

COMPUTER PROGRAM VERIFICATION: IMPROVEMENTS FOR HUMAN REASONING

By

Wayne David Heym, Ph.D.

The Ohio State University, 1995

Bruce W. Weide, Adviser

To ably create or modify computer programs that behave according to specification, programmers find it necessary to reason about their programs' behavior. We have formalized, in a direct, natural way, the informal pattern of reasoning generally used with programs written in modular, imperative languages. This formal system provides a solid basis against which to check the soundness and (relative) completeness of an informal reasoning method.

Formal proof that a program meets a specification can be done in this new system or in existing systems (e.g., calculating weakest preconditions using Hoare-style axioms or using symbolic execution). Each system prescribes a way to translate the program-specification pair to a mathematical assertion whose truth implies that the program satisfies the specification. Alternative systems are distinguished, however, by how well they fit programmers' informal reasoning methods. Programmers think of the effect that the execution of a given statement will have on variables' values, and they consider what conditions must hold for those values in each branch of the program. The new method is organized accordingly, unlike previous methods, which are organized for the convenience of mathematicians.

COMPUTER PROGRAM VERIFICATION:
IMPROVEMENTS FOR HUMAN REASONING

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the Graduate
School of The Ohio State University

By

Wayne David Heym, B.Phil., M.S., M.S.

* * * * *

The Ohio State University

1995

Dissertation Committee:

Bruce W. Weide

William F. Ogden

Stuart H. Zweben

Approved by

Adviser

Department of Computer
and Information Science

© Copyright by
Wayne David Heym
1995

For my loving wife, Kimberly Wells Heym

ACKNOWLEDGMENTS

My advisor, Bruce W. Weide, has guided and encouraged me through all the stages of graduate study, ever since my first visit to the campus. I am grateful for his wisdom in suggesting the topic for this dissertation. The number of helpful suggestions and ideas Bruce has offered in our regular meetings together is beyond counting, and his steadfast encouragement has helped me to persevere. I am thankful for William F. Ogden and his insightful, skeptical approach to my work. More than anyone else, he has guided the form of the proof rules, the explanation of the semantics, and the presentation of the proofs of soundness and relative completeness. Bill's hearty and healthy skepticism led the way to identifying the crucial ideas in the soundness proofs. Stuart H. Zweben encouraged me to work with him on testing and was instrumental in helping me learn most of what I know about the subject. He persevered in getting me to properly defend, with reference to the literature, the claim that my proposed new method, the indexed method of correctness proof, is more natural than the existing method. Stu's careful reading and corrections of early drafts were invaluable, and he completed his readings with amazing, and much appreciated, speed. These three professors regularly meet with a group of students interested in the effective production of quality software from the standpoint of reusability—the Reusable Software Research Group (RSRG) at The Ohio State University. The participants in RSRG have helped me learn many of the important issues in software engineering. This thesis has benefited greatly from the RSRG's intelligently dedicated effort to learn how to do software right.

We gratefully acknowledge the support of the National Science Foundation through grants CCR-9111892 and CCR-9311702; the Advanced Research Projects Agency under ARPA contract number F30602-93-C-0243, monitored by the USAF Material Command, Rome Laboratories, ARPA order number A714; and the Army Research Office through grant DAAH04-95-1-0457.

VITA

July 3, 1956	Born - Cleveland, Ohio
1978	B.Phil. Interdisciplinary Studies, Miami University
1980	M.S. Operations Research, Cornell University
1980-1983	Programmer/Analyst I, Eastman Kodak Company
1984-1988	Research Support Specialist, Cornell University
1988-1989	University Fellow, The Ohio State University
1989	M.S. Computer and Information Science, The Ohio State University
1989-1991	Graduate Teaching Associate, The Ohio State University
1991-1995	Graduate Research Associate, The Ohio State University
1994-1995	Instructor of Computer Science, Otterbein College

Publications

Research Publications

E. E. Ewing, W. D. Heym, E. J. Batutis, R. G. Snyder, M. Ben Khedher, K. P. Sandlan, and A. D. Turner. Modifications to the simulation model POTATO for use in New York. *Agricultural Systems*, 33(2):173–192, 1990.

W. D. Heym, E. E. Ewing, A. G. Nicholson, and K. P. Sandlan. Simulation by crop growth models of defoliation, derived from field estimates of percent defoliation. *Agricultural Systems*, 33(3):257–270, 1990.

S. H. Zweben, W. D. Heym, and J. Kimmich. Systematic testing of data abstractions based on software specifications. *The Journal of Software Testing, Verification and Reliability*, 1(4):39–55, 1992.

B. W. Weide, W. D. Heym, and W. F. Ogden. Procedure calls and local certifiability of component correctness. In *Sixth Annual Workshop on Software Reuse*, pages Weide– 1–6. In cooperation with the IEEE Computer Society Technical Committee on Software Engineering, November 1993.

S. H. Edwards, W. D. Heym, T. J. Long, M. Sitaraman, and B. W. Weide. Specifying components in RESOLVE. *Software Engineering Notes*, 19(4):29–51, Oct. 1994.

B. W. Weide, W. D. Heym, and J. E. Hollingsworth. Reverse engineering of legacy code exposed. In *Proceedings 17th International Conference on Software Engineering*, pages 327–331. ACM, Apr. 1995.

Fields of Study

Major Field: Computer and Information Science

Studies in:

Software Engineering	Prof. Bruce W. Weide
Programming Languages	Prof. Wolfgang W. Kuechlin
Theory of Computation	Prof. Timothy J. Long

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGMENTS	v
VITA	vii
LIST OF TABLES	xiii
LIST OF FIGURES	xv

CHAPTER	PAGE
I Introduction	1
1.1 How People Want to Reason About Program Behavior	3
1.2 Formal Bases for Reasoning	5
1.3 The Problem	8
1.4 The Thesis	9
1.5 Traditional Formal Reasoning Is Not Natural	9
1.5.1 An Example of Traditional Reasoning	11
1.5.2 An Example of More Natural Reasoning	13
1.5.3 Changing the Postcondition	24
1.5.4 Conclusions	26
1.6 Importance of Proving Soundness and Completeness	27
1.7 Outline of Dissertation	30
II Syntax and Semantics	31
2.1 Syntax	32
2.1.1 Aspects of the Syntax That Are Context-Free	32
2.1.2 Aspects of the Syntax That Are Not Context-Free	38

2.2	Semantics	41
2.2.1	Semantic Space	43
2.2.2	Semantic Definition	48
2.2.3	Validity	57
III	Proof Rules	61
3.1	Assertive Program Language Subsets	63
3.2	How the Rules are Defined	68
3.3	The Context Attribute	70
3.4	The Bridge Rule	72
3.5	The Rule for assume	78
3.6	The Rule for Procedure Call	80
3.7	Rules for Selection	83
3.8	The loop while Rule	89
3.9	The Rule for confirm	92
3.10	The Rule for Empty Guarded Blocks	95
3.11	The Rule for alter all	95
3.12	The Rule for Consecutive assume Statements	98
3.13	The assume-confirm Rule	103
3.14	The Rule for Consecutive confirm Statements	103
3.15	The Rule of Inference Bridging Predicate Logic and the Indexed Method	106
3.16	Example Application of the loop while Rule	107
3.17	Summary	110
IV	Soundness and Relative Completeness	111
4.1	Soundness	111
4.2	Relative Completeness	112
4.3	General Auxiliary Lemmas	119
4.3.1	Proof Rules Are Well-Formed	119
4.3.2	Factoring Statement Sequences	123
4.3.3	Shallow Lemmas	123
4.3.4	Deeper Lemmas	125
4.3.5	Negative-Branch-Condition Independence	138
4.3.6	Internal-Index Independence	153
4.4	Soundness Lemmas	154
4.5	Relative Completeness Lemmas	178

4.5.1	Related Valid Program Differing in Assertions Only	178
4.5.2	System of Rewrite Rules Is Terminating	178
4.5.3	Preservation of Invalidity in the Program Direction	181
4.5.4	The loop while Rule	191
4.5.5	The Procedure Call Rule	193
4.6	Relaxing a Simplifying Assumption	194
4.7	Summary	196
V	Conclusion	197
5.1	Informal and Formal Indexed Methods	197
5.2	Relationships Among Methods	198
5.3	Opportunities for Future Work	200
5.3.1	Miscellaneous Issues	200
5.3.2	The “ loop exit when ” Rule	201
5.3.3	Investigating the Worth of the Indexed Method	204
5.4	Contributions	205
	BIBLIOGRAPHY	207

LIST OF TABLES

TABLE		PAGE
1	Nonterminal Symbols Whose Definitions Are Assumed	33
2	Multipliers for Function Meas	179

LIST OF FIGURES

FIGURE		PAGE
1	Human Reasoning and Formal Bases	8
2	A Specification of Procedure Set_Maximum	10
3	An Implementation for Procedure Set_Maximum	11
4	Traditional Back Substitution Method: First Step	12
5	Traditional Back Substitution Method: Second Step	12
6	Traditional Back Substitution Method: Third Step	13
7	Traditional Back Substitution Method: Fourth Step	14
8	Traditional Back Substitution Method: Final Assertion	14
9	Indexed Method: Mark Between-Statement Spaces	15
10	Indexed Method: Mark Each Branch Condition	16
11	Indexed Method: Write Obligation at Last Position	17
12	Indexed Method: Replacing an Assignment Statement with a Fact . .	18
13	Indexed Method: Replacing the Second if-then Statement with Two Facts	19

14	Indexed Method: Replacing Statements Inside Branch Conditions with Facts	20
15	Indexed Method: Replacing the First if-then Statement with Two Facts	21
16	Indexed Method: Sequence of Facts and Obligations	22
17	Indexed Method: Final Assertion	23
18	Another Correct Implementation for Procedure Set_Maximum	25
19	An Improved Specification: Procedure Useful_Set_Max	25
20	Evidence of Unsoundness: First Step	28
21	A Specification of Procedure Stay_two	29
22	Evidence of Unsoundness: Second Step	29
23	Evidence of Unsoundness: Third Step	29
24	Context-free Grammar of Assertive Programs	39
25	Specification of Procedure Set_State_by_Addition	40
26	Specification of Procedure Add	41
27	Definition of Domains in the Semantic Space	45
28	Six Projection Functions on Environments	48
29	Notation for Environment Named “env”	49
30	Nested Loops and Old Variable Names	56
31	Transforming Programs to Mathematical Assertions in Phases	62
32	Context-free Grammar of Subsets of Assertive Programs	66

33	Grammar Productions Restricted for $\langle p_body \rangle$	67
34	Grammar Productions Restricted for $\langle in_code \rangle$, $\langle cd_prefix \rangle$, $\langle cd_suffix \rangle$, and $\langle cd_kern \rangle$ Inside Selection or Iteration, But Not Part of Procedure Body	67
35	Grammar Productions Restricted for $\langle in_code \rangle$, $\langle cd_prefix \rangle$, $\langle cd_suffix \rangle$, and $\langle cd_kern \rangle$ Outside Selection and Iteration, and Not Part of Proce- dure Body	67
36	Grammar Productions Restricted for $\langle top_lev_code \rangle$	68
37	Example Subgoal	70
38	Example Context	71
39	Application of Krone’s Procedure Declaration Rule	71
40	Equations Defining the Bridge Rule	73
41	Grammatic Derivation of Body of Change_X	74
42	Grammatic Derivation of Stows_Added(Body of Change_X)	75
43	Application of the Bridge Rule: $Inst(\mathcal{M})$	76
44	Equations Defining the Rule for assume	78
45	First Application of Rule for assume	79
46	Equations and Additional Syntactic Restriction Defining the Rule for Procedure Call	81
47	First Application of Procedure Call Rule	82
48	Equations Defining the Rule for Selection in the Absence of an else Clause	84

49	Definition of \mathcal{P} for the Rule for Selection in the Presence of an else Clause	85
50	Definition of \mathcal{M} for the Rule for Selection in the Presence of an else Clause	86
51	Second Application of Rule for assume	87
52	Application of Rule for Selection in the Presence of an else Clause	88
53	Definition of \mathcal{P} for the loop while Rule	90
54	Definition of \mathcal{M} for the loop while Rule	91
55	Equations Defining the Rule for confirm	93
56	Application of Rule for confirm	94
57	Third Application of Rule for assume	96
58	Second Application of Procedure Call Rule	97
59	Equations Defining the Rule for Empty Guarded Blocks	98
60	First Application of Rule for Empty Guarded Blocks	99
61	Application of Three Different Rules	100
62	Equations and Additional Syntactic Restriction Defining the Rule for alter all	101
63	Seven Applications of the Rule for alter all	101
64	Equations Defining the Rule for Consecutive assume Statements	102
65	Four Applications of the Rule for Consecutive assume Statements	102

66	Equations Defining the assume-confirm Rule	103
67	An Application of the assume-confirm Rule	104
68	Equations Defining the Rule for Consecutive confirm Statements . . .	104
69	An Application of the Rule for Consecutive confirm Statements . . .	105
70	Five Rule Applications	106
71	Mathematical Assertion in Context C'	107
72	Example: Iteration	108
73	Application of the loop while Rule	109
74	A Valid Assertive Program	113
75	An Invalid Assertive Program	115
76	An Assertive Program That Is Valid According to Functional Semantics	117
77	Environments Defined for \mathcal{P} of the Rule for Selection in the Presence of an else Clause	156
78	Environments Defined for \mathcal{M} of the Rule for Selection in the Presence of an else Clause	157
79	Environments Defined for \mathcal{P} of the Rule for Selection in the Absence of an else Clause	161
80	Environments Defined for \mathcal{M} of the Rule for Selection in the Absence of an else Clause	162
81	Environments Defined for \mathcal{P} of the loop while Rule	164
82	Environments Defined for \mathcal{M} of the loop while Rule	165

83	Environments Defined for \mathcal{P} and \mathcal{M} of the Rule for Procedure Call . . .	170
84	The Rule for a Procedure Call That Is Not the First Statement Within a whenever Statement	195
85	Action Directions Differ Among the Three Methods	199
86	Redefinition of $\langle \text{iter} \rangle$	201
87	Definition of \mathcal{P} for a Proof Rule That Would Handle the “ loop exit when ” Statement Having Exactly Two exit when Constructs	202
88	Definition of \mathcal{M} for a Proof Rule That Would Handle the “ loop exit when ” Statement Having Exactly Two exit when Constructs	203