

Contents

1	Introduction	1
1.1	The Problem	1
1.2	Our Solution	2
1.3	Overview of this Thesis	2
1.4	Relation to our Previous Work	3
2	Preliminaries	5
2.1	Use of Formal Specifications	5
2.1.1	Why Generate Compilers?	6
2.2	Ways to Specify Semantics	6
2.2.1	Interpreters	6
2.2.2	Abstract Machines	7
2.2.3	Attribute Grammars	7
2.2.4	Denotational Semantics	8
2.2.5	Action Semantics	9
2.2.6	Evolving Algebras	10
2.2.7	Structural Operational Semantics	10
2.3	Natural Semantics	11
2.3.1	Natural Deduction	11
2.3.2	Relation to Programming Languages	12
2.3.3	Example	12
2.3.4	Meaning	13
2.3.5	Pragmatics	14
2.3.6	Recent Extensions	15
3	The Design of RML	17
3.1	Syntax	18
3.2	Static Semantics	20
3.2.1	Bindings and Unknowns	21
3.2.2	Technicalities	21
3.3	Modelling Backtracking	22
3.3.1	Intuition	22
3.3.2	Origins	24
3.3.3	Denotational Semantics of Backtracking	25

3.4	Determinacy	30
3.5	History	31
3.5.1	Static Semantics	31
3.5.2	Dynamic Semantics	32
3.6	Differences from SML	32
4	Examples	35
4.1	A Small Example	35
4.1.1	Abstract Syntax	35
4.1.2	Inference Rules	36
4.1.3	Operational Interpretation	36
4.2	Mini-Freja	37
4.2.1	Abstract Syntax	37
4.2.2	Values	38
4.2.3	Environments	38
4.2.4	Evaluation	39
4.2.5	Modularity	41
4.2.6	Adding Recursion	42
4.2.7	Summary	44
4.3	Diesel	44
4.3.1	Static Elaboration	44
4.3.2	Flattening	45
4.3.3	Emitting Code	46
4.3.4	C Glue	46
4.3.5	Summary	46
4.4	Petrol	47
4.4.1	Summary	47
4.5	Mini-ML	47
4.5.1	Rémy-Style <code>let</code> -Polymorphism	48
4.5.2	Equality Types	50
4.5.3	Wright's Simple Imperative Polymorphism	50
4.5.4	Overloading	51
4.5.5	Specification Fragments	53
4.5.6	Summary	55
4.6	Problematic Issues	55
4.6.1	Environments	55
4.6.2	Default Rules	55
4.7	Summary	56
5	Implementation Overview	57
5.1	Compilation Strategy	57
5.1.1	Development	58
5.2	Alternatives	59
5.2.1	Prolog	59

5.2.2	Warren's Abstract Machine	59
5.2.3	Attribute Grammars	60
5.2.4	SML	60
5.3	Implementation Status	60
6	Reducing Nondeterminism	63
6.1	Background	64
6.1.1	Grammars	64
6.2	FOL Representation	65
6.3	The Front-End	66
6.4	The FOL-TRS Rewriting System	67
6.5	Properties	70
6.5.1	Termination	70
6.5.2	Confluence	73
6.5.3	Alternatives for Rewriting Negations	74
6.6	Examples	75
6.6.1	<code>append</code>	75
6.6.2	<code>lookup</code>	76
6.7	Missed Conditionals	78
6.8	Implementation Notes	81
6.8.1	Implementation Complexity	82
6.9	Limitations	83
6.10	Related Work	84
7	Compiling Pattern Matching	85
7.1	Introduction	85
7.1.1	What is Matching?	85
7.1.2	Compiling Term Matching	86
7.2	Troublesome Examples	87
7.2.1	Copied Expressions	88
7.2.2	Repeated and Sub-Optimal Tests	89
7.3	Intuitive Operation	89
7.4	Preliminaries	91
7.4.1	Objects	92
7.4.2	Operations	93
7.5	The Algorithm	95
7.5.1	Step 1: Preprocessing	95
7.5.2	Step 2: Generating the DFA	96
7.5.3	Step 3: Merging of Equivalent States	97
7.5.4	Step 4: Generating Intermediate Code	97
7.6	The Examples Revisited	98
7.6.1	The <code>demo</code> Function	98
7.6.2	The <code>unwieldy</code> Function	100
7.6.3	State Merging	102
7.7	Implementation Notes	105

7.7.1	Data Representation	105
7.7.2	Compile-Time Warnings	105
7.7.3	Matching Exceptions	106
7.7.4	Guarded Patterns	106
7.8	Related Work	107
7.9	Modifications for RML	108
7.10	Experiences and Conclusions	109
8	Compiling Continuations	111
8.1	Properties of CPS	111
8.2	Translating RML to CPS	113
8.2.1	Data Representation	113
8.2.2	Local Optimizations on CPS	116
8.3	Translating CPS to Code	116
8.3.1	Control	116
8.3.2	Copy Propagation	120
8.3.3	Memory Allocation	120
8.3.4	Data	121
8.4	Translating Code to C	121
8.4.1	Data Representation	122
8.4.2	Memory Management	122
8.5	A Code Generation Example	123
9	Simulating Tailcalls in C	127
9.1	The Problem	127
9.1.1	Overview	128
9.2	Why is C not Tail-Recursive?	129
9.2.1	Why do not Prototypes Help?	130
9.2.2	ANDF	130
9.3	Preliminaries	130
9.3.1	Tailcall Classification	133
9.4	Plain Dispatching Labels	133
9.4.1	Alternative Access Methods for Globals	135
9.5	The Monster Switch	136
9.6	Dispatching Switches	138
9.6.1	Step 1: Fast Known Intramodule Calls	138
9.6.2	Step 2: Recognizing Unknown Intramodule Calls	139
9.6.3	Step 3: Fast Unknown Intramodule Calls	140
9.6.4	Additional Benefits	144
9.7	Pushy Labels	144
9.7.1	Pushy Labels and Register Windows	147
9.8	The ‘Warped Gotos’ Technique	148
9.9	The <code>wamcc</code> Approach	150
9.10	Non-Solutions	151
9.11	Experimental Results	152

9.12	Conclusions	152
10	Performance Evaluation	153
10.1	Target Systems	153
10.2	Overview	154
10.3	Allocation Arena Size	155
10.4	State Access Methods	163
10.5	Compiler Optimizations	163
10.6	Facing the Opposition	166
10.6.1	Mini-Freja	166
10.6.2	Petrol	167
10.7	Conclusions	168
11	Concluding Remarks	169
11.1	Summary	169
11.2	Future Work	170
11.2.1	Default Rules	170
11.2.2	Programming Sub-Language	171
11.2.3	Taming Side-Effects	171
11.2.4	Moded Types	171
11.2.5	Linear Types	171
11.2.6	Compile to SML	172
11.2.7	Tuning the Runtime Systems	172
11.2.8	User-Friendliness	172
A	The Definition of RML	173
A.1	Introduction	173
A.1.1	Differences to SML	173
A.2	Notation for Natural Semantics	175
A.2.1	Lexical Definitions	175
A.2.2	Syntax Definitions	175
A.2.3	Sets	176
A.2.4	Tuples	176
A.2.5	Finite Sequences	176
A.2.6	Finite Maps	176
A.2.7	Substitutions	177
A.2.8	Disjoint Unions	177
A.2.9	Relations	177
A.2.10	Example	178
A.3	Lexical Structure	180
A.3.1	Reserved Words	180
A.3.2	Integer Constants	180
A.3.3	Real Constants	180
A.3.4	Character Constants	180
A.3.5	String Constants	180

A.3.6	Identifiers	181
A.3.7	Type Variables	181
A.3.8	Whitespace and Comments	181
A.3.9	Lexical Analysis	181
A.4	Syntactic Structure	183
A.4.1	Derived Forms, Full and Core Grammar	183
A.4.2	Ambiguity	183
A.5	Static Semantics	190
A.5.1	Simple Objects	190
A.5.2	Compound Objects	190
A.5.3	Initial Static Environments	191
A.5.4	Inference Rules	191
A.6	Dynamic Semantics	201
A.6.1	Simple Objects	201
A.6.2	Compound Objects	201
A.6.3	Initial Dynamic Objects	202
A.6.4	Inference Rules	202
A.7	Initial Objects	211
A.7.1	Initial Static Objects	211
A.7.2	Initial Dynamic Objects	211

Bibliography	223
---------------------	------------

Index	239
--------------	------------

List of Figures

3.1	Denotational semantics of DNF, part 1	26
3.2	Denotational semantics of DNF, part 2	27
3.3	Denotational semantics of DNF, part 3	27
4.1	Textbook style syntax for Mini-Freja	37
4.2	Rémy-style <code>let</code> -polymorphism	49
6.1	Syntax of FOL	65
6.2	Syntax of FOL patterns	81
7.1	Syntax of auxiliary objects	92
7.2	Automaton with a shared state	104
8.1	CPS calculus	112
8.2	Summary of CPS representation	114
8.3	Simplifying primitive operations	117
8.4	Simplifying trivial expressions	117
8.5	Simplifying CPS expressions	118
8.6	Summary of Code representation	119
8.7	Code generation example: RML source	123
8.8	Code generation example: intermediate CPS code	124
8.9	Code generation example: C code for test	125
8.10	Code generation example: C code for <code>sc114</code>	126
9.1	Prototypical intermediate language	131
10.1	<code>alpha-cc</code>	156
10.2	<code>alpha-gcc</code>	156
10.3	<code>hp-cc</code> , only <code>mf</code> example	157
10.4	<code>hp-gcc</code>	157
10.5	<code>i386-cc</code>	158
10.6	<code>i386-gcc</code> , only <code>pushy</code> and <code>warped</code> runtime systems	158
10.7	<code>mips-cc</code>	159
10.8	<code>mips-gcc</code>	159

10.9	ppc-cc	160
10.10	ppc-gcc	160
10.11	sparc-cc	161
10.12	sparc-gcc	161
A.1	Full grammar: auxiliaries	183
A.2	Full grammar: types	184
A.3	Full grammar: patterns	184
A.4	Full grammar: expressions	184
A.5	Full grammar: goals and clauses	185
A.6	Full grammar: declarations	185
A.7	Derived forms of type variable sequences, types, and type sequences	186
A.8	Derived forms of patterns and pattern sequences	186
A.9	Derived forms of expressions and expression sequences	187
A.10	Derived forms of goals and clauses	187
A.11	Derived forms of specifications and declarations	187
A.12	Core grammar: types	188
A.13	Core grammar: patterns	188
A.14	Core grammar: expressions	188
A.15	Core grammar: goals and clauses	188
A.16	Core grammar: declarations	189
A.17	Auxiliary grammar: programs	189
A.18	Compound semantic objects	190
A.19	Simple semantic objects	201
A.20	Compound semantic objects	201
A.21	Grammar of continuation terms	201
A.22	Interface of the standard <code>rm1</code> module	212
A.23	Interface of the standard <code>rm1</code> module (contd.)	213
A.24	Interface of the standard <code>rm1</code> module (contd.)	214
A.25	Derived types and relations	218
A.26	Derived types and relations (contd.)	219
A.27	Derived types and relations (contd.)	220
A.28	Derived types and relations (contd.)	221

List of Tables

6.1	Termination functions for FOL-TRS	71
10.1	Time as a function of tailcall strategy	162
10.2	Timings for different state access methods	164
10.3	Lines of generated C code for ‘plain’	164
10.4	Max stack usage in words	165
10.5	Number of tailcalls	165
10.6	Accumulated execution times	165
10.7	MF, RML vs. Typol, time in seconds	167
10.8	MF, RML vs. Prolog, time in seconds	167
10.9	Petrol, RML vs. Pascal, time in seconds	168



<http://www.springer.com/978-3-540-65968-6>

Compiling Natural Semantics

Pettersson, M. (Ed.)

1999, XVIII, 246 p., Softcover

ISBN: 978-3-540-65968-6