



UNIVERSITEIT VAN PRETORIA  
UNIVERSITY OF PRETORIA  
YUNIBESITHI YA PRETORIA

# FINE-GRAIN TRANSFORMATIONS FOR REFACTORING

By  
EMMAD I. M. SAADEH

## THESIS

Submitted in partial fulfillment of the requirements  
for the degree of Philosophiae Doctor (Computer Science) in the Faculty of  
Engineering, Built Environment and Information Technology of the  
University of Pretoria, 2009

Pretoria, South Africa

# FINE-GRAIN TRANSFORMATIONS FOR REFACTORING

By: Emmad I. M. Saadeh  
Supervisor: Prof. Derrick Kourie  
Department: Computer Science  
Degree: PhD (Computer Science)

## ABSTRACT

This thesis proposes a new approach to formalize refactorings, principally at the UML class diagram design level (but incorporating a limited amount of code-level information—basic *access-related* information). A set of *abstract* and *atomic* fine-grain transformations (FGTs) is defined as prototypical building blocks for constructing refactorings. The semantics of each FGT is specified in terms of its pre- and postcondition conjuncts. Various logical relationships between FGT pre- and postcondition conjuncts are fully catalogued. These include uni- and bi-directional sequential dependency relationships; absorbing and cancelling reduction relationships; and uni- and bi-directional conflict relationships.

The principle container for FGTs is an FGT-list in which the ordering of FGTs respects the sequential relationships between them. Such a list is characterised by the set of FGT precondition conjuncts (which a system should satisfy if the FGTs are to be sequentially applied to the system) as well as the resulting postcondition conjuncts (that describe the effect of applying the list). In the thesis, twenty-nine commonly used primitive refactorings are specified as such FGT-lists, together with their associated FGT-enabling precondition conjuncts. Refactoring-level pre- and postconditions are also identified for each primitive refactoring FGT-list. These are, of course, required to guarantee behaviour preservation.

An alternative container for FGTs is defined, called an FGT-DAG. It is a directed acyclic graph with FGTs as nodes, and with arcs that reflect the sequential dependency relationships between constituent FGTs. An algorithm is provided to convert a list of FGTs into a corresponding set of FGT-DAGs. Thus design level refactorings specified as FGT-lists can be also be converted to corresponding sets of FGT-DAGs. The precondition for applying such a refactoring to a given system is specified at two levels: the FGT-enabling precondition conjuncts that apply to each FGT-DAG, and the refactoring-level precondition conjuncts.

The thesis provides various algorithms that operate on FGT-DAGs. These include an algorithm to remove redundancies from an FGT-DAG. It also includes algorithms that operate on the

elements of a set of FGT-DAGs: to detect sequential dependencies between these elements, to detect whether they are in deadlock, and to detect and possibly remove or modify FGTs causing conflicts between them. In addition, an algorithm is provided to build composite refactorings from primitive refactorings. It indicates how composite-level and FGT-enabling precondition conjuncts can be derived and utilised to avoid the rollback problem.

A Prolog prototype FGT-based refactoring tool has been implemented. The tool stores all of the above-mentioned catalogued information as Prolog rules and facts. This includes the twenty-nine commonly used primitive refactorings (stored as Prolog FGT-lists) and their associated refactoring-level pre- and postcondition conjuncts. The tool also implements all the previously mentioned algorithms as Prolog procedures.

The thesis thus establishes the foundations for a tool in which end users can create (and apply without rollback) not only composite refactorings, but also completely new refactorings whose semantics is constrained only by the fine-grained semantics of FGTs, rather than by the more course-grained semantics of primitive refactorings.

Furthermore, using FGTs as refactoring building blocks (i.e. instead of primitive refactorings) means that redundancies and conflicts can be more accurately pin-pointed and removed; and opportunities for parallel execution are exposed at a more fine-grained level. These advantages come at the cost of having to carry out more computations because analysis has to take place at the FGT-level rather than at the refactoring-level.



To my father, mother & my wife for their encouragement and support.

## Acknowledgments

First, I am thankful to God for having granted me the skills, power, patience, and opportunities that made this possible in spite of all the other commitments and responsibilities that I had.

I would like to thank my supervisor, Prof. Derrick Kourie, for his support, guidance and patience along these years, and for providing constant direction. He has been a source of encouragement and inspiration. I learned a lot from the feedback he gave me. I was amazed by both his insights and his stamina. He invested his most valuable resource on my behalf: his time.

I owe thanks to my colleagues in the Espresso Research Group and in the Computer Science Department in University of Pretoria for all the encouragements during the different stages of this work.

I thank my parents, brothers and sisters for their support. I especially appreciated my father's words of encouragement. I thank my children Ibraheem, Marah, and Adam for many encouraging times together. I owe them all the time I spent to accomplish this work.

I am especially thankful to my wife Hadeel for her love, support, and patient throughout this entire process. I really thank her for all the responsibilities she took on behalf of me to give me a chance to do my research. Without her, I never would have made it through the program. As a wife and mother, she picked up the slack and encouraged me in an extraordinary way. She deserves an award at least as valuable as my PhD.

# TABLE OF CONTENTS

	Page
ABSTRACT.....	i
ACKNOWLEDGEMENTS.....	iv
TABLE OF CONTENTS.....	v
LIST OF FIGURES.....	x
LIST OF TABLES.....	xii
LIST OF ALGORITHMS.....	xiii
I Prologue	
CHAPTER	
1. INTRODUCTION.....	2
1.1 The Problem.....	2
1.2 The Proposed Formalism.....	7
1.3 Thesis Overview.....	9
2. REFACTORING—STATE OF THE ART.....	11
2.1 Software Evolution.....	11
2.2 Refactoring.....	12
2.2.1 Codes Level.....	13
2.2.1.1 Non-Object-Oriented Programming Languages.....	13
2.2.1.2 Object-Oriented Programming Languages.....	13
2.2.2 Design Level Models.....	14
2.2.3 Database Schemas Level.....	16
2.2.4 Software Architectural Level.....	16
2.2.5 Software Requirements Level.....	16
2.3 Formalisms.....	17
2.3.1 Graph Transformations.....	17
2.3.2 Pre- and Postcondition.....	17
2.3.3 Program Slicing.....	18
2.3.4 Formal Concept Analysis.....	18
II The Approach	
3. LOGIC-BASED REPRESENTATION.....	20
3.1 Introduction.....	20
3.2 Object Element Logic-Terms.....	23
3.3 Relation Element Logic-Terms.....	25
3.4 Example.....	27
3.5 Reflection on this Chapter.....	29

4. FGT-BASED APPROACH.....	30
4.1 Introduction.....	30
4.2 Fine-Grain Transformations (FGTs).....	30
4.2.1 Object Element FGTs.....	36
4.2.1.1 addObject FGT.....	36
4.2.1.2 renameObject FGT.....	40
4.2.1.3 changeOAMode FGT.....	42
4.2.1.4 changeODefType FGT.....	46
4.2.1.5 deleteObject FGT.....	48
4.2.2 Relational Element FGTs.....	51
4.2.2.1 addRelation FGT.....	51
4.2.2.2 renameRelation FGT.....	57
4.2.2.3 deleteRelation FGT.....	57
4.3 FGT Sequential Dependency.....	59
4.3.1 Definition.....	60
4.3.2 Uni-Directional Sequential Dependencies.....	61
4.3.3 Bi-Directional Sequential Dependency.....	62
4.3.4 Mapping Feasible FGT-Lists to FGT-DAGs.....	63
4.4 FGTs for Primitive and Composite Refactorings.....	67
4.4.1 Definitions.....	67
4.4.2 FGT-Enabling Preconditions in an FGT-DAG.....	69
4.4.3 FGTs and Primitive Refactorings Preconditions.....	70
4.4.3 Applying Refactorings.....	72
4.5 Reflection on this Chapter.....	72
5. PRIMITIVE REFACTORINGS AS FGT COLLECTIONS.....	73
5.1 Introduction.....	73
5.2 Add Element Refactorings.....	76
5.2.1 addClass.....	76
5.2.2 addMethod.....	76
5.2.3 addAttribute.....	76
5.2.4 addParameter.....	76
5.2.5 addGetter.....	76
5.2.6 addSetter.....	78
5.3 Change Element Refactorings.....	80
5.3.1 Changing Characteristics.....	80
5.3.1.1 renameClass.....	80
5.3.1.2 renameMethod.....	80
5.3.1.3 renameAttribute.....	80
5.3.1.4 renameParameter.....	80
5.3.1.5 changeClassAccess.....	80
5.3.1.6 changeMethodAccess.....	80
5.3.1.7 changeAttributeAccess.....	81
5.3.1.8 changeMethodReturnType.....	81
5.3.1.9 changeAttributeDefType.....	81
5.3.1.10 changeParameterDefType.....	81
5.3.2 Change Structure (Restructuring).....	81

5.3.2.1	changeSuper	81
5.3.2.2	moveMethod	82
5.3.2.3	moveAttribute	85
5.3.2.4	attributeReadsToMethodCall	87
5.3.2.5	attributeWritesToMethodCall	88
5.3.2.6	pullUpMethod	89
5.3.2.7	pushDownMethod	91
5.3.2.8	pullUpAttribute	92
5.3.2.9	pushDownAttribute	94
5.4	Delete Element Refactorings	95
5.4.1	deleteClass	95
5.4.2	deleteMethod	96
5.4.3	deleteAttribute	96
5.4.4	deleteParameter	96
5.5	Reflection on this Chapter	97
6.	MOTIVATED EXAMPLE	98
6.1	LAN Simulation	98
6.2	Logic-Based Representation	100
6.3	encapsulateAttribute Refactoring	101
6.4	createClass Refactoring	104
6.5	pullUpMethod Refactoring	105
6.6	LAN after Refactorings	106
<b>III Features Of The Approach</b>		
7.	REDUNDANCY REMOVAL	111
7.1	Introduction	111
7.2	Absorbing Reduction	112
7.3	Cancelling Reduction	116
7.4	Advantages of Reduction Process	118
7.5	Reduction Algorithm	119
7.6	Example	121
7.7	Efficiency Considerations	123
8.	DETECTING AND RESOLVING CONFLICTS	124
8.1	Introduction	124
8.2	Conflicts in FGT-Based Approach	126
8.3	FGT's Conflicts-Pairs	130
8.3.1	Bi-Directional Conflict	130
8.3.2	Uni-Directional Conflict	136
8.4	Conflict Algorithm	138
8.5	LAN Motivated Example	142
8.6	Reflections on Conflicts	143
9.	SEQUENTIAL DEPENDENCY BETWEEN REFACTORINGS	144
9.1	Introduction	144
9.2	Sequential Dependency in Previous Approaches	145



9.3 Sequential Dependency between FGT-Based Refactorings	148
9.4 Sequential Dependency Algorithm	150
9.5 Deadlock Problem	153
9.6 LAN Motivated Example	157
<b>10. COMPOSITE REFACTORINGS</b>	<b>158</b>
10.1 Introduction	158
10.2 FGT-based Composite Refactoring	160
10.3 Examples	165
10.3.1 encapsulateAttribute Composite Refactoring	165
10.3.2 enh-pullUpAttribute Composite Refactoring	168
10.4 Reflection on this Chapter	173
<b>11. PARALLELIZING OPPORTUNITIES</b>	<b>174</b>
11.1 Introduction	174
11.2 Parallelizing Opportunities	175
11.3 Reflection on Parallelization	176
<b>12. NEW REFACTORINGS</b>	<b>177</b>
12.1 Introduction	177
12.2 Example	178
12.3 New Refactorings in the FGT-Based Approach	180
12.4 Reflection on this Chapter	182
 <b>IV Epilogue</b>	
<b>13. CONCLUSIONS</b>	<b>184</b>
13.1 Summary	184
13.2 Conclusions	186
13.3 Future Work	189
 <b>V Appendix</b>	
<b>A. FGT SEQUENTIAL DEPENDENCY</b>	<b>192</b>
A.1 Uni-Directional Sequential Dependencies	192
A.2 Bi-Directional FGTs Sequential Dependencies	194
<b>B. PRIMITIVE REFACTORINGS AS FGT SEQUENCES</b>	<b>195</b>
B.1 Add Element Refactorings	195
B.1.1 addClass	195
B.1.2 addMethod	196
B.1.3 addAttribute	197
B.1.4 addParameter	197
B.2 Rename Element Refactorings	198
B.2.1 renameClass	198
B.2.2 renameMethod	199
B.2.3 renameAttribute	200
B.2.4 renameParameter	200
B.3 Change Characteristics Refactorings	201



B.3.1	changeClassAccess	201
B.3.2	changeMethodAccess	202
B.3.3	changeAttributeAccess	202
B.3.4	changeMethodReturnType	203
B.3.5	changeAttributeDefType	204
B.3.6	changeParameterDefType	205
B.4	Delete Element Refactorings	205
B.4.1	deleteMethod	205
B.4.2	deleteAttribute	206
B.4.3	deleteParameter	207
BIBLIOGRAPHY		208

# LIST OF FIGURES

	Page
1.1 pullUpMethod Refactoring: (a) before refactoring, (b) after refactoring.....	3
1.2 Refactorings as black box.....	4
1.3 Refactorings as hard coded sequence of statements.....	5
1.4 Refactoring as a set of FGT-DAGs.....	7
1.5 Simplified UML meta-model.....	9
3.1 A simple UML class diagram of the <i>SimpleSys</i> .....	27
3.2 A code-level implementation of the <i>SimpleSys</i> .....	27
3.3 Underlying logic representations of the <i>SimpleSys</i> .....	29
4.1 Potential sequential dependencies between FGTs.....	61
4.2 FGT-DAGs of refactoring X.....	67
4.3 Primitive, composite refactorings and FGTs.....	69
4.4 Primitive refactoring different considerations.....	71
5.1 Class A before and after addGetter(A.x).....	77
5.2 Class A before and after addSetter(A.x).....	79
5.3 Class A & B before and after moveMethod(B.m, A, [int]).....	83
5.4 Class A & B before and after moveAttribute(A.x, B).....	85
6.1 A UML class diagram of the LAN simulation before refactoring.....	99
6.2 A code-level implementation of the LAN simulation before refactoring.....	99
6.3 Underlying logic representations of the LAN simulation before refactoring.....	101
6.4 Packet & Workstation classes before and after encapsulateAttribute refactoring.....	104
6.5 Underlying logic representations of the LAN simulation after refactorings.....	108
6.6 A UML class diagram of the LAN simulation after refactoring.....	109
6.7 A code-level implementation of the LAN simulation example after refactoring.....	109
7.1 Part of the reduction facts as implemented in Prolog.....	118
7.2 Reduction inside refactoring.....	122
7.3 Refactoring X after reduction.....	123
8.1 Conflict between refactorings $R_i$ & $R_j$ .....	125
8.2 Conflicts detection in FGT-based approach.....	126
8.3 Possible conflicts between FGTs.....	129
8.4 A Selection of fgtConflict facts as implemented in Prolog.....	129
8.5 A simplified UML class diagram of a college system.....	130
8.6 Conflict detection & resolving algorithm.....	140
8.7 Conflicts between refactorings moveMethod & pullUpMethod.....	143
9.1 Sequential dependency between refactorings $R_i$ & $R_j$ .....	145
9.2 Ambiguous sequential dependency.....	147
9.3 Sequential dependency in FGT-based approach.....	149
9.4 Refactoring Directed Acyclic Graphs (REF-DAGs).....	151
9.5 Deadlock problem.....	154
9.6 Sequential dependency between refactorings createClass & pullUpMethod.....	157
10.1 Straightforward approach.....	159
10.2 Composite refactoring in composite preconditions approaches.....	160



10.3 Composite refactoring in FGT approach.....	164
10.4 A simplified UML class diagram of a <i>college</i> system.....	165
10.5 encapsulateAttribute composite refactoring.....	166
10.6 encapsulateAttribute composite refactoring in FGT approach.....	168
10.7 A simplified UML class diagram. (a) before and (b) after refactoring.....	169
10.8 enh-pullUpAttribute composite refactoring.....	172
12.1 Part of the LAN system's class diagram.....	178
12.2 Part of the LAN system's class diagram after enh-pullUpMethod.....	181

## LIST OF TABLES

	Page
4.1 Primitive refactorings.....	68
6.1 encapsulateAttribute refactoring.....	103
6.2 createClass refactoring.....	105
6.3 pullUpMethod refactoring.....	106
7.1 Absorbing reduction.....	114
7.2 Cancelling reduction.....	117
8.1 Bi-directional FGT conflict-pairs.....	131
8.2 Uni-directional FGT conflict-pairs.....	136
10.1 encapsulateAttribute refactoring.....	167
10.2 enh-pullUpAttribute refactoring.....	170
13.1 A comparison between FGTs-based and alternative formalisms.....	188

# LIST OF ALGORITHMS

	Page
4.1 Building FGT-DAGs algorithm .....	65
7.1 Reduction algorithm.....	120
8.1 Conflict detection & resolving algorithm .....	139
9.1 Sequential dependency algorithm .....	152
9.2 Deadlock detection algorithm.....	156