

Abstractions for Language-Independent Program Transformations

Karl Trygve Kalleberg



Thesis for the degree of Philosophiae Doctor (PhD) at the
University of Bergen

2007-05-11

ISBN 978-82-308-0441-4
Bergen, Norway 2007
Copyright Karl Trygve Kalleberg
Produced by: Allkopi Bergen

Abstractions for Language-Independent Program Transformations

Karl Trygve Kalleberg
Department of Informatics



Thesis for the degree of Philosophiae Doctor (PhD) at the
University of Bergen

2007-05-11

Contents

Acknowledgements	xi
I Introduction	1
1 Introduction	3
1.1 Software Evolution	4
1.2 Program Transformation	4
1.2.1 Strategic Programming	5
1.3 Program Models	6
1.4 Language Abstractions for Program Transformations	7
1.4.1 Extensible Languages	8
1.5 Method	9
1.6 Contributions	9
1.7 Outline	10
1.8 Summary	12
II Software Transformation Systems	13
2 Programmable Software Transformation Systems	15
2.1 Software Transformation Systems	15
2.1.1 Anatomy of a Transformation System	16
2.1.2 Features of Software Transformation Systems	17
2.2 Feature Models	18
2.3 Program Representation	19
2.3.1 Runtime Representation	20
2.3.2 Storage Representation	30
2.4 Transformation Language	32
2.4.1 Organisation	32
2.4.2 Transformation Atoms	36
2.4.3 Typing	47
2.5 Discussion	49
2.6 Summary	50
3 Strategic Term Rewriting	51
3.1 Term Rewriting	51
3.1.1 Algebraic Signatures and Language Signatures	51
3.1.2 Patterns and Terms	53

3.1.3	Rewrite Rules	55
3.1.4	Rewriting Strategies	55
3.2	System S – Strategic Term Rewriting.	56
3.2.1	Primitive Operators and Strategy Combinators	58
3.2.2	Primitive Traversal Strategies	59
3.2.3	Building and Matching Terms	61
3.2.4	Variable Scoping	61
3.2.5	Rewrite Rules	62
3.2.6	Additional Constructs	62
3.3	Stratego	63
3.3.1	Signatures, Patterns and Terms	63
3.3.2	Congruences	65
3.3.3	Scoped, Dynamic Rules	65
3.3.4	Overlays	67
3.3.5	Modules	67
3.3.6	Stratego/XT	67
3.4	Summary	68
 III Abstractions for Language Independence		69
4	Program Object Model Adapters	71
4.1	Introduction	71
4.2	The Program Object Model Adapter	73
4.2.1	Architecture Overview	73
4.2.2	Design Overview	74
4.3	Implementation	81
4.3.1	Term Interface	81
4.3.2	Design Considerations	86
4.4	Related Work	88
4.5	Discussion	89
4.6	Summary	90
5	Modularising Cross-Cutting Transformation Concerns	91
5.1	Introduction	91
5.2	Constant Propagation	92
5.3	AspectStratego	95
5.3.1	Joinpoints	96
5.3.2	Pointcuts	96
5.3.3	Advice	98
5.3.4	Cloning	99

5.3.5	Weaving	99
5.3.6	Modularisation	100
5.4	Case Studies	100
5.4.1	Logging	101
5.4.2	Type Checking	102
5.4.3	Extending Algorithms	103
5.5	Implementation of the Weaver	109
5.5.1	A Weaving Example	111
5.5.2	Aspects as Meta Programs	112
5.6	Discussion	112
5.7	Summary	114
 IV Supportive Abstractions for Transformations		115
6	An Extensible Transformation Language	117
6.1	An Extensible Compiler	117
6.1.1	Declaring Syntax and Assimilator	117
6.1.2	Language Extensions	119
6.1.3	Compiler Pipeline	119
6.1.4	StrategoCore	121
6.2	An Extensible Runtime	122
6.2.1	Design	122
6.2.2	Implementation	123
6.2.3	Performance	123
6.3	Light-Weight and Portable Transformation Components	124
6.3.1	Transformlets	124
6.3.2	Implementation	125
6.4	Summary	125
7	Strategic Graph Rewriting	129
7.1	Abstract	129
7.2	Introduction	130
7.3	Extending Term Rewriting Strategies to Term Graphs	131
7.3.1	Term Rewriting Strategies	131
7.3.2	References	133
7.3.3	Rewrite Rules and References	134
7.3.4	Term Graph Traversal	135
7.4	From Terms to Term Graphs	137
7.4.1	Use-Def Chains	137
7.4.2	Call Graphs	139

7.4.3	Flow Graphs	140
7.5	Graph Algorithms and Applications	141
7.5.1	Finding Mutually Recursive Functions	142
7.5.2	Lazy Graph Loading	142
7.6	Implementation	143
7.7	Related Work	144
7.8	Discussion and Further Work	145
7.9	Conclusion	145
V	Case Studies	147
8	Language Extensions as Transformation Libraries	149
8.1	Abstract	149
8.2	Introduction	150
8.3	The Alert DSAL	151
8.3.1	The TIL Language	152
8.3.2	Alert Declarations and Handlers	152
8.4	Implementation of TIL+Alert	155
8.4.1	DSAL = library + notation	155
8.4.2	Type Checking	156
8.4.3	Alert Weaving	157
8.4.4	Coordination	160
8.5	Discussion	160
8.5.1	Program Transformation	160
8.5.2	Program Transformation Languages for Aspect Implemen- tation	162
8.5.3	Related Work	164
8.6	Conclusion	165
8.7	TIL Grammar	166
9	Interactive Transformation and Editing Environments	167
9.1	Introduction	167
9.2	Core Functionality	169
9.2.1	Architecture	169
9.2.2	Build Weave	170
9.2.3	Project Rebuilding	170
9.3	Editor	171
9.3.1	Content Completion	171
9.3.2	Syntax Highlighting	171
9.3.3	Parenthesis Highlighter	172

9.3.4	Outline	172
9.3.5	Source Code Navigation	173
9.3.6	Build Console	173
9.4	Scripting	174
9.4.1	Script View	174
9.4.2	Script Console	174
9.4.3	Analysing and Transforming Source	176
9.4.4	Transformation Hooks	176
9.5	Implementation	176
9.6	Related Work	177
9.7	Summary	178
10	Extending Compilers with Transformation and Analysis Scripts	179
10.1	Introduction	179
10.2	Scriptable Domain-Specific Analysis and Transformation	181
10.2.1	Architecture	182
10.2.2	MetaStratego as a Scripting Engine	183
10.3	Examples of Domain Support Scripting	184
10.3.1	Project-Specific Code Style Checking	184
10.3.2	Custom Data-Flow Analysis	188
10.3.3	Domain-specific Source Code Transformations	190
10.4	Implementation	191
10.4.1	Analysis Architecture	191
10.4.2	Transformlet Repositories	192
10.5	Related work	192
10.6	Discussion	193
10.7	Summary	194
11	Code Generation for Axiom-based Unit Testing	195
11.1	Introduction	196
11.2	Expression Axioms in Java	197
11.2.1	JUnit Assertions	197
11.2.2	Java Specification Logics	198
11.3	Structuring the Specifications	201
11.3.1	Associating Axioms with Types	202
11.3.2	Optional and Inherited-only Axioms	204
11.4	Java API caveats	206
11.4.1	Override and Overload	206
11.4.2	clone and Other Protected Methods	207
11.4.3	The equals “congruence” relation	208
11.5	Testing	208

11.5.1	Test Data Generator Methods	209
11.5.2	Determining Test Set Quality	210
11.5.3	Running the Tests	210
11.5.4	Interpreting Test Results	211
11.6	Test Suite Generation	211
11.6.1	Generating Tests from Axioms	212
11.6.2	Organising Generated Tests	215
11.6.3	Executing Tests	215
11.7	Implementation	216
11.8	Discussion and Related Work	217
11.9	Summary	218
VI	Conclusion	221
12	Discussion	223
12.1	Techniques for Language Independence	223
12.1.1	Abstracting over Data	223
12.1.2	Expressing Generic Algorithms	224
12.1.3	Adapting Generic Algorithms	225
12.1.4	Modular Language Descriptions	225
12.2	Other Approaches to Software Evolution	226
12.3	Availability of Research Systems	226
13	Further Work	227
14	Conclusions	229
15	Summary	231

Acknowledgements

Most dissertation prefaces start with a sentence saying that the work would not have been possible if it had not been for the supervisor(s). That's the third sentence in this preface. If it had not been for Magne Haverlaen and Eelco Visser, this dissertation would not exist. I am extremely grateful to both for allowing me to play around with what I consider to be really fun stuff for the last three and half years – and getting paid for it! The different styles of Eelco and Magne have been a source of inspiration, and sometimes a little bit of frustration. In some odd way, it's like having two parents who don't always agree on certain parts of your upbringing. Magne's Socratic style of teaching and counseling together with his insistence on thorough (algebraic!) analysis before anything else has taught me to think a lot harder about the assumptions I would normally make without noticing. Eelco's style of exploration and research through construction has in many ways reinforced my natural tendency to build things in order to understand how they work. The two styles can certainly be made to mix, as Eelco and Magne have both done and shown themselves. I've come to notice, however, that finding the best mix requires a good amount of patience and experience.

Magne must bear most of the blame for luring me into computer science research in the first place. His sneaky trick was: "If you're not planning on going into research, there's no need for me to spend any time commenting on your master thesis, since you won't be needing good marks on it anyway." It worked. (I was young and naive.) I agreed to applying for a PhD scholarship, and the journey since has been most enjoyable. This journey took me to Eelco's lab at Utrecht University, The Netherlands for the academic year 2004/2005. Escaping the rain in Bergen for ten months was great. It was better still to work with Eelco and his band of merry hackers. Most of the Stratego-specific work, such as GraphStratego, AspectStratego and MetaStratego, has its origins from my time in Utrecht. Martin Bravenboer, Eelco Dolstra, Rob Vermaas, Rui Guerra, Karina Olmos, Armijn Hemel, Iris Reinbacher, Andres Löh, Peter Verbaan, Peter Lennartz and the rest of the gang in Utrecht made my stay there truly memorable.

My journey did not end in Utrecht, however. I've also had the pleasure of visiting the lab of Krzysztof Czarnecki at the University of Waterloo, Canada. I greatly appreciate the insights provided by Krzysztof. While there, I conducted most of the work that went into the software transformation systems survey of this dissertation. Again, I was extremely fortunate to spend time with some great lab mates: Barry Pekilis, Sean Lau, Michal Antkiewicz, Krzysztof Pietroszek, Chang Hwan Peter Kim, Abbas Heydarnoori, Igor Ivkovic and other passers by.

But wait. There's more. I spent last summer at IBM's T.J. Watson Research Center in New York, USA. While there, I worked with Norman Cohen, Paul Chou and Vijay Saraswat. Though not directly relevant to my thesis project, I got to hack

on compilers for three months straight, and was happy as a clam. I met heaploads of nice people there as well, including my lab mates Marco Zimmerling and Young-Ri Choi, and fellow interns Ilona Gaweda and Shane Markstrum. I also got very useful feedback on my research from Bob Fuhrer, Frank Tip and John Field. My time at Watson enforced my motivation for plugging transformation systems into existing language infrastructures (of which they have a lot).

During my occasional stays Bergen, I've been surrounded by rain and supportive friends. My brother Arne Kristian and Eva Kamilla have always had a warm meal and a mattress ready. Tilde Broch Østborg, the person apart from my family who knows me the best, has been a constant support. Lately, Tormod Haugen has pleasantly invaded my privacy and offered his dry wit and juicier cooking. At the office, Anya Bagge has always had a solution to my LaTeX problems and there is nothing worth knowing about building Stratego programs that Valentin David hasn't mastered. Paul Simon Svanberg has been my link to Real Life in the last few months and Eva Succi has made me believe that I know a little bit about LaTeX, too.

Throughout the entire process, whenever something went awry, or whenever something went alright, my mother has been but a phone call away. And in the way only mothers can, she's put things in perspective when I've been suffering from the compulsory fits of PhD despair.

During the writing of this dissertation I have received thorough input from Eelco and Magne, but I've also had great help from some very good friends. In alphabetical order (because any rating would be inaccurate and unfair): Martin Bravenboer, Tormod Haugen, Barry Pekilis.