NASA-TM-103,845

NASA Technical Memorandum 103845

NASA-TM-103845 19940028400

# Translating an AI Application From Lisp to Ada – A Case Study

Gloria J. Davis

November 1991



National Aeronautics and
Space Administration

NASA Technical Memorandum 103845

# Translating an AI Application From Lisp to Ada – A Case Study

Gloria J. Davis, Ames Research Center, Moffett Field, California

November 1991

National Aeronautics and
Space Administration

**Ames Research Center**
Moffett Field, California 94035-1000

## Abstract

A set of benchmarks was developed to test the performance of a newly designed computer executing both Lisp and Ada. Among these was AutoClassII—a large Artificial Intelligence (AI) application written in Common Lisp. The extraction of a representative subset of this complex application was aided by a Lisp Code Analyzer (LCA). The LCA enabled rapid analysis of the code, putting it in a concise and functionally readable form. An equivalent benchmark was created in Ada through manual translation of the Lisp version. A comparison of the execution results of both programs across a variety of compiler-machine combinations indicate that line-by-line translation coupled with analysis of the initial code can produce relatively efficient and reusable target code.

## Introduction

The Department of Defense and NASA's Space Station Freedom Program Office specify that all future operational software will be developed in Ada. However, there are many AI programs currently written in Lisp that are targeted for use by these agencies. Current capabilities in automatic translation of Lisp to Ada do not as yet produce acceptable efficient, readable, and maintainable target code. This case study of manual translation of the newly created AutoClassII benchmark highlights some of the surrounding issues of translating code from Lisp into Ada.

In 1987, the Spaceborne VHSIC Multiprocessor System (SVMS) project was started at NASA Ames Research Center. The goal of this project was to build a space-qualified multiprocessor system that would execute Lisp and Ada efficiently. To evaluate the performance of the SVMS computer, a set of benchmarks was selected for implementation in both languages. Among the programs selected was AutoClassII, a large Lisp program.

To create a benchmark from AutoClassII, the functionality of the program was extracted and translated to produce an equivalent Ada version. A description of the approach, methods, issues, and results are given in this paper. The paper begins with a description of AutoClassII followed by several paradigms for translating Lisp programs to Ada. A description of the Lisp Code Analyzer, a tool developed to assist in the analysis of the original program along with the specific translation effort utilized in this study is then presented. Samples of the functions in both languages are compared along with the execution results of the two programs on several machines. The paper concludes with specific recommendations for future translation efforts.

## AutoClassII Description

AutoClassII is a large AI application program developed at NASA Ames Research Center by Peter Cheeseman et al. (ref. 1). The program's function is to determine classes in a data set automatically. Based on a Bayesian approach to classification, it has been used successfully to classify databases ranging from soybean plants and horse colic to infrared astronomic spectroscopy (IRAS). Recently, AutoClassII identified previously undetected classes of stars in an IRAS database, and these results appear in a recently published star catalog.

AutoClassII performs a two-phase mutual relaxation algorithm based on Bayes' theorem. The program searches for a specified maximum number of classes existing in a database and indicates the probability of each case in the database belonging to each class found. The program first initializes global parameters based on the database file. Each case in the database is described by the same number of attributes. The initial probabilities of each case existing in each class are randomly generated. Class models are then calculated based on case attributes and the initial class probabilities. At this point in the program, a classification-identifying indicator is presented. The model parameters and the class probabilities are updated each cycle of this mutual-relaxation algorithm using those class models and class probabilities most recently calculated. This two-phase cycle is repeated until the classification-identifying indicator has converged to a specified tolerance. The result describes the classes discovered and the probability of each case belonging to each of these classes. A more detailed description of the full system and the theory behind it are presented by Cheeseman et al. (refs. 1, 2).

## Benchmark Creation

The AutoClassII program is written in Common Lisp. In its original form, the program is not a good benchmark because the initialization phase contains a random probability generator, and the program utilizes a machine-dependent graphical interface. Therefore, both pieces of the program were eliminated leaving the input phase and the internal classification engine. This subset represented AutoClassII's basic functionality.

In order to produce a usable benchmark, it was necessary to ensure that a local change would not create unforeseen alterations in other sections of the code. The code was also checked to ensure that all essential functions were included so that the benchmark would run correctly and produce predictable results. Finally, all unnecessary user-interface code was removed. Although the excess code may not alter the program results, the execution speed may be affected.

To assist in understanding of AutoClassII program and to help organize the code into a readily understood form, the Lisp Code Analyzer (LCA) was developed. The function of this tool is to enable rapid understanding of a program and to ensure that the newly created benchmark is self-contained, complete, and concise.

The Lisp Code Analyzer, written in Common Lisp, was developed specifically for this effort to aid in the static analysis of Lisp code. The LCA accepts Lisp code as input, extracts the function names, and reports on their static frequency of use. From this information, the functions that are user-defined or members of the Common Lisp primitives can be determined. This also aids in identifying functions that are defined but never called. Many of the unused functions in the AutoClassII program were graphics-related. These were quickly identified by the report, and subsequently extracted. With all of the unused code removed, the remaining code formed the AutoClassII Lisp benchmark. Also revealed by the LCA is the functional control flow diagram of the benchmark, shown in figure 1. This pictorial representation of the hierarchical order of all functions was derived from the LCA report. This diagram was the key to understanding the program. Using this, a functional description of the Lisp benchmark was created. This document was used in a parallel effort by P. Collard. The functions are shown in the order that they are referenced, and from which functions they are called. The final benchmark program had 1/3 the lines of code of the original and was in a concise and easy-to-read form, ready for translation.

## Translation

A survey of automatic translators from Lisp to Ada code yielded only two candidates. Systems Research Laboratories, Inc (SRL) of Dayton, Ohio has a working translator/interpreter (ref. 3) that accepts Lisp code as input and produces Ada code as output. Though what is produced conforms to the Ada standard, it implements only 95 Lisp primitives, not all of which are Common Lisp compatible. Another candidate, the Lisp to Ada

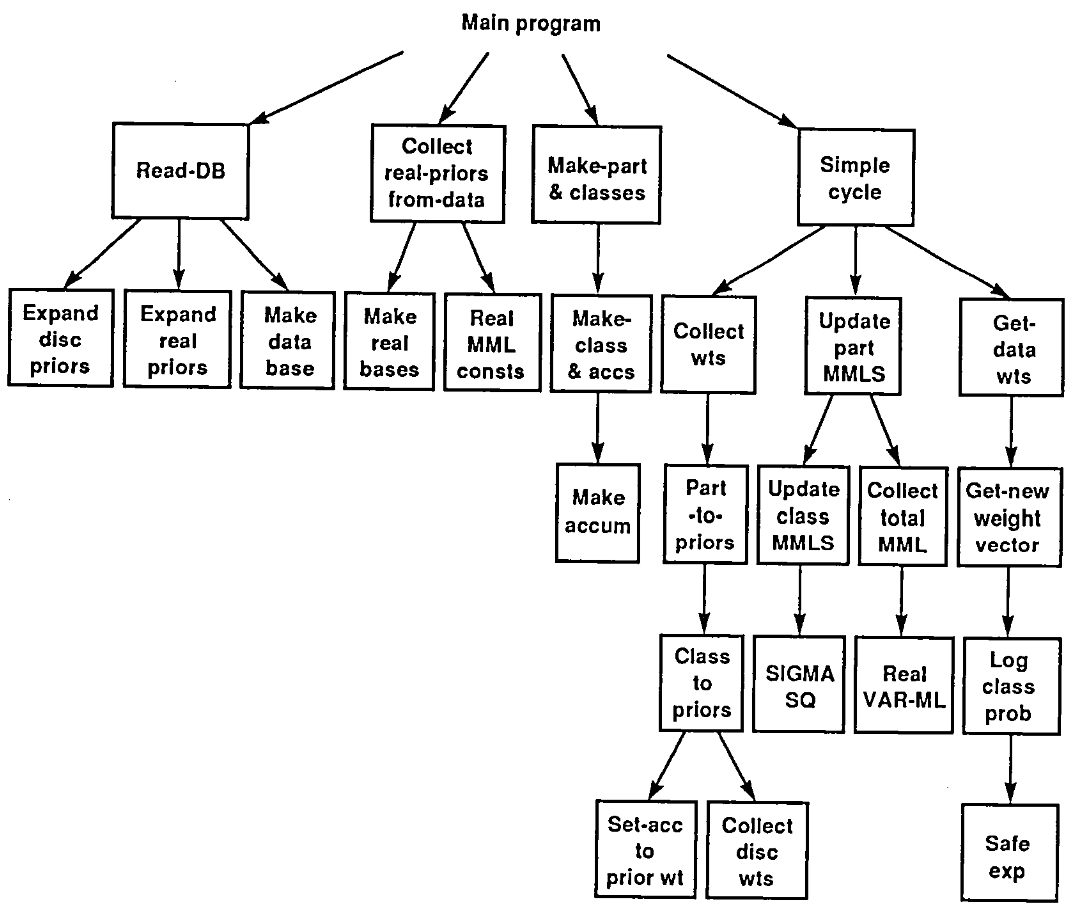


*Figure 1. Autoclass benchmark – functional diagram.*

"translator" developed by Intellimac, as referenced by Bughra (ref. 4), produces Ada code that resembles input for a list-processing interpreter, written in Ada. Neither of these efforts produce Ada code that is readable or maintainable, because of the inherent limitation on the number and types of primitives supported. As concluded by Wallis (ref. 5), program translation is performed so that an investment in existing software can be preserved, but if the generated code is not readable or maintainable, the previous investment may be lost.

Given the lack of usable automatic translators, the only alternative is manual translation. Two efforts were subsequently pursued and accomplished to translate AutoClassII from Lisp to Ada. A parallel Ada version was created using the functional description created with the LCA, as reported on by P. Collard (ref. 6), and the present effort, a direct line-by-line translation of the Lisp version, was undertaken. The remainder of this paper reports on this second approach.

Once the overall functional behavior of the program was analyzed, the translation was performed in two parts. First, due to the lack of data-typing in Lisp, an iterative process of identifying parameters and running the Lisp program to examine them at various points was performed to determine the type of the variables. Then a line-by-line translation was performed on the manipulations of the data-structures.

In the actual code translation, the two tasks that required the most effort were the generation of the data-structures, their definition and type, and the reading in of the input data. The AutoClassII program is designed to handle any of three different data types: integers, floating-point, and that which is to be ignored, which could be anything. These are all handled the same way in Lisp, as symbols, and readily coerced into their respective type. In order for the Ada version to process the same input, without any a priori knowledge of data type, all input data were read in one character at a time, and the strings, delimited by spaces, commas, and/or parentheses, were forced into their true type. Thus, the data structures had to be designed with the flexibility to handle any of the prospective types, and these types had to be determined as the data were read in from the database.

## Data-Structures

The two large record structures of AutoClassII, shown in figure 2, were defined by the **defstruct** construct in Lisp and linked together to be used as the primary data structure. Named "Partition" and "Class," they are lists

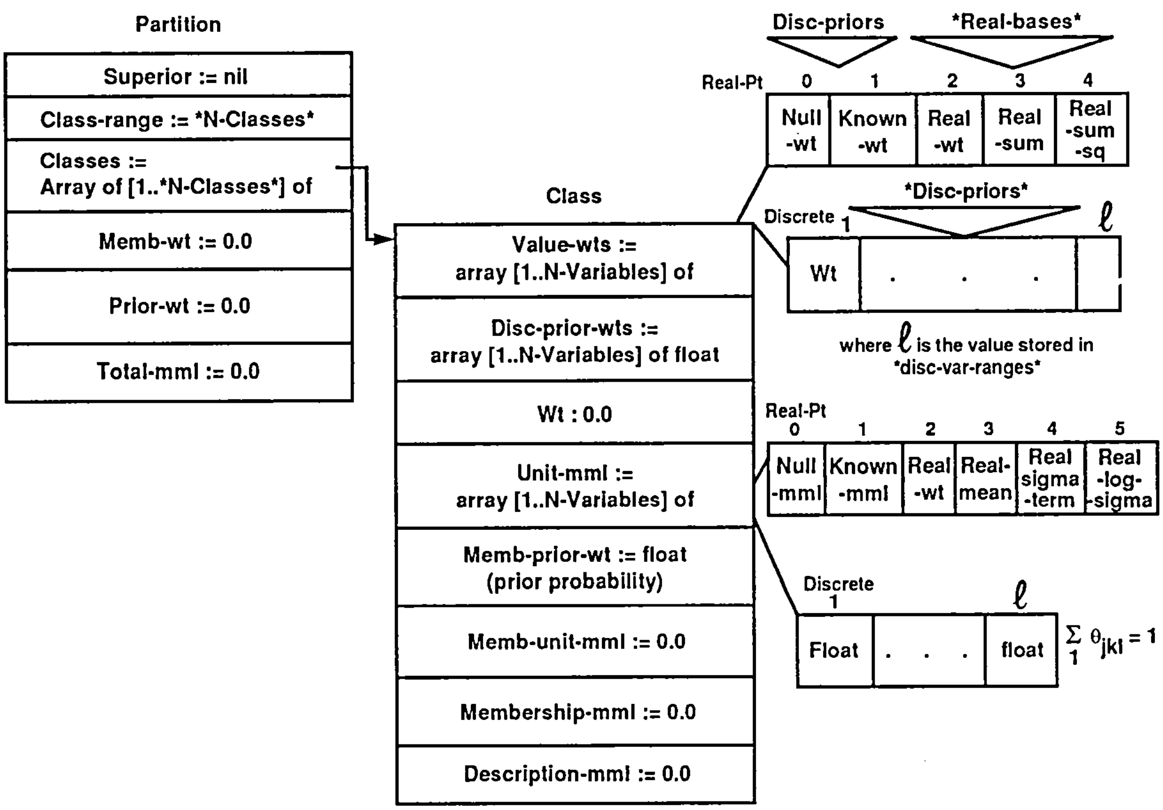


*Figure 2. Partition and classes structure.*

of lists. This same partition/classes structure was programmed in Ada using the discriminant and variant record constructs. The discriminant was used because of the variety of types of data (one of three) to be stored in the record and the variant was used because each type was stored differently. Different Ada compilers handle memory allocation differently. When using variant discriminant records, most compilers will allocate the maximum size possible for that record. Such is the case with the Symbolics Ada compiler. It attempts to allocate sufficient memory to hold the largest possible structure, but this can exceed the maximum amount allowed by the compiler for a single structure. Because of this, the partition/classes structure of the Lisp version had to be separated into two structures in Ada in order to eliminate the memory allocation problem.

The greatest programming effort of this translation was required in the input section of the initialization. This is where a basic difference in philosophy of the two languages created a major obstacle in the translation effort, that of strong data-typing in Ada versus the typelessness of Lisp. This difficulty was primarily due to the fact that the input to the program, without a priori knowledge of its type, had to be parsed a character at a time and assigned to its specific data type. In the entire AutoClassII Lisp program, reading the input data and setting up the data-structures was the only part that did not have a corresponding function that provided a direct model for translation. Only after this input text parser was developed was it possible to perform a line-by-line direct mapping of each function into Ada. This was accomplished by preserving the organization of the structures used by the Lisp version. This additional programmed parsing capability is reflected in the number of lines of code, presented in table 1. The total size of Ada code was 58% larger due to the global data-typing and the initialization phase of parsing the input text a character at a time. Also, for fair comparisons, the number of lines shown in the data-typing column only represent the global data-types and variables. Variables defined for local functions are reflected in the other sections. The entire translation effort was accomplished in 0.5 man-year.

**Table 1. AutoClassII benchmark lines of code**

| | Initialization | Data-typing | 2-phase alg. | Total |
|---|---|---|---|---|
| Lisp | 169 | 61 | 456 | 686 |
| Ada | 543 | 130 | 494 | 1167 |

## Comparison

The Lisp benchmark consisted of 686 lines and 22 user-defined functions, whereas the Ada translation was 1167 lines long and spanned 5 packages. This difference in size is primarily due to declarations, data typing, and package specifications in Ada. Performances of the two programs varied across several different platforms, as shown in table 2. Three different scenarios were tested. These vary in the number of cycles (c) through the mutual-relaxation algorithm and the number of data cases (n). The input data used was extracted from the IRAS database, each case described by 95 attributes of floating point values. The machines used in the tests were those available in the Advanced Architecture Testbed at the Information Sciences Division at NASA Ames Research Center. They represent a wide range of numeric and symbolic processing capabilities. A brief description of each machine used is as follows:

SYMBOLICS 3675 – a special purpose architecture designed for the efficient execution of Lisp. Both compilers used on this machine were developed by SYMBOLICS, specifically for this architecture.

MIPS R2000 – based on the MIPS' RISC chip, this is a 32-bit numeric machine.

Compaq 386/20e – this computer uses the i80386 chip, which is the baseline processor for the Space Station Freedom Data Management System.

IIM – another special purpose Lisp machine, it also has a floating point co-processor.

DEC 8800 – A 32-bit numeric-oriented multi-user minicomputer by Digital Equipment Corporation.

As can be seen, the special-purpose Lisp machines (Symbolics and IIM) executing the Lisp code had the best overall performance. The IIM was the most efficient because it has a math co-processor coupled with the symbolic processor. The performance of the Ada version on the MIPS machine indicates that efficiency can be retained in a translation effort. The slower timings are attributed more to the compilers than the actual code. Ada on the Symbolics machine yields poor performance, primarily because of the mismatch between the hardware architecture and the software virtual machine.

These performance comparisons indicate that, on this particular type of AI application, translation to Ada can be performed efficiently, yielding acceptable results, both in execution performance and maintainability of the target code. This is attributed to a combination of the mathematical nature of the program and the in-depth analysis done as part of the translation.

Table 2. AutoClassII benchmark execution time (secs)

| Machine | Compiler | c = 2 , n = 50 | c = 1 , n=531 | c = 2 , n=531 |
|---|---|---|---|---|
| Symbolics 3675 gc on, space 1 speed 1, safety 1 | SymbolicLisp | 48.14 | 203.96 | 315.93 |
| | SymbolicAda | 630.00 | 3867.0 | N/A |
| MIPS R2000 | Verdix Ada | 36.0 | 210.0 | 340.0 |
| Compaq 386/20e space 0, speed 3, safety 1 | Lucid Lisp | 433.29 | 3136.5 | 5106.3 |
| | Alsys Ada | 95.57 | 536.4 | 839.98 |
| Integrated Inference Machine (IIM) gc on | IIM Lisp | 29.10 | 131.30 | 203.00 |
| DEC 8800 | Franz Lisp | 391.08 | 2091.53 | 3369.12 |
| | Telesoft Ada | 75.65 | 497.20 | 657.55 |

## Conclusions

The paper presents the creation of a benchmark from an AI application program written in Lisp and its successful translation into Ada, as supported by execution results of each version on a variety of machines. The benchmark was created with the help of a Lisp Code Analyzer, also developed in this effort, that performed static analysis of the application automatically. After analysis of the resulting benchmark, the translation to Ada was rapidly achieved. The entire effort of benchmark creation, program analysis and translation into Ada was accomplished in 0.5 man-years. An automatic translator producing results comparable to this effort would require pre-translation analysis of the code. This analysis would need to be either built into the translator using heuristics or provided through extensive user interaction. This interaction would be with the designer of the original code or someone who had subsequently analyzed the sequential functional performance of the code and could represent this information in such a manner that could be processed by the translator.

The performance comparisons indicate that although this line-by-line, function-to-function translation to Ada was performed concisely, the actual program execution times of the Lisp and Ada implementations still differ widely across different machine-compiler combinations. This supports the notion that implementation of AI programs is not simply a language issue, but also depends substantially on the hardware and software environment in which the program resides.

## References

1. Cheeseman, Peter; Self, Matthew; Kelly, Jim; Taylor, Will; Freeman, Don; and Stutz, John: Bayesian Classification. AAAI88, Proc. Seventh National Conf. on Artificial Intelligence, St. Paul, MN, August 1988.

2. Cheeseman, Peter; Freeman, Don; Kelly, James; Self, Matthew; and Stutz, John: AutoClass: A Bayesian Classification System. Proc. Fifth International Conf. on Machine Learning, Ann Arbor, MI, June 1988.

3. Systems Research Laboratories, Inc., Ada Lisp User's Manual, Revision 4, April 1988.

4. Bughra, Paul; and Mudge, Trevor N.: Comparisons Between Ada and Lisp. U. Michigan, Research Report, 1985.

5. Wallis, P. J. L.: Automatic Language Conversion and its Place in the Translation to Ada. Proc. Ada Int. Conf., Paris, 1985, pp. 275-284.

6. Collard, Philippe; and Goforth, Andre: Ada as a Parallel Language for High Performance Computers: Experience and Results. Proc. Tri-Ada, Baltimore, MD, 1990, pp. 346-351.

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



**1. AGENCY USE ONLY** *(Leave blank)*

**2. REPORT DATE**
November 1991

**3. REPORT TYPE AND DATES COVERED**
Technical Memorandum

**4. TITLE AND SUBTITLE**
Translating an AI Application from Lisp to Ada—A Case Study

**5. FUNDING NUMBERS**
549-03-61

**6. AUTHOR(S)**
Gloria J. Davis

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Ames Research Center
Moffett Field, CA 94035-1000

**8. PERFORMING ORGANIZATION REPORT NUMBER**
A-91094

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
National Aeronautics and Space Administration
Washington, DC 20546-0001

**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
NASA TM-103845

**11. SUPPLEMENTARY NOTES**
Point of Contact: Gloria J. Davis, Ames Research Center, MS 244-4, Moffett Field, CA 94035-1000; (415) 604-4858 or FTS 464-4858

**12a. DISTRIBUTION/AVAILABILITY STATEMENT**
Unclassified — Unlimited
Subject Category 61

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** *(Maximum 200 words)*

A set of benchmarks was developed to test the performance of a newly designed computer executing both Lisp and Ada. Among these was AutoClassII—a large Artificial Intelligence (AI) application written in Common Lisp. The extraction of a representative subset of this complex application was aided by a Lisp Code Analyzer (LCA). The LCA enabled rapid analysis of the code, putting it in a concise and functionally readable form. An equivalent benchmark was created in Ada through manual translation of the Lisp version. A comparison of the execution results of both programs across a variety of compiler-machine combinations indicate that line-by-line translation coupled with analysis of the initial code can produce relatively efficient and reusable target code.

**14. SUBJECT TERMS**
Lisp, Ada, Benchmarks, Program translation

**15. NUMBER OF PAGES**
8

**16. PRICE CODE**
A02

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | | |

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102