

IMPROVEMENTS TO THE TRACE ASSERTION METHOD FOR SOFTWARE ENGINEERING

By

ANDREW C. P. DOOKHAN, B.Sc.Hon.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Engineering

McMaster University

© Copyright by Andrew Dookhan, January 1999

MASTER OF ENGINEERING (1999)
(Computer Engineering)

McMASTER UNIVERSITY
Hamilton, Ontario, Canada

TITLE: Improvements to the Trace Assertion Method for Software Engineering

AUTHOR: Andrew C. P. Dookhan, B.Sc.Hons. (Queen's University)

SUPERVISORS: Dr. David Lorge Parnas

NUMBER OF PAGES: xvii, 214

Abstract

Many formal methods have been suggested for documenting software work assignments, some are discussed in [19]. This thesis presents a new version of **Trace Assertion Method** [1,10]. We refer to this new version as **ITAM**. TAM originated in 1977, [1,2]. Although TAM is sound, TAM specifications seem inappropriately complex. The complexity of a specification is not proportional to the complexity of the concepts being described. In this thesis we reintroduce TAM to document software work assignments, (i.e. module interface specifications).

This thesis suggests improvements to TAM (ITAM) for programmers and specification writers. In doing this we :-

- (a) allow specification method writers to represent the state of an object, using sets.
- (b) view the collection of objects implemented by one module, as a single object (refer to as a *primary* object,) which may be composed of other objects.
- (c) presents objects' state representation semantics and a syntax that is more in tune with programmers' perceptions.
- (d) provide a tool that checks the syntax describing the canonical representation of an object's state.
- (e) provide commonly used predefined auxiliary functions to reduce redundant mathematical notation.
- (f) provide predefined auxiliary functions to be used when it is necessary to "check type" and "check availability" of an object, before performing operations on an object.
- (g) provide a method of abbreviating, long and duplicate, mathematical expressions.
- (h) use tables which can be easily checked for completeness and consistency, when we

describe state changes of a module's objects, caused by invocations of that module's programs, (*operation tables*).

(i) show that it is possible for ITAM to document polymorphic and non-deterministic module specifications.

(j) provide a common compact and systematic document format for ITAM's module interface specifications.

Acknowledgements

I would like to express my sincere thanks to Dr. David L. Parnas for the guidance, freedom of thought and the tremendous support he has given me throughout this thesis.

I would also like to thank Dr. Mark Lawford and Dr. Ryszard Janicki for their helpful comments, suggestion and solutions after reading the thesis, Dr. Martin von Mohrenschildt for his comments and help with the syntax checking tool, Dr. Emil Sekerinski and Dr. Ridha Khedri for their mathematical input, Dennis Peters for our long discussions, Cui Feng, Li Puili for being at the pre-seminar rehearsal, and the other graduate students of McMaster Software Engineering Research Group for their comments and discussions.

My appreciation goes to the staff members Doris Burns, Cheryl Gies for their help with some administrative matters during my work on this thesis.

I would like to express my deepest appreciation to Linda Cox, Ron Dolson, Dr. Irene Siotis and Dr. David Parnas, an invaluable team, who provided a special kind of support that helped me to maintain my focus on this thesis. Special thanks to Jane Schouten for her moral support.

Finally, I would like to acknowledge the Natural Sciences and Engineering Research Council of Canada, Bell Canada, the Telecommunications Research Institute of Ontario (TRIO/CITO) and DEC for the use of resources that they provided.

I would like to dedicate this thesis to my dearest sisters Bernice, Jennifer and her two children Bendita and Antoine, my mother Ruby and my brother Trevor.

Table of Contents

Abstract	iii
Acknowledgements	v
Table of Contents	vi
List of Figures	xvi
List of Tables	xvii
CHAPTER 1: INTRODUCTION	1
1.0 Thesis organisation	1
1.1 TAM and some basic definitions	1
1.1.1 A view of software construction	2
1.1.2 What is an algorithm (program)	2
1.1.3 Variables/Objects and Types/Classes	2
1.1.4 Modules and Access Programs	3
1.1.5 Modules and Objects	3
1.1.6 Documenting Software	4
1.1.6.1 Specifications vs. Descriptions	4
1.1.6.2 Documenting programs	5

1.1.6.3 Documenting Modules and Objects	6
1.1.7 Composition	6
1.2 What is a Trace Assertion Method	6
1.2.1 Detailed explanation of a "trace"	8
1.2.2 What is TAM's relation to algebraic methods	9
1.2.3 A brief history of the trace assertion method	10
1.2.3.1 Other related TAM work	11
1.2.4 The essential characteristics of trace assertion methods	12
1.2.4.1 Use of a canonical representation	12
1.2.4.2 Systematic Descriptions	13
1.2.4.3 An older TAM format	14
1.3 Motivation for a new TAM (ITAM)	18
1.3.1 Canonical Representations	18
1.3.2 Canonical Rep. for a bounded "multi-object" module	21
1.3.3 Use of Program Function and Other Tables	22
1.3.4 Parameterised Specifications	22
1.3.5 Using "Legality" to reduce table size	23
1.3.6 Using abbreviations to reduce table size	24
1.3.7 Using auxiliary relations for further reductions	24

1.3.8 Other format changes to improve reading	25
CHAPTER 2: CANONICAL REPRESENTATIONS	26
2.1 Semantics for ITAM module specifications	26
2.1.1 Terminology and notation	26
2.1.1.1 Abbreviations, new primitive types and the TYPE relation	27
2.1.2 Representing an object's state in a module's specification	28
2.1.3 Sets used to represent an object's state in module specifications	28
2.1.3.1 SP	29
2.1.3.2 PSET	29
2.2 The PSET structure–name tree	29
2.2.1 Naming structures for sets of ordered pairs	31
2.3 The syntax for state representations in module specifications	33
2.3.1 SP and PSET	33
2.3.1.1 SP syntax	34
2.3.1.1.1 Explicit SP set	34
2.3.1.1.2 SUBP	35
2.3.1.2 PSET structure names and syntax	35
2.3.1.2.1 SSET	36
2.3.1.2.2 NSET	36

2.3.1.2.2.1 ARRAY	36
2.3.1.2.3 REC	37
2.3.2 Summary of notation for UDT descriptions	38
2.4 A new rule, old relations, connectives, new notation and relations for ITAM ...	39
2.4.1 Common set relations, set operations, and predicate connectives	39
2.4.2 Operations on string and real	40
2.4.3 New common set notation and relations	42
2.4.4 New PSET predicates	43
2.4.5 Naming PSET's predefined non-predicate auxiliary functions	44
2.4.5.1 New PSET functions that are not predicates	46
2.4.5.1.1 Predefined auxiliary functions for ARRAY operations	47
2.4.5.1.2 Operations on ARRAYs	48
2.4.5.2 Reducing the number of arguments for "del", "add" and "sub"	49
2.4.6 Other predefined functions	49
2.4.7 Predefined relations which are not functions	50
CHAPTER 3: A SIMPLE GRAMMAR FOR CANONICAL REPs.	51
3.1 Introduction to the grammar for describing the Canonical Representation	51
3.2 Trace and set syntax for a canonical representation	52
3.2.1 An example using traces	52

3.2.2 The revised example, bounded and using set representation	52
3.2.2.1 Revised example using our notation for syntax testing purposes	53
3.3 Setting precedence	53
3.4 Convention	55
3.4.1 Grammar for describing the Canonical Representation	55
3.4.2 Grammar tool’s terminal symbols	60
3.5 Grammar testing tool for the canonical representation description section	63
CHAPTER 4: A FORMAT FOR ITAM DOCUMENTS	64
4.1 The four sections of the ITAM module specification	64
4.2 The general format	64
4.2.1 Notable format changes from older TAMs	64
4.2.2 Skeleton of format	65
4.2.3 Systematic format rules and other notable format changes	67
4.2.3.1 Header Section	67
4.2.3.2 Canonical Representation Section	68
4.2.3.3 Syntax Section	68
4.2.3.4 Operation Tables Section	69
CHAPTER 5: DISCUSSION OF EXAMPLES	71
5.1 Introduction to examples of Appendices A and B	71

5.2 Examples of Appendix A	72
5.2.1 Integer stack module examples	72
5.2.1.1 Demonstrating the use of ARRAY	73
5.2.1.2 Argument, "rep"	73
5.2.1.3 Returned values and Extension class	73
5.2.1.4 Module parameter and External parameter	74
5.2.1.5 Considering the TYPE relation	74
5.2.1.6 Polymorphic PUSHCHK	74
5.2.1.7 Predefined auxiliary functions	74
5.2.1.7.1 Not providing predefine auxiliary function's arguments	74
5.2.2 Set of stacks, module examples	75
5.2.2.1 Where to define Auxiliary functions in ITAM's format	75
5.2.2.2 SSET and cross products	75
5.2.2.3 Using the predefined auxiliary function N_fnd	76
5.2.2.4 A return value that is not a simple 1-tuple	76
5.2.2.5 Notation: NC, NA, CA	76
5.2.2.6 Local versus predefined auxiliary function	77
5.2.3 Integer queue module examples	78
5.2.4 Bounded integer table list module example	78

5.2.4.1	Demonstrating the use of REC	78
5.2.4.2	Abbreviations	78
5.2.4.3	Transition from a trace of program invocations to a set of objects	79
5.2.4.4	Applying the New Rule of section 2.4	80
5.2.4.5	Different status for different conditions	80
5.2.5	Unique integer producer module examples	80
5.2.5.1	Demonstrating the use of SUBP	80
5.2.5.2	ITAM table organisation for return values in terms of after-state	81
5.2.6	Bounded priority integer queue module examples	82
5.2.6.1	Syntax when using SNE, +, \+, -, \-	82
5.2.6.2	Intermediate abbreviation, function calls in abbreviations	82
5.2.6.3	Some ways of placing bounds on this module	82
5.2.6.4	Comments in a specification	84
5.2.6.5	Demonstrating how abbreviations reduce table size	84
5.2.6.6	Syntax for indexing objects	85
5.2.6.7	Demonstrating the use of +, \+, -, \- in access programs	85
5.2.6.8	Non-deterministic property of this module	85
5.2.7	Bounded multiple binary integer tree module examples	86
5.2.7.1	External Type	86

5.2.7.2	An explicit set	86
5.2.7.3	A recurring syntax for dynamic structures	87
5.2.7.4	Tcard, Int, Gtr functions	87
5.2.7.5	Demonstrating how status reduce table size	87
5.2.7.6	Abbreviations, cost of reducing operation tables and output tuples	88
5.2.8	Bounded path holder module example	88
5.2.8.1	Interchanging the Syntax and Canonical Representation sections	89
5.3	Examples of Appendix B	89
5.3.1	A nondeterministic single object, room module example	89
5.3.1.1	Explicit sets and their effect on parameters	89
5.3.1.2	Demonstrating set union syntax	90
5.3.2	A polymorphic, two-rooms module example	90
5.3.3	A non-deterministic polymorphic, two-rooms module example	90
5.3.3.1	Eliminating rows of an operation table	91
5.3.4	A bounded polymorphic multi-object, non-empty room module example	91
	CHAPTER 6: CONCLUSION AND FUTURE WORK	92
6.1	Conclusion	92
6.2	Future work	93
6.2.1	Possible extensions to this work in the context of reverse engineering	93

6.2.2 Automatically checking a module specification	94
6.2.2.1 Format checking tools	94
6.2.2.2 Completeness and consistency checks	95
6.2.3 Canonical Representation sets, Abbreviations, Aliases and Macros	95
6.2.4 A trace simulator for deterministic and non-deterministic ITAM modules . . .	95
APPENDIX A: OLDER EXAMPLES	97
Example 1a: Unbounded integer stack module. (Figure 1)	97
Example 1b: Overflow integer stack module. (Figure 4)	100
Example 1c: Overflow integer stack module with push check.	103
Example 2a: An unbounded set of unbounded stacks. (Figure 2)	106
Example 2b: A bounded set of bounded stacks.	112
Example 3a: Unbounded integer queue module. (Figure 3)	120
Example 3b: Overflow integer queue module.	123
Example 3c: Overflow integer queue module with insert check.	126
Example 4: Bounded integer table list module. (Figure 5)	129
Example 5a: Unbounded unique integer producer module. (Figure 6)	136
Example 5b: Bounded unique integer producer module.	138
Example 5c: Bounded unique integer producer module.	140
Example 6a : Bounded priority integer queue module. (Figure 7)	142
Example 6b : Bounded priority integer queue module.	147

Example 6c : Bounded priority integer queue module.	153
Example 7a: Bounded multiple binary integer tree module. (Figure 8)	159
Example 7b: Bounded multiple binary integer tree module.	171
Example 8: Bounded path holder module. (Figure 9)	180
APPENDIX B: ADDITIONAL EXAMPLES	185
Example 9: A nondeterministic single object, room module.	185
Example 10: A polymorphic, two-rooms module.	188
Example 11: A non-deterministic polymorphic, two-rooms module.	194
Example 12: A bounded polymorphic multi-object, non-empty room module.	202
APPENDIX C: USING THE CANONICAL REP. GRAMMAR TOOL	207
Explaining the general grammar	207
The grammar checking tool for our system	207
REFERENCES	211

List of Figures

Fig 1. A tree of the PSET's structure names base.	31
Fig 2. A tree of predefined auxiliary PSET function names.	44
Fig 3. Operation Table STKSTATE of Example 2a	77
Fig 4. Operation Table STKSTATE of Example 2b	77
Fig 5. Converting n in trace to "ptr" in canonical representation	79
Fig 6. Return value in terms of an object's after-state	81
Fig 7. An object's after-state in terms of return value	81
Fig 8. H1 of INSERT operation table of 6b and 6c	84
Fig 9. A possible addition to the abbreviation tables of 6b and 6c	85
Fig 10. A single row H1 for the INSERT operation table	85
Fig 11. The "predicate of non-determinism"	86
Fig 12. An abbreviated tuple's element	88
Fig 13. Another representation of PUTO of Example 11	91

List of Tables

Table 1. A Meta table from TAM's format, section (3)	14
Table 2. Equivalence T.POP of stack example using Table 1.	14
Table 3. Common type definition syntax	34
Table 4. Meaning of an SP set using typical math syntax	34
Table 5. Meaning of SUBP syntax	35
Table 6. Useful PSET structure—names and sets	35
Table 7. Meaning of SSET syntax	36
Table 8. Meaning of NSET syntax	36
Table 9. Meaning of ARRAY syntax	36
Table 10. Meaning of REC syntax	37
Table 11. UDT description notation	38
Table 12. Common allowable predicates on set, S	39
Table 13. Common allowable set relations	40
Table 14. Common connectives and notational conveniences for predicates	40
Table 15. Allowable predicates on strings	40
Table 16. Allowable real operations	41
Table 17. Allowable predicates on reals	41

Table 18. New common set notation	42
Table 19. New common set predicates	42
Table 20. New predicates for a PSET, S	43
Table 21. Terminal Symbols	63
Table 22. Tabular format for a function definition	68
Table 23. Other operation tables object state entries	70
Table 24. Translation of familiar symbols for text testing	209