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

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

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

LIST OF FIGURES

Page

LIST Of TABLES

Page

LIST OF ALGORITHMS

Page