

Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

Continuations and Martin-Löf's Type Theory

A thesis presented in partial fulfilment of the requirements of the degree of

Doctor of Philosophy
in
Computer Science

at Massey University, Albany, New Zealand.

Neil Leslie
February 29, 2000

Abstract

We explain how to program with continuations in Martin-Löf's theory of types (M-LTT).

M-LTT is a theory designed to formalize constructive mathematics. By appealing to the Curry-Howard 'propositions as types' analogy, and to the Brouwer-Heyting-Kolmogorov interpretation of intuitionistic logic we can treat M-LTT as a framework for the specification and derivation of correct functional programs. However, programming in M-LTT has two weaknesses:

- we are limited in the functions that we can naturally express;
- the functions that we do write naturally are often inefficient.

Programming with continuations allows us partially to address these problems. The continuation-passing programming style is known to offer a number of advantages to the functional programmer. We can also observe a relationship between continuation passing and type lifting in categorical grammar.

We present computation rules which allow us to use continuations with inductively-defined types, and with types not presented inductively. We justify the new elimination rules using the usual proof-theoretic semantics. We show that the new rules preserve the consistency of the theory.

We show how to use well-orderings to encode continuation-passing operators for inductively defined types.

Acknowledgements

An earlier version of some of the material in Chapter 6 appeared as [70].

I would like to thank:

- Peter Kay of Massey University's Albany campus, and Steve Reeves of Waikato University, for providing invaluable support and guidance;
- Ross Renner, and the School of Mathematical and Computing Sciences at Victoria University of Wellington, for indulging me with time and money to visit Waikato University to talk with Steve;
- Mhairi for caring (again) that I should finish a thesis;
- Keir and Ailidh for not caring about theses at all.

I would also like to acknowledge the debt which I owe to Aaron T. Beck.

Contents

Introduction	1
Part I Martin-Löf's Type Theory	4
1. Syntax, judgement and inference	5
1.1 Rules of syntax: arited expressions	7
1.2 Forms of judgement	13
1.2.1 Judgement of being a type	13
1.2.2 Judgement of being an object of a type	14
1.2.3 Judgement of two types being equal	14
1.2.4 Judgement of two objects being equal	14
1.2.5 Hypothetical judgements	15
1.2.6 Other interpretations of the judgement forms	16
1.3 Syntax of proof rules	17
1.4 Justification of the rules	18
1.4.1 Using 'meaning is use'	19
1.4.2 Adopting a molecular approach	23
1.4.3 The challenge of tonk	23

Contents	v
1.4.4 Clarifying our notion of rules	25
1.4.5 Self-justifying introduction rules	26
1.4.6 Computation rules	27
1.4.7 Elimination rule	28
1.4.8 Summary of the relationship between the rules	30
1.5 Chapter Summary	31
2. Some useful types, and applications	32
2.1 ‘Logical’ types	33
2.1.1 Disjoint union of two types: $+$	33
2.1.2 Disjoint union of a family of types: Σ types	36
2.1.3 Non-dependent Σ	38
2.1.4 Cartesian product of a family of types: Π types	38
2.1.5 Non-dependent Π	40
2.1.6 The two abstractions	40
2.2 Finite sets, or enumerated types	41
2.2.1 Booleans	42
2.2.2 The empty type	43
2.3 Applications	46
2.3.1 Defining $+$ _{df}	46
2.3.2 Representing the logical constants	48
2.3.3 Dealing with donkey sentences	52
2.3.4 Program development	54
2.4 Inductively defined types	57

2.4.1	Natural numbers	58
2.4.2	Polymorphic lists	60
2.4.3	Binary Trees	63
2.5	Well-orderings	64
2.5.1	Using well-orderings to represent binary trees	66
2.6	Equality	68
2.6.1	Intensional equality: ID	68
2.6.2	Extensional equality: EQ	70
2.7	Universes	71
2.7.1	Tarskian Universes	72
2.7.2	Hierarchies of Universes	75
2.7.3	Russellian Universes	76
2.8	Using Martin-Löf's Type Theory in practice	77
2.9	Chapter Summary	77
 Part II Continuations		 80
 3. Tail-recursion, and continuations		 81
3.1	Introducing tail-recursion and continuations	81
3.2	Some simple functions	84
3.2.1	A CPS version of Fibonacci's function	85
3.3	Uses of continuations	88
3.3.1	Continuations and I/O	91
3.4	Chapter Summary	92

4. Type-lifting in categorial grammar	93
4.1 A simple categorial grammar	94
4.1.1 Section summary	98
4.2 Examples of simple derivations	98
4.2.1 Section summary	101
4.3 Type-raising as continuation-passing	101
4.4 Chapter Summary	104
 Part III Adding continuations to Martin-Löf's Type Theory	 109
5. CPS non-canonical constants for non-inductive types	110
5.1 Alternative extensions	111
5.2 CPS non-canonical constant for the disjoint union of two types	113
5.2.1 Presenting whentail as a primitive	113
5.2.2 Presenting whentail _{df} as a defined constant	116
5.2.3 Section summary	119
5.3 CPS non-canonical constant for the disjoint union of a fam- ily of types	119
5.3.1 Presenting splittail as a primitive	120
5.3.2 Presenting splittail _{df} as a defined constant	120
5.3.3 Section summary	122
5.4 Cartesian product of a family of types: Π	122
5.5 Other non-inductive types	122
5.6 An observation about equality	123
5.7 Chapter Summary	123

6. CPS non-canonical constants for inductive types	128
6.1 Natural numbers	129
6.1.1 Computation rules	129
6.1.2 Elimination rule	130
6.1.3 Equality rules	131
6.2 Polymorphic lists	131
6.2.1 Computation rules	131
6.2.2 Elimination rule	132
6.2.3 Equality rules	132
6.3 Binary trees	132
6.3.1 Computation rules	132
6.3.2 Elimination rule	133
6.3.3 Equality rules	133
6.4 Comparison with the standard rules	134
6.4.1 Relating natrec and nattail	134
6.4.2 Comparison with Thompson's tprim	138
6.4.3 Relating listrec and listtail	141
6.4.4 Relating treerec and treetail	144
6.5 Chapter Summary	147
7. Implementing CPS using well-order types	164
7.1 Computation rule	164
7.2 Elimination rule	165
7.3 Chapter Summary	166

8. Examples	168
8.1 Program structure	168
8.2 Proof structure	170
8.3 Example of program derivation	171
8.3.1 A CPS sorting algorithm	172
8.3.2 Finding a minimum	174
8.4 Chapter Summary	174
 Part IV Conclusions	 177
9. Conclusions and future work	178
9.1 Topics for further investigation	178
9.2 Summary	179
9.3 Conclusions	180

List of Rules

1.1	An inference rule schema	17
1.2	<i>Tertium Non Datur</i>	21
1.3	An NK proof of Peirce's Law	21
1.4	Peirce's Law as an inference rule	22
1.5	A proof of $P \vee \neg P$	22
1.6	tonk introduction on the left	23
1.7	tonk introduction on the right	23
1.8	tonk elimination on the left	24
1.9	tonk elimination on the right	24
1.10	A proof of B from A	24
1.11	Normalising Rule 1.10	24
1.12	A De Morgan law as an inference rule	25
1.13	Symmetry of $\&$	26
1.14	<i>Modus Tollendo Tollens</i>	26
1.15	The form of a computation rule	28
1.16	A computation rule	29
1.17	An example of an introduction rule	29
1.18	A proof rule	30

2.1	+ formation	33
2.2	+ introduction on the left	33
2.3	+ introduction on the right	33
2.4	when computation 1	34
2.5	when computation 2	34
2.6	+ elimination	34
2.7	Equality of expressions involving when 1	35
2.8	Equality of expressions involving when 2	36
2.9	Σ formation	36
2.10	Σ introduction	36
2.11	split computation	37
2.12	Σ elimination	37
2.13	Π formation	38
2.14	Π introduction	38
2.15	funsplit computation	38
2.16	Π elimination	39
2.17	Evaluating $\text{apply}(f, a)$	39
2.18	apply computation	39
2.19	Alternative Π elimination	40
2.20	$\lambda(\text{succ})$ is a function	41
2.21	Enumerated type formation	41
2.22	Enumerated type introduction	41
2.23	Introduction of equal enumerated types	41
2.24	$\text{case}_{\{x_1, \dots, x_n\}}$ computation	42

2.25 Enumerated type elimination	42
2.26 Equality of expressions involving $\text{case}_{\{x_1, \dots, x_n\}}$	42
2.27 A putative empty type introduction rule	43
2.28 $\{\}$ formation	43
2.29 $\text{case}_{\{\}}$ computation	45
2.30 $\{\}$ elimination	45
2.31 Equality of expressions involving $\text{case}_{\{\}}$	45
2.32 Derivation of $+_{df}$ formation	46
2.33 $+_{df}$ formation	47
2.34 $+_{df}$ introduction	47
2.35 $\vee$ introduction on the left	48
2.36 $\vee$ introduction on the right	48
2.37 $\vee$ elimination	49
2.38 $\&$ elimination	50
2.39 Natural number formation	58
2.40 Natural number introduction 1	58
2.41 Natural number introduction 2	59
2.42 natrec computation 1	59
2.43 natrec computation 2	59
2.44 Natural number elimination	60
2.45 Equality of expressions involving natrec 1	60
2.46 Equality of expressions involving natrec 2	60
2.47 List formation	60
2.48 List introduction 1	61

2.49	List introduction 2	61
2.50	listrec computation 1	61
2.51	listrec computation 2	61
2.52	List elimination	62
2.53	Equality of expressions involving listrec 1	62
2.54	Equality of expressions involving listrec 2	62
2.55	Binary tree formation	63
2.56	Binary tree introduction 1	63
2.57	Binary tree introduction 2	63
2.58	treerec computation 1	63
2.59	treerec computation 2	64
2.60	Binary tree elimination	64
2.61	W formation	65
2.62	W introduction	65
2.63	wrec computation	65
2.64	W elimination	66
2.65	Tree' introduction 1	67
2.66	Tree' introduction 2	67
2.67	ID formation	68
2.68	ID introduction	69
2.69	Alternative ID introduction	69
2.70	idpeel computation	69
2.71	ID elimination	69
2.72	EQ formation	70

2.73	EQ introduction	70
2.74	eqpeel computation	70
2.75	EQ elimination 1	71
2.76	EQ elimination 2	71
2.77	TYPE as a type	71
2.78	Tarskian U_0 formation	72
2.79	Tarskian U_0 introduction 1	72
2.80	Tarskian U_0 introduction 2	73
2.81	$\vee$ formation	73
2.82	Unquoting $\text{Nat}^\wedge$	73
2.83	Unquoting $\text{List}^\wedge$	73
2.84	urec computation 1	74
2.85	urec computation 2	74
2.86	Tarskian U_0 elimination	75
2.87	Derivation of $+_{df}$ formation using universes	75
2.88	Russellian U_0 introduction 1	76
2.89	Russellian U_0 introduction 2	76
2.90	Inhabitants of the Russellian U_0 are types	76
2.91	Tree' elimination	79
3.1	A tail call	82
4.1	The $\Rightarrow \backslash$ sequent rule	94
4.2	The $\Rightarrow /$ sequent rule	95
4.3	The $\backslash \Rightarrow$ sequent rule	95

4.4	The $/ \Rightarrow$ sequent rule	95
4.5	The cut rule	95
4.6	$\backslash$ Introduction	96
4.7	$/$ introduction	97
4.8	$\backslash$ elimination	97
4.9	$/$ elimination	97
4.10	Lexicon lookup	99
4.11	Showing ‘John loves Mary’	100
4.12	Type raising sequent rule 1	102
4.13	Type raising sequent rule 2	102
4.14	Type raising natural deduction rule 1	102
4.15	Type raising natural deduction rule 2	102
4.16	Showing ‘He loves Mary’	104
4.17	Lifting ‘John’	104
4.18	One reading of ‘Murray believes Souness resigned foolishly’	106
4.19	The other reading of ‘Murray believes Souness resigned foolishly’	107
4.20	‘He or John loves Mary’	108
5.1	whentail computation 1	113
5.2	whentail computation 2	113
5.3	Derivation of the computation rules for whentail_{df}	117
5.4	Derivation of a computation rule for whentail_{df} 1	117
5.6	whentail_{df} computation 1	117
5.5	Derivation of a computation rule for whentail_{df} 2	118

5.7	whentail _{df} computation 2	118
5.8	splittail computation	120
5.9	Derivation of splittail _{df} computation	121
5.10	splittail _{df} computation	121
5.11	+ elimination using whentail	124
5.12	Typing whentail _{df}	125
5.13	Σ elimination using splittail	126
5.14	Derivation of Σ elimination with splittail _{df}	127
6.1	nattail computation 1	129
6.2	nattail computation 2	129
6.3	Equality of expressions involving nattail 1	131
6.4	listtail computation 1	131
6.5	listtail computation 2	132
6.6	Equality of expressions involving listtail 1	132
6.7	treetail computation 1	133
6.8	treetail equality	133
6.9	Evaluating nattail($d, e, \text{zero}, \lambda(f)$)	135
6.10	Evaluating apply($\lambda(f), \text{natrec}(d, e, \text{zero})$)	135
6.11	Induction step	136
6.12	Evaluating nattail($d, e, \text{succ}(m), \lambda(f)$)	136
6.13	Evaluating apply($\lambda(f), \text{natrec}(d, e, \text{succ}(m))$)	136
6.14	A proof that nattail(d, e, n, g) is equal to apply($g, \text{natrec}(d, e, n)$)	137
6.15	A corollary of Rule 6.37	138
6.16	tprim computation 1	139

6.17	tprim computation 2	139
6.18	tprim computation 3	139
6.19	Evaluating listtail($d, e, \text{nil}, \lambda(f)$)	141
6.20	Evaluating apply($\lambda(f), \text{listrec}(d, e, \text{nil})$)	141
6.21	Induction step	142
6.22	Evaluating listtail($d, e, \text{cons}(a, as), \lambda(f)$)	142
6.23	Evaluating apply($\lambda(f), \text{listrec}(d, e, \text{cons}(a, as))$)	142
6.24	A corollary of Rule 6.43	143
6.25	Evaluating treetail($d, e, \text{leaf}, \lambda(f)$)	144
6.26	Evaluating apply($\lambda(f), \text{treerec}(d, e, \text{leaf})$)	144
6.27	Induction step	145
6.28	Evaluating treetail($d, e, \text{node}(v, l, r), \lambda(f)$)	145
6.29	Evaluating apply($\lambda(f), \text{treerec}(d, e, \text{node}(v, l, r))$)	146
6.30	Natural number elimination with tail recursion	148
6.31	Equality of expressions involving nattail 2	149
6.32	List elimination with tail recursion	150
6.33	Equality of expressions involving listtail 2	151
6.34	treetail computation 2	152
6.35	Binary tree elimination with tail recursion	153
6.36	treetail equality 2	154
6.37	When nattail and natrec are the same	155
6.38	Sub-proof π_1	156
6.39	Sub-proof π_2	157
6.40	Sub-proof $\pi_{2.1}$	158

6.41 Sub-proof $\pi_{2.2}$ 159

6.42 Sub-proof $\pi_{2.2.1}$ 160

6.43 When listtail and listrec are the same 161

6.44 When treetail and treerec are the same 162

6.45 A corollary of Rule 6.44 163

7.1 wtail computation 165

7.2 W elimination using wtail 167

8.1 Typing $\text{length}(l)$ 169

8.2 Typing $\text{lengthtail}(l, k)$ 170

8.3 Typing $\text{ls}(l, m)$ 171

8.4 Sorting a list 173

8.5 Typing $\text{lstail}(l, m, k)$ 176

List of Programs

3.1	The length of a list 1	82
3.2	The length of a list 2	83
3.3	The length of a list 3	83
3.4	The factorial function	84
3.5	A CPS factorial	84
3.6	Fibonacci's function in Haskell	85
3.7	A structurally recursive version of Fibonacci's function . . .	86
3.8	A neater version of Fibonacci's function	86
3.9	A CPS version of Fibonacci's function	86
3.10	Looping	87
3.11	Computing Fibonacci's function in C	88

List of Figures

3.1	Evaluating $\text{cpsfac } 3 \ k$	85
3.2	Evaluating $\text{cpsfibs } 3 \ k$	87
3.3	Plotkin's CPS-conversion	89
3.4	CPS-translation some particular terms	89
4.1	Proof normalisation in categorial grammar 1	98
4.2	Proof normalisation in categorial grammar 2	98
4.3	A lexicon	99
4.4	Extending our lexicon	100
4.5	Further extending our lexicon	103
6.1	Informal evaluation of $\text{tprim}(n, c, f, p, v)$	140
6.2	Informal evaluation of $\text{nattail}(d, e, q, k)$	140