Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

66

Neil D. Jones
Steven S. Muchnick

TEMPO: A Unified Treatment of Binding Time and Parameter Passin Concepts in Programming Languag

Springer-Verlag
Berlin Heidelberg New York 1978

Editorial Board
P. Brinch Hansen D. Gries C. Moler G. Seegmüller
J. Stoer N. Wirth

Authors
Neil D. Jones
Steven S. Muchnick
The University of Kansas
Department of Computer Science
18 Strong Hall
Lawrence, KS 66045/USA

AMS Subject Classifications (1970): 68A05, 68A30
CR Subject Classifications (1974): 4.22

ISBN 3-540-09085-1 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-09085-1 Springer-Verlag New York Heidelberg Berlin

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically those of translation, reprinting, re-use of illustrations, broadcasting, reproduction by photocopying machine or similar means, and storage in data banks. Under § 54 of the German Copyright Law where copies are made for other than private use, a fee is payable to the publisher, the amount of the fee to be determined by agreement with the publisher.

© by Springer-Verlag Berlin Heidelberg 1978
Printed in Germany

Printing and binding: Beltz Offsetdruck, Hemsbach/Bergstr.
2145/3140-543210

PREFACE

The design of TEMPO was motivated by difficulties encountered in teaching the semantic and pragmatic concepts of programming languages with current texts and languages. If the topic is taught as a comparative exposure to programming in several languages, a disproportionate amount of time must be spent teaching the irrelevant and trivial but exceedingly bothersome details of syntactic peculiarities, local hardware representations, and operating system interfaces. These variations of detail in turn obscure the essential issues--both the similarities and the differences in such basic areas as name-value binding, storage allocation, and procedure parameter passing. On the other hand it is difficult to be precise and concrete in teaching these concepts without reference to specific languages. TEMPO is designed to reconcile this need for precision with the problems inherent in the multiple language approach by providing first a base language known as the "dynamic version", characterized by virtually the latest possible binding times, and then a series of syntactic extensions and concurrent semantic restrictions which modify the language in the direction of earlier binding times and make greater implementation efficiency possible.

The language is useful in a variety of ways. Study of the language definition itself provides insight into the formal techniques for the specification of syntax and semantics. Hand simulation of the execution of a TEMPO program (or reading an annotated execution trace produced by an implementation) clarifies the ideas of information binding as to what is bound, and when and how binding occurs. The effect of a change in binding time discipline may be discovered by executing the same program in different versions of the language. Discussion of efficient implementation techniques made possible by the restrictions in the various versions makes it possible to observe the consequences of design decisions with respect to execution speed, compilability, runtime data structures, ease of programming, and so on.

To summarize the design goals of TEMPO, we have the following:

1. It must be precisely specified with respect to both syntax and semantics;

2. It must be as simple as possible, so complete exposition is practical;

3. It must allow binding times which are late enough to encompass relevant aspects of the behavior of such languages as APL and SNOBOL, yet be easily modifiable to produce versions with earlier binding times;

4. It must be completely unambiguous;

5. It should be a convenient and powerful programming language.

These goals, as might be expected, have numerous and far reaching consequences. The first and second dictate the omission of such features as nested statements and the *do* statement. While these are essential to the coherent structuring of programs, both may be straightforwardly expressed in terms of the statement types available in the language and contribute nothing to the understanding of binding time concepts. On the other hand, it is easy to envision such structural augmentations to TEMPO, either as syntax macros or as extensions to an implementation, and in fact we discuss such extensions in Appendix B.

The third goal was set to make it possible to discuss the more restricted versions of the language from a teleological viewpoint. On encountering a feature of TEMPO which is particularly inefficient for machine implementation, the student is moved to consider how to restrict the language just enough to provide a particular type or degree of efficiency. In this way concepts such as stack or static storage allocation and pre-execution type checking suggest themselves quite naturally.

The fourth goal is satisfied to a considerable degree by fulfilling the first, but it goes further as well. Thus procedure calls during expression evaluation are ruled out because of their interaction with the order of evaluation of the expression.

The fifth goal is viewed as secondary to the others. The language is as powerful as it can be in the theoretical sense of being universal, i.e., capable of expressing all algorithms, but it lacks some elements of convenience, as was noted in the discussion of the first and second goals above. A version of TEMPO with these structural conveniences and improved input/output facilities could easily prove to be a particularly powerful and versatile language for a variety of applications areas.

We gratefully acknowledge the thoughtful comments provided by James Arnold, Nigel Derrett, Michael Dyer, Margot Flowers, and Uwe Pleban on earlier versions of these notes. We also thank Susan Walker and Linda McClain for their excellent typing of the final version.

CONTENTS

1. Introduction 1

1.1. Overview 1

1.2. Bindings and Binding Times 1

1.3. Organization of This Volume 3

2. Examples from TEMPO and Some Current Programming Languages 5

2.1. A Simple Algorithm Expressed in Seven Different Languages 5

2.2. Some Features of TEMPO 16

2.2.1. Dynamic Data Structures 16

2.2.2. Symbolic Indirect Addressing 17

2.2.3. Dynamic Generation of Program Text 17

2.2.4. Procedure Parameter Substitution 18

3. Syntax of TEMPO 20

4. Semantics of TEMPO 23

4.1. Introduction and Informal Overview of TEMPO Semantics 23

4.2. Values of Variables 25

4.3. Snapshots and Segments 28

4.4. The Abstract Interpreter 34

4.4.1. Utility Routines 37

4.4.2. Routines to Handle Blocks and Scopes of Names 38

4.4.3. Expression Evaluation and Assignment 39

4.4.4. The IF Statement 42

4.4.5. The Goto Statement 43

4.4.6. Procedure Call and Return 44

5. Implementation Techniques for TEMPO 46

5.1. Semantics Versus Implementation 46

5.2. Linked Lists 47

5.3. The TEMPO Implementation Data Structures 48

5.4. The Program List 48

6. Machine Efficiency & Programmer Convenience 53

6.1. The Extremes--TEMPO versus FORTRAN 53

6.2. Trading Machine Efficiency for Programmer Convenience (and Vice Versa) 55

6.3. Sources of Inefficiency in TEMPO 56

7. Improvements to Increase Machine Efficiency 59

7.1. Overview 59

7.2. Storage Allocation 59

7.3. Creation and Manipulation of Program Text 67

7.4. Variable Names and Labels in the Snapshot 70

7.5. Data Types 77

7.6. Conditions for Compilability 80

8. Parameter Passing and Reference Variables 84

8.1. Procedures and Parameters 84

8.2. Reference Variables and Operations 84

8.3. Methods of Parameter Passing and Their Relative Efficiencies 88

8.4. Comparison of the Six Methods of Parameter Passing 93

8.5. The Dangling Reference Problem 94

9. Binding Times in Some Current Programming Languages 96

9.1. Introduction 96

9.2. Languages Designed for Efficient Execution: FORTRAN, COBOL, ALGOL 60, PASCAL 96

9.3. Multipurpose Languages: PL/I, ALGOL 68 97

9.4. Languages Designed for Programmer Convenience: APL, LISP, SNOBOL 98

9.5. Summary 100

10. Conclusions 103

10.1. Summary 103

10.2. Implications for the Design of Programming Languages 104

10.3. Further Topics in Programming Languages 106

Appendix A. Extended Backus-Naur Form Syntax Notation 108

Appendix B. TEMPO/SP - A Syntactically-Enriched Version of TEMPO for Structured Programming 111

References 116