QUALIFIED TYPES: THEORY AND PRACTICE
Distinguished Dissertations in Computer Science

Edited by C.J. van Rijsbergen, University of Glasgow

The Conference of Professors of Computer Science (CPCS), in conjunction with the British Computer Society (BCS), selects annually for publication up to four of the best British PhD dissertations in computer science. The scheme began in 1990. Its aim is to make more visible the significant contribution made by Britain, in particular by students, to computer science, and to provide a model for future students. Dissertations are selected on behalf of CPCS by a panel whose members are:

M. Clint, Queen's University, Belfast (Chairman)
R. Milner, University of Edinburgh
K. Moody, University of Cambridge
M.S. Paterson, University of Warwick
S. Srinivastava, University of Newcastle upon Tyne
A. Sloman, University of Birmingham
F. Sumner, University of Manchester
Qualified Types:
Theory and Practice

Mark. P. Jones

University of Nottingham
Contents

Summary of notation ix
Acknowledgements xii

1 Introduction 1
1.1 Type systems for programming languages 1
1.2 Type inference and polymorphism 2
1.3 The choice between ‘all’ and ‘one’ 3
1.4 Qualified types 4
1.5 Outline of thesis 4

2 Predicates 6
2.1 Basic definitions 6
2.2 Type classes 8
  2.2.1 Class declarations 8
  2.2.2 Instance declarations 9
  2.2.3 Superclasses 10
  2.2.4 The entailment relation for type classes 10
2.3 Subtyping 11
2.4 Extensible records 13

3 Type inference for qualified types 16
3.1 An extension of ML with qualified types 17
  3.1.1 Type expressions 17
  3.1.2 Terms 17
  3.1.3 Typing rules 18
3.2 Understanding type schemes 18
  3.2.1 Constrained type schemes 19
  3.2.2 Examples 20
  3.2.3 Properties of the (≤) ordering 21
  3.2.4 Generalisation 22
### Contents

3.2.5 Ordering of type assignments ........................................... 22

3.3 A syntax-directed approach .................................................. 23

3.3.1 Syntax-directed typing rules ............................................. 23

3.3.2 Properties of the syntax-directed system .............................. 23

3.3.3 Relationship with original type system ............................... 25

3.4 Type inference ................................................................. 25

3.4.1 Unification ................................................................. 25

3.4.2 A type inference algorithm .............................................. 26

3.4.3 Principal type schemes .................................................. 27

3.4.4 Decidability of type inference problems .............................. 29

3.5 Related work ................................................................. 29

4 Evidence ............................................................................. 31

4.1 Simple implementations of overloading ................................... 32

4.2 Overloading made explicit .................................................... 33

4.3 Predicate entailment with evidence ....................................... 35

4.4 Evidence, predicates and implementations .............................. 36

4.5 Type classes and dictionaries .............................................. 37

4.6 Subtyping and coercion ...................................................... 39

4.7 Implementation of extensible records .................................... 40

5 Semantics and coherence ....................................................... 43

5.1 A version of polymorphic λ-calculus with qualified types .......... 44

5.2 Translation from OML to OP ................................................. 46

5.3 The coherence problem ...................................................... 48

5.4 A definition of equality for OP terms .................................... 49

5.4.1 Uniqueness of evidence .................................................. 49

5.4.2 Reduction of OP terms ................................................... 50

5.4.3 Equalities between OP terms ........................................... 51

5.5 Ordering and conversion functions ...................................... 53

5.5.1 Motivation and conversions between type schemes .............. 53

5.5.2 Conversions between constrained type schemes ................. 55

5.5.3 Conversions between type assignments ............................. 57

5.6 Syntax-directed translation ................................................ 58

5.7 Type inference and translation .......................................... 60

5.8 Coherence results ............................................................. 62

5.8.1 Equality of conversions .................................................. 62

5.8.2 Equality of translations ................................................ 63

5.8.3 A weaker notion of ambiguity ........................................ 64
Contents

5.9 Comparison with related work .................................. 65

6 Theory into practice ........................................... 67
   6.1 Evidence parameters considered harmful .................. 67
      6.1.1 Simplification ........................................ 68
      6.1.2 Unnecessary polymorphism .......................... 69
      6.1.3 The monomorphism restriction ....................... 70
      6.1.4 Constant and locally-constant overloading .......... 71
      6.1.5 A template-based implementation ..................... 72
   6.2 Satisfiability ............................................. 73
   6.3 Incorporating the rule of subsumption .................... 76

7 Type classes in Haskell ........................................ 78
   7.1 Context reduction ........................................ 79
   7.2 Implementation of type classes in HTC ................... 82
      7.2.1 Implementation of HTC dictionaries ............... 82
   7.3 The problem of repeated construction .................... 84
   7.4 Repeated construction caused by recursion ............... 85
      7.4.1 Recursion in the definition of overloaded functions 85
      7.4.2 Mutually recursive groups of functions ........... 87
      7.4.3 The need for a form of full-laziness ............... 88
      7.4.4 Recursion in dictionary constructors ............. 91
   7.5 Other opportunities for shared dictionaries .............. 92
      7.5.1 Simple examples ..................................... 92
      7.5.2 Sharing in hierarchies .............................. 94
      7.5.3 Repetition in superclass hierarchies ............... 97
   7.6 Alternative implementations of dictionary construction 99

8 Type classes in Gofer ........................................... 101
   8.1 The basic principles of GTC ................................ 101
   8.2 The Gofer implementation of dictionaries ................. 104
      8.2.1 A notation for records with subtyping ............. 104
      8.2.2 Dictionaries as records ............................ 105
      8.2.3 An optimisation to reduce the number of dictionary parameters 107
   8.3 A concrete implementation ................................ 108
      8.3.1 Representation of dictionaries ....................... 108
      8.3.2 Abstract machine support ........................... 110
      8.3.3 Some comments on performance ...................... 112
Contents

9 Summary and future work 114
  9.1 Towards a categorical semantics .......................... 115
  9.2 Constructor classes ........................................ 116
  9.3 Reasoning in the presence of overloading ............... 118

10 Epilogue 121
  10.1 Constructor classes ...................................... 121
  10.2 Template based implementations ........................ 122
  10.3 Making use of satisfiability .............................. 123
  10.4 ML typing, explicit polymorphism and qualified types ... 124
  10.5 The use of subsumption .................................. 124

Appendix A Proofs 126

References 148

Index 155
Summary of notation

Substitutions

$R, S, T, U, \ldots$ Substitutions (capture avoiding) of types for type variables.
$id$ The identity substitution.
$RS$ Composition of substitutions $R$ and $S$.
$S[\tau_1/t_1, \ldots, \tau_n/t_n]$ Substitution mapping each $t_i$ to $\tau_i$, and any other variable $t$ to $St$.
$[\tau_1/t_1, \ldots, \tau_n/t_n]$ Abbreviation for $id[\tau_1/t_1, \ldots, \tau_n/t_n]$.
$S[\tau_i/t_i]$ Substitution mapping each $t_i$ to $\tau_i$ as $i$ ranges over some (implicit) set of index values, and any other variable $t$ to $St$.
$[\tau_i/t_i]$ Abbreviation for $id[\tau_i/t_i]$.
$R = S$ Equality of substitutions, ignoring 'new' variables, Section 3.4.2, page 26.

Terms

$E, F, \ldots$ Term expressions.
$x, y, \ldots$ Term variables, Section 3.1.2, page 18.
$EF$ Application of $E$ to $F$, Section 3.1.2, page 18.
$\lambda x. E$ $\lambda$-abstraction, Section 3.1.2, page 18.
let $x = E$ in $F$ Local definition of $x$ in $F$, Section 3.1.2, page 18.
$\lambda v. E$ Evidence abstraction, Section 4.2, page 34.
If $v = v_1, \ldots, v_n$ is a list of evidence variables, then $\lambda v. E$ is used as an abbreviation for $\lambda v_1, \ldots, \lambda v_n. E$, Section 4.2, page 35.
$E e$ Evidence application, Section 4.2, page 34.
If $e = e_1, \ldots, e_n$ is a list of evidence expressions, then $E e$ is used as an abbreviation for $(\ldots (E e_1) \ldots) e_n$, Section 4.2, page 35.
$FV(E)$ Term variables appearing free in $E$, Section 3.1.2, page 18.
### Summary of Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>([E/x]F)</td>
<td>Capture free substitution of (E) for each free occurrence of (x) in (F), Section 3.1.2, page 18.</td>
</tr>
<tr>
<td>(id)</td>
<td>The identity term (\lambda x.x).</td>
</tr>
<tr>
<td>(Erase\ E)</td>
<td>Erasure of (E), Section 5.2, page 47.</td>
</tr>
</tbody>
</table>

### Predicates and evidence

- \(\pi, \pi', \ldots\) Predicates.
- \(P, Q, R, \ldots\) Predicate sets/assignments.
- \(e, f, g, \ldots\) Evidence expressions (also used to represent lists of evidence expressions), Section 4.2, page 34.
- \(u, v, w, \ldots\) Evidence variables (also used to represent lists of evidence variables), Section 4.2, page 34.
- \(v : P\) Predicate assignment. If \(v = u_1, \ldots, u_n\) is a list of evidence variables and \(P = \pi_1, \ldots, \pi_n\) is a list of predicates, then \(v : P\) is used as an abbreviation for the predicate assignment \(v_1 : \pi_1, \ldots, v_n : \pi_n\), Section 4.2, page 35.
- \(EV(e)\) Type variables appearing free in \(\sigma\), Section 4.2, page 34.
- \(\emptyset\) The empty set/predicate assignment, Section 2.1, page 7.
- \(P, Q\) Union (concatenation) of \(P\) and \(Q\), Section 2.1, page 7.
- \(P \vdash Q\) Predicate entailment, Section 2.1, page 6.
- \(P \vdash e : Q\) Construction of evidence \(e\) for \(Q\) from predicate assignment \(P\), Section 4.2, page 34.
- \(P \vdash e = f : Q\) Evidence equality judgement, Section 5.4.1, page 49.

### Types and type schemes

- \(\sigma, \eta, \ldots\) OML Type schemes, Section 3.1.1, page 17.
- \(\rho, \rho', \ldots\) Qualified types, Section 3.1.1, page 17.
- \(\tau, \nu, \mu, \ldots\) Simple types, Section 3.1.1, page 17.
- \(t, \alpha, \beta, \gamma, \ldots\) Type variables, Section 3.1.1, page 17.
- \(\forall T.\sigma\) Polymorphic type, Section 3.1.1, page 17.
Summary of Notation

\[ P \Rightarrow \sigma \]
Qualified type, Section 3.1.1, page 17.

If \( \sigma = Q \Rightarrow \tau \) and \( P, Q \) are predicate sets then \( P \Rightarrow \sigma \) is used as an abbreviation for \( (P, Q) \Rightarrow \sigma \), Section 3.1.1, page 17.

If \( P = \pi_1, \ldots, \pi_n \) is a list of predicates, then \( P \Rightarrow \sigma \) is used as an abbreviation for \( \pi_1 \Rightarrow \ldots \Rightarrow \pi_n \Rightarrow \rho \), Section 4.2, page 35.

\( (P|\sigma) \)
Constrained type scheme, Section 3.2.1, page 19.

\( T(\tau) \)
The set of all simple type expressions, Section 3.1.1, page 17.

\( TV(\sigma) \)
Type variables appearing free in \( \sigma \), Section 3.1.1, page 17.

\( AV(\sigma) \)
Ambiguous type variables in \( \sigma \), Section 5.8.3, page 64.

\( \text{Gen}(A, \rho) \)
Generalisation of \( \rho \) with respect to \( A \), Section 3.2.4, page 22.

\( (P|\sigma) \leq (Q|\eta) \)
Ordering on constrained type schemes, Section 3.2.1, page 20.

\( C : (Q|\eta) \Rightarrow (P|\sigma) \)
Conversion \( C \) from \( (Q|\eta) \) to \( (P|\sigma) \), Section 5.5, page 54.

\( C' \circ C \)
Composition of conversions, Section 5.5, page 55.

\( \tau \approx \tau' \)
Most general unifier \( U \) of \( \tau \) and \( \tau' \), Section 3.4.1, page 26.

Typing judgements and related notation

\( A, A', \ldots \)
Type assignments, Section 3.1.3, page 18.

\( \text{dom} A \)
Domain of type assignment \( A \), Section 3.1.3, page 18.

\( A, x: \sigma \)
Type assignment obtained from \( A \) by adding a new binding for \( x \), Section 3.1.3, page 18.

\( A(x) \)
Value assigned to \( x \) in type assignment \( A \), Section 3.1.3, page 18.

\( P | A \vdash E : \sigma \)
Typing judgement, Section 3.1.3, page 18.

\( P | A \not\vdash E : \sigma \)
Syntax-directed typing judgement, Section 3.3.1, page 23.

\( P | A \vdash E : \sigma \)
Type inference algorithm judgement, Section 3.4.2, page 26.

\( P | A \vdash E \Rightarrow F : \sigma \)
Typed translation judgement, Section 5.2, page 47.

\( P | A \vdash E \triangleright F : \sigma \)
Typed reduction judgement, Section 5.4.2, page 50.

\( P | A \vdash E = F : \sigma \)
Typed equality judgement, Section 5.4, page 49.

\( \vdash E = F \)
Equality of terms in all applicable contexts, Section 5.4, page 49.
Acknowledgements

One small page is not enough to do justice to the people that have helped me, either directly or indirectly, with the development and production of this thesis. In the first place, my thanks go to Keble College, Oxford, and the Programming Research Group, University of Oxford for the facilities that they have provided for my use, and to Richard Bird and Bernard Sufrit for their guidance as my supervisors during the past three years. I am also pleased to acknowledge the support of the Science and Engineering Research Council of Great Britain, without whose funding this work could never have been completed.

Particular thanks to Phil Wadler whose own work provides much of the motivation for this thesis and whose comments, during an enlightening week at the University of Glasgow in July 1990, helped me to develop the original definitions on which it is based. I am also very grateful to Simon Peyton Jones, Satish Thatte, Ignacio Trejos-Zelaya and Wayne Luk for their comments and interest in my work.

Thanks also to the ever-growing number of Gofer users for their feedback and suggestions during the past year. I am happy that this part of my work has already been useful to others and also grateful for the opportunities and contacts that it has given me in return.

On a personal note, I would like to thank my parents for their love and support. A final and special thank-you to my wife, Melanie, for reminding me of the better things in life and for believing in me when I didn’t.