computer (M') is explored. To reconstruct old machine language programs on the object computer the program package, MALIN, is implemented on it. A thorough understanding of MALIN’s user language\(^6\) and the language of M and M' is essential for successful code reconstruction.

---

5 From Latin aemulator, he who strives after. According to The Concise Oxford Dictionary: emulate: try to equal or excel. Simulate: feign, pretend to have or feel.

6 A user language enables conversation between members of a specific group of users of a support system
2.2.3.1 Different language levels

The language level of the old program strongly determines the profoundness of the code transformation problem. If the program is available in a higher-level language it is relatively easy to achieve transformation into another high-level language, in contrast to if it is in machine language. It is important, therefore, to discuss the notion of language level.

When considering computer languages different levels are distinguished. Usually natural language is placed at the highest level of the scale. ‘Highest’ is not synonymous with ‘best’. In terms of grammatical and semantic constraints it could be considered as ‘worst’. At the other end of the scale is machine language. It is called a low level language. People state in *human activity language* (HAL) what they want to achieve. For example, a director of a corporation may want to know the current state of accounts. Before the development of computers the accountant would manually prepare the report. Today the computer accomplishes this activity. Preferably, the computer should interpret the HAL command, produce the current state of accounts, and take actions accordingly. However, this has yet to be achieved. Some basic HAL commands can be interpreted by the operating system, such as ‘format’, ‘copy’ and ‘type’. To couple statements made in HAL to patterns of events that help in problem solving, computer programs have to be written. This can be done at different language levels. For human agents it is tedious to issue commands at the lowest language level: machine language. Human agents prefer natural language. Therefore, more user-friendly languages are devised and implemented by defining those commands that invoke specific event sequences. Above machine language comes assembly language. It uses statements expressed in mnemonics such as MVI. This stands for “move immediate”. On the next level are the higher level programming languages. Some writers (Conway, 1958, Steel Jr, T.B., 1961) have tried to devise a universal higher level programming language. However, they are usually only problem-oriented. For example, administrative commands are expressed in the computer language COBOL (Common

---

7 Regarding computer languages different distinctions are possible. For example, functional and imperative. An assembler is an example of an imperative language. An example of a functional higher-level language is LISP.

8 JAVA and C++ are recent developments that proximate a universal language. They make it possible to cross the border between different platforms.
Business Oriented Language). For the formulation of mathematical commands ALGOL (Algorithmic Language) was devised.

Each element of a higher level programming language evokes a more complex pattern of events, such as: searching, printing, storing and retrieving. These can be further combined to create even more intricate patterns of events. Each set of sequences is defined to achieve the desired result. The process of producing statements that generate computer-produced series of events is defined as *program design* and the result is a *program*. A program selects, depending on certain conditions, commands that have to be executed. Each time a program is started an *event sequence* is initiated. A program describes different series of events and the way the sequences are produced by the computer. Similarities between such sequences of events exist and can be placed in classes. These classes have been commonly labeled as *word processing, error detection*, etc.

Commands issued to a computer in higher-level language have to be converted into code the computer can execute. The set of these codes has already been referred to as the lowest level of computer language: *machine language*. It applies to a particular type of computer. As a computer can only execute this type of code, programs are necessary that translate higher-level language program statements into machine code. An *assembler* translates a program written in assembly language into computer code. This language is machine-oriented. Each type of computer has its own assembly language. A *compiler* translates statements from a higher level programming language into machine code. It is also machine-oriented. Each type of computer needs a specific compiler for each higher-level language. However, the programming language is not confined to a specific computer.

It will be shown that the severity of the code transformation problem depends on the language in which the program is available. This then further leads to the distinction of different ways in which the problem can be solved. These ways will also be explored. Further, the code transformation problem has different aspects. Some of the more important will be addressed in the following sections.

### 2.3 HARDWARE AND SOFTWARE ASPECTS

The code transformation problem has a software and a hardware aspect.
Its hardware problem is formulated as follows: it must be possible to make a physical transport from the source computer that contains the software to be transformed to the object computer. This is the transfer or migration problem. In the beginning of the computer era the distinction between the technical and software aspects of the transformation problem was vague. Skelton speaks about migrability. (Skelton, 1986) The notions migrability and transferability are used when discussing the technical aspects of transformation. Moving a program from a tape to a disc is an example. The term reconstruction applies to the transformation of an old program into mnemonic code of the old computer. Little attention will be given to transfer technicalities. Difficulties emerging from using information carriers like: tapes and disks are not discussed. In what follows attention will be given to hardware differences between the source and object computers insofar as they influence the intricacy of the reconstruction problem. The main interest of this thesis is the development of a software package to access obsolete machine code programs and not its technical aspects.

2.3.1 Differences in hardware

The notion transformation is used to describe converting a computer program from one language into another. The code reconstruction problem is defined as transforming software in machine code (object code) into mnemonic code (source code). A special type of transformation is the conversion of source computer software into that of an object computer in such a way that the effect of the evoked event sequences is maintained. Different cases can be discerned according to the type of processors used. Three cases presented by Skelton are summarized below:

Case 1 Source and object computer are from the same producer and use the same processor.

It may seem that this presents no problems, as the code written for the source computer can be executed also on the object. However, there can be problems such as the:

a size of the main memory differs.
b I-O interfaces differ.
c versions of the operating systems differ.
d operating systems differ.
Case 2  The source and object computer use the same processor, but are made by a different manufacturer. When others copy a computer, called cloning, it can lead to the same problems as in case 1. Also, identical processors are used in different computers. In these cases differences between the operating systems are greater than with cloning.

Case 3  The processors of the source and object computer differ. This presents more intricate problems. It was selected for further investigation and resulted in the development of MALIN.

2.3.2 Differences in the language of the software

The transformation problem and its solution are not solely dependent on the type of processors/computers involved. The solution also depends on the language of the program to be transformed. Three cases can be discerned according to the nature of the code: higher-level language, assembler, and machine language.

Case A  The source program is available in a higher-level language. If, on the object computer, a program is available that translates the source code into object computer instructions there is no basic problem. However, manual adjustments may be necessary because:
1  Specific hardware aspects of the object computer must be accounted for.
2  The interpreters or compilers on the source and object computer may assign different meanings to the same source code. In computer jargon these different versions are called dialects.

Case B  A source version of the program is available in assembler. The translation of the program written for M, into code for M' causes no special problems. There must be a translator that converts the source code

---

9 In the past the numerous O.S.es created problems for users. For example Waites described how an attempt to transfer an IBM System 360 program to an identical machine failed. The operating systems differed. One machine used DOS, the other OS. According to him the same was true using SCOPE and KRONOS on Control Data 6000 machines. WAITE, W.M. System Interface, University of Colorado, USA in [Brown, 1977 b]

10 The 68000 processor for example was used in the Amiga, Atari, Macintosh, and Next computer.
of M into source code for M'. Manual adjustments may be needed for the same reasons as mentioned in Case A.

Case C The program is only available in machine language. This case presents the most problems. These are solvable for specific combinations of source and object computers. Programs can be written which indicate unambiguously which event sequences are to be executed by the object computer. Below is a chart depicting all the possible configurations.

\[
\begin{array}{ccc}
1 & 2 & 3 \\
\text{same make} & \text{different make} & \text{different make} \\
\text{same processor} & \text{same processor} & \text{different processor} \\
\hline
\text{A high level language} & \Delta & \Delta & \Delta \\
\text{B assembler} & \Delta & \Delta & \Delta \\
\text{C machine language} & \Delta & \Delta & \Downarrow \\
\end{array}
\]

\(\Delta = \text{possible}\)

\(\Downarrow = \text{possible with MALIN}\)

*figure 1  Possible configurations for transformation*

The above table describes different combinations of processors and language levels. Combination 3C is the focus of this thesis.

In the past researchers have given little attention to this configuration. It was not pursued because the technology required was virtually nonexistent and support of computer and software manufacturers was lacking. (Horspool and Morovac, 1980, Yoo, 1985). The combination 3C will be discussed further in the next chapter.

2.4 DIFFERENT ROUTES TO SOLVE THE TRANSFORMATION PROBLEM

To reiterate, the goal of this thesis is to reconstruct programs on the lowest level of computer language. To ensure that faithful reconstruction has been achieved it must be possible to transform the program into code for
another machine. To run a program on another machine than it was written for various routes can be taken. One is *emulation*. It entails defining event sequences for the object computer \( (M') \) by interpreting the instructions of the program of the source computer \( (M) \) during each run of the source program. The old computer is *simulated*. An interface is constructed between the obsolete program and the new computer that evokes all events necessary to make the old program function in the same way as on the source computer\(^{11}\). If the interpreted instructions of the source program function on \( M' \) as *if* running on \( M \), it is possible to port the program to \( M' \).

Another route concentrates upon the principle of code *disassembly*. This notion implies that machine code on \( M \) is disassembled and reassembled into machine code for \( M' \). Code disassembly is a means of producing a raw textual version of a program that is only available in machine code. After eventual adaptations of the raw output of the disassembler, the code is used to define event sequences on the new machine.

Still another route is *translation*. If the program is in machine code it is a difficult route as it involves reprogramming the old program in a higher-level language on \( M' \). Higher-level languages are able to retain more information than lower level ones. They can retain the objectives of the HAL on the object computer.

A choice has to be made between the routes mentioned above. Each has its advantages and disadvantages. The aim is to provide for computerized, not manual transformation.

Emulation provides a reasonable guarantee that the user can transfer old experiences to the new computer. It can be cumbersome when emulation forces the user to interact with a system that has unfamiliar properties. Code reconstruction, as discussed in this thesis, uses emulation as an intermediate step. The advantage of emulation is its ability to interpret and execute each instruction of the old program so it functions in the same way on the object computer as on the source. As a result of emulation, event sequences can be generated. Information about these can be stored in secondary storage. MALIN uses this information to transform

---

11 The National Programming Systems and Design Projects Library in Bulgaria was founded in 1973. One of its principles was to organize and maintain interaction to the original authors of programs wherever they might be employed. (Kalaidjieva, 1977)
the old code into one that invokes event sequences that have the same effect on the object computer.

When a program is reconstructed, for other than purely archaeological reasons, transformation of the instructions alone may be not sufficient. Adaptation may be required if the program must act as if it had been written for the new working environment\textsuperscript{12}. This will be discussed in a later chapter.

2.5 RECAPITULATION

This chapter gave an overview of a number of notions. The process of producing statements to generate a computer-produced series of events is called \textit{program design}. The result of the design is called a \textit{program}. A program determines a series of events that take place if it is executed on a computer. If executed, it produces the events designed to achieve the desired results as originally stated in the HAL. It represents the result of the reformatting from the HAL (source), to the language of a specific computer (object). If the HAL statements are no longer available because there is only a machine code version of the program, emulation makes it possible to retrieve information on the level of the user's activity language.

The notion event sequence highlights the problem. The code reconstruction problem, for which a solution is presented in this thesis, was placed in the context of event sequences. In the beginning of the computer era literature concentrated on the problem how to run programs written for a specific computer on another. Its solution was pursued along different lines. One was to find a solution for existing programs. Another was how to prevent problems from occurring with future ones. A third was emulation. At that time it was considered as inadequate because the user had to work in an unfamiliar working environment and execution was time consuming.

The notion of language had a fair amount of interest. The language of the program to be reconstructed determines the severity of the code reconstruction problem. It is difficult to solve if the program is only

\textsuperscript{12} Adaptation is not unusual in the translation of literary works. The reader is helped by the use of footnotes and comments.
available in machine language. One of the main characteristics of today's computers is its incredible processing speed. This makes it feasible to retrieve information from the old program to effect code reconstruction.

The history of the code reconstruction problem and early attempts to solve it will be discussed in the next chapter.