Contents

1	\mathbf{Intr}	oductio	n									1
	1.1	The Pro	${ m oblem}$									1
	1.2 Our Solution											2
	1.3	Overview of this Thesis										2
	1.4	Relation	n to our Previous Work			•	•	•	•	•	i	3
2	Prel	iminari	es									5
	2.1	Use of I	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								. 1	0
		2.2.7	Structural Operational Semantics								. 1	0
	2.3	Natural	Semantics								. 1	1
		2.3.1	Natural Deduction								. 1	1
		2.3.2	Relation to Programming Languages								. 1	2
		2.3.3	Example					Ì			. 1	$\overline{2}$
		2.3.4	Meaning		į		÷				. 1	3
		235	Pragmatics							•	1	4
		2.3.6	Recent Extensions								. 1	5
3	The	Design	of BML								1	7
Ŭ	31	Syntax									1	8
	3.2	Static S	emantics		·	•	·		·	•	. 1	0
	0.2	321	Bindings and Unknowns		•	•	•	•	•	•	. 2	1
		200	Tochnicalities	•	•	•	·	•	•	•	. ຼ າ	1 1
	22	J.2.2 Modolli	ng Backtracking		•	•	•	•	•	•		т Э
	J.J	2 2 1	ng Dackoracking	•	•	•	•	•	•	•	. ⊿ ົ	2 9
		229	Ω_{riging}	•	•	•	•	•	•	•	. ⊿ ົ	2 1
		ປ.ປ.⊿ ຊູຊູຊູ	Depotational Somentics of Restaurashing	•	•	•	•	•	•	•	. 4 ຈ	4 5
		ე.ე.ე	Denotational Semantics of Dacktracking	•	•	•	•	•	•	•	. 2	J

	3.4	Determinacy 30
	3.5	History
		3.5.1 Static Semantics
		3.5.2 Dynamic Semantics
	3.6	Differences from SML 32
	Б	
4	Exa	Mples 35
	4.1	A Sman Example
		4.1.1 ADSTRUCT Syntax \dots
		4.1.2 Interence rules \dots
	4.9	4.1.5 Operational Interpretation
	4.2	$Mini-Freja \dots \dots$
		4.2.1 ADStract Syntax
		4.2.2 Values
		4.2.3 Environments
		$4.2.4 \text{Evaluation} \dots \dots \dots \dots \dots \dots \dots \dots \dots $
		$4.2.5 \text{Modularity} \dots \dots \dots \dots \dots \dots \dots \dots \dots $
		4.2.6 Adding Recursion
		4.2.7 Summary
	4.3	Diesel
		4.3.1 Static Elaboration
		4.3.2 Flattening
		4.3.3 Emitting Code
		4.3.4 C Glue
		4.3.5 Summary
	4.4	Petrol
		4.4.1 Summary 47
	4.5	Mini-ML
		4.5.1 Rémy-Style let-Polymorphism
		4.5.2 Equality Types
		4.5.3 Wright's Simple Imperative Polymorphism
		4.5.4 Overloading
		4.5.5 Specification Fragments 53
		456 Summary 55
	46	Problematic Issues 55
	1.0	4.6.1 Environments 55
		4.6.2 Default Bulos
	47	Summery 56
	4.7	Summary
5	Imp	lementation Overview 57
	5.1	Compilation Strategy 57
		5.1.1 Development $\ldots \ldots 58$
	5.2	Alternatives
		5.2.1 Prolog

		5.2.2	Warren's Abstract Machine									59
		5.2.3	Attribute Grammars									60
		5.2.4	SML									60
	5.3	Implen	nentation Status	•	·	·	·	•		•	•	60
6	\mathbf{Red}	ucing 1	Nondeterminism									63
	6.1	Backgr	ound									64
		6.1.1	Grammars									64
	6.2	FOL R	epresentation									65
	6.3	The Fr	ont-End									66
	6.4	The F(DL-TRS Rewriting System									67
	6.5	Proper	ties									70
		6.5.1	Termination									70
		6.5.2	Confluence									73
		6.5.3	Alternatives for Rewriting Negations .									74
	6.6	Examp	les									75
		6.6.1	append									75
		6.6.2	lookup									76
	6.7	Missed	Conditionals									78
	6.8	Implen	nentation Notes									81
		6.8.1	Implementation Complexity									82
	6.9	Limita	tions									83
	6.10	Related	1 Work									84
7	Con	nilina	Pattern Matching									85
•	71	Introdu	iction									85
	1.1	711	What is Matching?	•	•	•	•	•	• •	•		85
		712	Compiling Term Matching	•	•	•	•	•	• •	•	•	86
	7.2	Troubl	esome Examples	•	•	•	•	•	• •	•		87
	1.4	791	Conjed Expressions	•	•	•	•	•	• •	•	•	88
		799	Beneated and Sub-Ontimal Tests	•	•	•	•	•	• •	•	•	80
	73	Intuitiv	ve Operation	•	•	•	•	•	• •	•	•	80
	7.0	Prolim		•	•	•	•	•	• •	•	•	01
	1.1	7 4 1	Objects	•	•	•	•	•	• •	•	•	02
		7.4.9	Operations	•	•	•	•	•	• •	•	•	03
	75	The Δ	gorithm	•	•	•	•	•	• •	•	•	95 05
	1.0	751	Step 1: Preprocessing	•	•	•	•	•	• •	•	•	95 05
		759	Step 2: Concreting the DEA	•	•	•	•	•	• •	•	•	06
		753	Step 2: Generating the DFA	•	•	•	•	•	• •	•	•	90 07
		7.5.3	Step 5. Merging of Equivalent States.	•	•	•	•	•	• •	•	•	97
	76	7.0.4 Tho F a	step 4. Generating intermediate Code	•	•	•	·	•	• •	•	•	91 08
	1.0	761	The demo Function	•	•	•	•	•	• •	•	•	90 08
		7.0.1 7.6.9	The upprieldy Function	•	•	•	·	•	• •	•	•	90 100
		7.0.4 7.6.3	State Morging	•	•	•	•	•	• •	•	•	100
	77	1.0.3 Imples	state Merging	•	•	•	•	•	• •	•	•	10Z
	1.1	impien	Tentation Notes	•	•	•	•	•	• •	•	•	109

		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	Con	apiling 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	Sim	ulating Tailcalls in C	127
9	Sim 9.1	ulating Tailcalls in C The Problem	127 127
9	Sim 9.1	ulating Tailcalls in C The Problem 9.1.1 Overview	127 127 128
9	Sim 9.1 9.2	ulating Tailcalls in C The Problem 9.1.1 Overview Why is C not Tail-Recursive?	127 127 128 129
9	Sim 9.1 9.2	ulating Tailcalls in C The Problem 9.1.1 Overview Why is C not Tail-Recursive? 9.2.1 Why do not Prototypes Help?	127 127 128 129 130
9	Sim 9.1 9.2	ulating Tailcalls in C The Problem 9.1.1 Overview Why is C not Tail-Recursive? 9.2.1 Why do not Prototypes Help? 9.2.2 ANDF	127 127 128 129 130 130
9	Sim 9.1 9.2 9.3	ulating Tailcalls in C The Problem 9.1.1 Overview Why is C not Tail-Recursive? 9.2.1 Why do not Prototypes Help? 9.2.2 ANDF Preliminaries	127 127 128 129 130 130 130
9	Sim [*] 9.1 9.2 9.3	ulating Tailcalls in C The Problem 9.1.1 Overview Why is C not Tail-Recursive? 9.2.1 Why do not Prototypes Help? 9.2.2 ANDF Preliminaries 9.3.1 Tailcall Classification	127 127 128 129 130 130 130 133
9	Sim 9.1 9.2 9.3 9.4	ulating Tailcalls in C The Problem 9.1.1 Overview Why is C not Tail-Recursive? 9.2.1 Why do not Prototypes Help? 9.2.2 ANDF Preliminaries 9.3.1 Tailcall Classification Plain Dispatching Labels	127 127 128 129 130 130 130 133 133
9	Sim 9.1 9.2 9.3 9.4	ulating Tailcalls in C The Problem 9.1.1 Overview Why is C not Tail-Recursive? 9.2.1 Why do not Prototypes Help? 9.2.2 ANDF Preliminaries 9.3.1 Tailcall Classification Plain Dispatching Labels 9.4.1 Alternative Access Methods for Globals	127 127 128 129 130 130 130 133 133 133
9	Sim: 9.1 9.2 9.3 9.4 9.5	ulating Tailcalls in C The Problem 9.1.1 Overview Why is C not Tail-Recursive? 9.2.1 Why do not Prototypes Help? 9.2.2 ANDF Preliminaries 9.3.1 Tailcall Classification Plain Dispatching Labels 9.4.1 Alternative Access Methods for Globals The Monster Switch	127 128 129 130 130 130 133 133 135 136
9	Sim: 9.1 9.2 9.3 9.4 9.5 9.6	ulating Tailcalls in C The Problem 9.1.1 Overview Why is C not Tail-Recursive? 9.2.1 Why do not Prototypes Help? 9.2.2 ANDF 9.2.2 ANDF Preliminaries 9.3.1 Tailcall Classification Plain Dispatching Labels 9.4.1 Alternative Access Methods for Globals The Monster Switch Dispatching Switches	127 128 129 130 130 130 133 133 135 136 138
9	Sim: 9.1 9.2 9.3 9.4 9.5 9.6	ulating Tailcalls in C The Problem 9.1.1 Overview Why is C not Tail-Recursive? 9.2.1 Why do not Prototypes Help? 9.2.2 ANDF 9.2.2 ANDF Preliminaries 9.3.1 Tailcall Classification Plain Dispatching Labels 9.4.1 Alternative Access Methods for Globals The Monster Switch Dispatching Switches 9.6.1 Step 1: Fast Known Intramodule Calls	127 128 129 130 130 130 133 133 135 136 138 138
9	 Sim 9.1 9.2 9.3 9.4 9.5 9.6 	ulating Tailcalls in C The Problem 9.1.1 Overview Why is C not Tail-Recursive? 9.2.1 Why do not Prototypes Help? 9.2.2 ANDF 9.2.2 ANDF Preliminaries 9.3.1 Tailcall Classification Plain Dispatching Labels 9.4.1 Alternative Access Methods for Globals The Monster Switch 9.6.1 Step 1: Fast Known Intramodule Calls 9.6.2 Step 2: Recognizing Unknown Intramodule Calls	127 128 129 130 130 130 133 133 135 136 138 138 138
9	 Sim 9.1 9.2 9.3 9.4 9.5 9.6 	ulating Tailcalls in C The Problem 9.1.1 Overview Why is C not Tail-Recursive? 9.2.1 Why do not Prototypes Help? 9.2.2 ANDF 9.2.2 ANDF Preliminaries 9.3.1 Tailcall Classification Plain Dispatching Labels 9.4.1 Alternative Access Methods for Globals The Monster Switch Dispatching Switches 9.6.1 Step 1: Fast Known Intramodule Calls 9.6.2 Step 2: Recognizing Unknown Intramodule Calls 9.6.3 Step 3: Fast Unknown Intramodule Calls	127 127 128 129 130 130 133 133 135 136 138 138 138 139 140
9	Sim: 9.1 9.2 9.3 9.4 9.5 9.6	ulating Tailcalls in C The Problem 9.1.1 Overview Why is C not Tail-Recursive? 9.2.1 Why do not Prototypes Help? 9.2.2 ANDF Preliminaries 9.3.1 Tailcall Classification Plain Dispatching Labels 9.4.1 Alternative Access Methods for Globals The Monster Switch Dispatching Switches 9.6.1 Step 1: Fast Known Intramodule Calls 9.6.3 Step 3: Fast Unknown Intramodule Calls 9.6.4 Additional Benefits	127 127 128 129 130 130 133 133 135 136 138 138 138 139 140 144
9	Sim: 9.1 9.2 9.3 9.4 9.5 9.6 9.7	ulating Tailcalls in C The Problem 9.1.1 Overview Why is C not Tail-Recursive? 9.2.1 Why do not Prototypes Help? 9.2.2 ANDF 9.2.2 ANDF Preliminaries 9.3.1 Tailcall Classification Plain Dispatching Labels 9.4.1 Alternative Access Methods for Globals The Monster Switch Dispatching Switches 9.6.1 Step 1: Fast Known Intramodule Calls 9.6.2 Step 2: Recognizing Unknown Intramodule Calls 9.6.3 Step 3: Fast Unknown Intramodule Calls 9.6.4 Additional Benefits	127 127 128 129 130 130 133 133 135 136 138 138 138 139 140 144
9	Sim: 9.1 9.2 9.3 9.4 9.5 9.6 9.7	ulating Tailcalls in C The Problem 9.1.1 Overview Why is C not Tail-Recursive? 9.2.1 Why do not Prototypes Help? 9.2.2 ANDF 9.2.2 ANDF Preliminaries 9.3.1 Tailcall Classification Plain Dispatching Labels 9.4.1 Alternative Access Methods for Globals The Monster Switch Dispatching Switches 9.6.1 Step 1: Fast Known Intramodule Calls 9.6.2 Step 2: Recognizing Unknown Intramodule Calls 9.6.3 Step 3: Fast Unknown Intramodule Calls 9.6.4 Additional Benefits Pushy Labels 9.7.1 Pushy Labels and Register Windows	127 127 128 129 130 130 133 133 133 135 136 138 138 139 140 144 144
9	 Sim 9.1 9.2 9.3 9.4 9.5 9.6 9.7 9.8 	ulating Tailcalls in C The Problem 9.1.1 Overview Why is C not Tail-Recursive? 9.2.1 Why do not Prototypes Help? 9.2.2 ANDF 9.2.2 ANDF Preliminaries 9.3.1 Tailcall Classification Plain Dispatching Labels 9.4.1 Alternative Access Methods for Globals The Monster Switch Dispatching Switches 9.6.1 Step 1: Fast Known Intramodule Calls 9.6.2 Step 2: Recognizing Unknown Intramodule Calls 9.6.3 Step 3: Fast Unknown Intramodule Calls 9.6.4 Additional Benefits 9.7.1 Pushy Labels and Register Windows 9.7.1 Pushy Labels and Register Windows	127 127 128 129 130 130 133 133 135 136 138 138 138 138 139 140 144 144 147
9	Sim: 9.1 9.2 9.3 9.4 9.5 9.6 9.7 9.8 9.9	ulating Tailcalls in C The Problem 9.1.1 Overview Why is C not Tail-Recursive? 9.2.1 Why do not Prototypes Help? 9.2.2 ANDF 9.2.2 ANDF Preliminaries 9.3.1 Tailcall Classification Plain Dispatching Labels 9.4.1 Alternative Access Methods for Globals The Monster Switch Dispatching Switches 9.6.1 Step 1: Fast Known Intramodule Calls 9.6.2 Step 2: Recognizing Unknown Intramodule Calls 9.6.3 Step 3: Fast Unknown Intramodule Calls 9.6.4 Additional Benefits Pushy Labels 9.7.1 Pushy Labels and Register Windows The 'Warped Gotos' Technique The wamcc Approach	127 127 128 129 130 130 133 133 135 136 138 138 138 138 140 144 144 147 148 150
9	Sim: 9.1 9.2 9.3 9.4 9.5 9.6 9.7 9.8 9.9 9.10	ulating Tailcalls in C The Problem 9.1.1 Overview Why is C not Tail-Recursive? 9.2.1 Why do not Prototypes Help? 9.2.2 ANDF Preliminaries 9.3.1 Tailcall Classification Plain Dispatching Labels 9.4.1 Alternative Access Methods for Globals The Monster Switch Dispatching Switches 9.6.1 Step 1: Fast Known Intramodule Calls 9.6.2 Step 2: Recognizing Unknown Intramodule Calls 9.6.3 Step 3: Fast Unknown Intramodule Calls 9.6.4 Additional Benefits Pushy Labels 9.7.1 Pushy Labels and Register Windows The 'Warped Gotos' Technique The wamcc Approach	127 127 128 129 130 130 133 133 135 136 138 138 139 140 144 144 147 148 150 151

	9.12	Conclu	usions	152
10	Perf	orman	nce Evaluation	153
	10.1	Target	$\mathbf{Systems}$	153
	10.2	Overvi	iew	154
	10.3	Allocat	tion Arena Size	155
	10.4	State A	Access Methods	163
	10.5	Compi	iler Optimizations	163
	10.6	Facing	the Opposition	166
		10.6.1	Mini-Freja	166
		10.6.2	Petrol	167
	10.7	Conclu	isions	168
11	Con	cludin	g Bemarks	169
	11 1	Summ	arv	169
	11.2	Future	e Work	170
	11.2	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
Δ	The	Defini	ition of RML	173
	A 1	Introdu	uction	173
	11.1	A 1 1	Differences to SML	173
	A 2	Notati	on 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	Belations	177
		A.2.10	Example	178
	A.3	Lexical	l 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.3.7	Type Variables	181
	A.3.8	Whitespace and Comments	181
	A.3.9	Lexical Analysis	181
A.4	Syntac	tic 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	Dynan	nic 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
		· ·	
Bibliog	raphy		223

Index

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 let-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 sc114	126
9.1	Prototypical intermediate language	131
10.1	alpha-cc	156
10.2	alpha-gcc	156
10.3	hp-cc, only mf example	157
10.4	hp-gcc	157
10.5	i386-cc	158
10.6	i386-gcc, only pushy and warped runtime systems	158
10.7	mips-cc	159
10.8	mips-gcc	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 se-	
quences	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 rml module	212
A.23 Interface of the standard rml module (contd.)	213
A.24 Interface of the standard rml 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	
10.1	Time as a function of tailcall strategy 162)
10.2	Timings for different state access methods	ŀ
10.3	Lines of generated C code for 'plain' 164	ŀ
10.4	Max stack usage in words 165)
10.5	Number of tailcalls)
10.6	Accumulated execution times)
10.7	MF, RML vs. Typol, time in seconds	7
10.8	MF, RML vs. Prolog, time in seconds 167	7
10.9	Petrol, RML vs. Pascal, time in seconds	;