Naoki Kobayashi (Ed.)

Programming Languages and Systems

LNCS 4279

4th Asian Symposium, APLAS 2006
Sydney, Australia, November 2006
Proceedings

Springer

Naoki Kobayashi (Ed.)

Programming Languages and Systems

4th Asian Symposium, APLAS 2006
Sydney, Australia, November 8-10, 2006
Proceedings

Springer

Volume Editor

Naoki Kobayashi
Tohoku University, Graduate School of Information Sciences
Department of Computer and Mathematical Sciences
6-3-9 Aoba, Aramaki, Aoba-ku, Sendai-shi, Miyagi 980-8579, Japan
E-mail: koba@ecei.tohoku.ac.jp

Library of Congress Control Number: 2006935552

CR Subject Classification (1998): D.3, D.2, F.3, D.4, D.1, F.4.1

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-48937-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-48937-5 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11924661 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 4279

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Lecture Notes in Computer Science

For information about Vols. 1–4200

please contact your bookseller or Springer

Vol. 4292: G. Bebis, R. Boyle, B. Parvin, D. Koracin, P. Remagnino, A. Nefian, G. Meenakshisundaram, V. Pascucci, J. Zara, J. Molineros, H. Theisel, T. Malzbender (Eds.), Advances in Visual Computing, Part II. XXXII, 906 pages. 2006.

Vol. 4291: G. Bebis, R. Boyle, B. Parvin, D. Koracin, P. Remagnino, A. Nefian, G. Meenakshisundaram, V. Pascucci, J. Zara, J. Molineros, H. Theisel, T. Malzbender (Eds.), Advances in Visual Computing, Part I. XXXI, 916 pages. 2006.

Vol. 4283: Y.Q. Shi, B. Jeon (Eds.), Digital Watermarking. XII, 474 pages. 2006.

Vol. 4281: K. Barkaoui, A. Cavalcanti, A. Cerone (Eds.), Theoretical Aspects of Computing - ICTAC. XV, 371 pages. 2006.

Vol. 4279: N. Kobayashi (Ed.), Programming Languages and Systems. XI, 423 pages. 2006.

Vol. 4278: R. Meersman, Z. Tari, P. Herrero (Eds.), On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, Part II. XLV, 1004 pages. 2006.

Vol. 4277: R. Meersman, Z. Tari, P. Herrero (Eds.), On the Move to Meaningful Internet Systems: OTM 2006 Workshops, Part I. XLV, 1009 pages. 2006.

Vol. 4276: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, Part II. XXXII, 752 pages. 2006.

Vol. 4275: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, Part I. XXXI, 1115 pages. 2006.

Vol. 4272: P. Havinga, M. Lijding, N. Meratnia, M. Wegdam (Eds.), Smart Sensing and Context. XI, 267 pages. 2006.

Vol. 4271: F.V. Fomin (Ed.), Graph-Theoretic Concepts in Computer Science. XIII, 358 pages. 2006.

Vol. 4270: H. Zha, Z. Pan, H. Thwaites, A.C. Addison, M. Forte (Eds.), Interactive Technologies and Sociotechnical Systems. XVI, 547 pages. 2006.

Vol. 4269: R. State, S. van der Meer, D. O'Sullivan, T. Pfeifer (Eds.), Large Scale Management of Distributed Systems. XIII, 282 pages. 2006.

Vol. 4268: G. Parr, D. Malone, M. Ó Foghlú (Eds.), Autonomic Principles of IP Operations and Management. XIII, 237 pages. 2006.

Vol. 4267: A. Helmy, B. Jennings, L. Murphy, T. Pfeifer (Eds.), Autonomic Management of Mobile Multimedia Services. XIII, 257 pages. 2006.

Vol. 4266: H. Yoshiura, K. Sakurai, K. Rannenberg, Y. Murayama, S. Kawamura (Eds.), Advances in Information and Computer Security. XIII, 438 pages. 2006.

Vol. 4265: N. Lavrač, L. Todorovski, K.P. Jantke (Eds.), Discovery Science. XIV, 384 pages. 2006. (Sublibrary LNAI).

Vol. 4264: J.L. Balcázar, P.M. Long, F. Stephan (Eds.), Algorithmic Learning Theory. XIII, 393 pages. 2006. (Sublibrary LNAI).

Vol. 4263: A. Levi, E. Savas, H. Yenigün, S. Balcisoy, Y. Saygin (Eds.), Computer and Information Sciences – ISCIS 2006. XXIII, 1084 pages. 2006.

Vol. 4261: Y. Zhuang, S.-Q. Yang, Y. Rui, Q. He (Eds.), Advance in Multimedia Information Processing - PCM 2006. XXII, 1040 pages. 2006.

Vol. 4260: Z. Liu, J. He (Eds.), Formal Methods and Software Engineering. XII, 778 pages. 2006.

Vol. 4259: S. Greco, Y. Hata, S. Hirano, M. Inuiguchi, S. Miyamoto, H.S. Nguyen, R. Słowiński (Eds.), Rough Sets and Current Trends in Computing. XXII, 951 pages. 2006. (Sublibrary LNAI).

Vol. 4257: I. Richardson, P. Runeson, R. Messnarz (Eds.), Software Process Improvement. XI, 219 pages. 2006.

Vol. 4256: L. Feng, G. Wang, C. Zeng, R. Huang (Eds.), Web Information Systems – WISE 2006 Workshops. XIV, 320 pages. 2006.

Vol. 4255: K. Aberer, Z. Peng, E.A. Rundensteiner, Y. Zhang, X. Li (Eds.), Web Information Systems – WISE 2006. XIV, 563 pages. 2006.

Vol. 4254: T. Grust, H. Höpfner, A. Illarramendi, S. Jablonski, M. Mesiti, S. Müller, P.-L. Patranjan, K.-U. Sattler, M. Spiliopoulou (Eds.), Current Trends in Database Technology – EDBT 2006. XXXI, 932 pages. 2006.

Vol. 4253: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part III. XXXII, 1301 pages. 2006. (Sublibrary LNAI).

Vol. 4252: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part II. XXXIII, 1335 pages. 2006. (Sublibrary LNAI).

Vol. 4251: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part I. LXVI, 1297 pages. 2006. (Sublibrary LNAI).

Vol. 4249: L. Goubin, M. Matsui (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2006. XII, 462 pages. 2006.

Vol. 4248: S. Staab, V. Svátek (Eds.), Engineering Knowledge in the Age of the Semantic Web. XIV, 400 pages. 2006. (Sublibrary LNAI).

Vol. 4247: T.-D. Wang, X. Li, S.-H. Chen, X. Wang, H. Abbass, H. Iba, G. Chen, X. Yao (Eds.), Simulated Evolution and Learning. XXI, 940 pages. 2006.

Vol. 4246: M. Hermann, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning. XIII, 588 pages. 2006. (Sublibrary LNAI).

Vol. 4245: A. Kuba, L.G. Nyúl, K. Palágyi (Eds.), Discrete Geometry for Computer Imagery. XIII, 688 pages. 2006.

Vol. 4244: S. Spaccapietra (Ed.), Journal on Data Semantics VII. XI, 267 pages. 2006.

Vol. 4243: T. Yakhno, E.J. Neuhold (Eds.), Advances in Information Systems. XIII, 420 pages. 2006.

Vol. 4241: R.R. Beichel, M. Sonka (Eds.), Computer Vision Approaches to Medical Image Analysis. XI, 262 pages. 2006.

Vol. 4239: H.Y. Youn, M. Kim, H. Morikawa (Eds.), Ubiquitous Computing Systems. XVI, 548 pages. 2006.

Vol. 4238: Y.-T. Kim, M. Takano (Eds.), Management of Convergence Networks and Services. XVIII, 605 pages. 2006.

Vol. 4237: H. Leitold, E. Markatos (Eds.), Communications and Multimedia Security. XII, 253 pages. 2006.

Vol. 4236: L. Breveglieri, I. Koren, D. Naccache, J.-P. Seifert (Eds.), Fault Diagnosis and Tolerance in Cryptography. XIII, 253 pages. 2006.

Vol. 4234: I. King, J. Wang, L. Chan, D. Wang (Eds.), Neural Information Processing, Part III. XXII, 1227 pages. 2006.

Vol. 4233: I. King, J. Wang, L. Chan, D. Wang (Eds.), Neural Information Processing, Part II. XXII, 1203 pages. 2006.

Vol. 4232: I. King, J. Wang, L. Chan, D. Wang (Eds.), Neural Information Processing, Part I. XLVI, 1153 pages. 2006.

Vol. 4231: J. F. Roddick, R. Benjamins, S. Si-Saïd Cherfi, R. Chiang, C. Claramunt, R. Elmasri, F. Grandi, H. Han, M. Hepp, M. Hepp, M. Lytras, V.B. Mišić, G. Poels, I.-Y. Song, J. Trujillo, C. Vangenot (Eds.), Advances in Conceptual Modeling - Theory and Practice. XXII, 456 pages. 2006.

Vol. 4229: E. Najm, J.F. Pradat-Peyre, V.V. Donzeau-Gouge (Eds.), Formal Techniques for Networked and Distributed Systems - FORTE 2006. X, 486 pages. 2006.

Vol. 4228: D.E. Lightfoot, C.A. Szyperski (Eds.), Modular Programming Languages. X, 415 pages. 2006.

Vol. 4227: W. Nejdl, K. Tochtermann (Eds.), Innovative Approaches for Learning and Knowledge Sharing. XVII, 721 pages. 2006.

Vol. 4226: R.T. Mittermeir (Ed.), Informatics Education – The Bridge between Using and Understanding Computers. XVII, 319 pages. 2006.

Vol. 4225: J.F. Martínez-Trinidad, J.A. Carrasco Ochoa, J. Kittler (Eds.), Progress in Pattern Recognition, Image Analysis and Applications. XIX, 995 pages. 2006.

Vol. 4224: E. Corchado, H. Yin, V. Botti, C. Fyfe (Eds.), Intelligent Data Engineering and Automated Learning – IDEAL 2006. XXVII, 1447 pages. 2006.

Vol. 4223: L. Wang, L. Jiao, G. Shi, X. Li, J. Liu (Eds.), Fuzzy Systems and Knowledge Discovery. XXVIII, 1335 pages. 2006. (Sublibrary LNAI).

Vol. 4222: L. Jiao, L. Wang, X. Gao, J. Liu, F. Wu (Eds.), Advances in Natural Computation, Part II. XLII, 998 pages. 2006.

Vol. 4221: L. Jiao, L. Wang, X. Gao, J. Liu, F. Wu (Eds.), Advances in Natural Computation, Part I. XLI, 992 pages. 2006.

Vol. 4219: D. Zamboni, C. Kruegel (Eds.), Recent Advances in Intrusion Detection. XII, 331 pages. 2006.

Vol. 4218: S. Graf, W. Zhang (Eds.), Automated Technology for Verification and Analysis. XIV, 540 pages. 2006.

Vol. 4217: P. Cuenca, L. Orozco-Barbosa (Eds.), Personal Wireless Communications. XV, 532 pages. 2006.

Vol. 4216: M.R. Berthold, R. Glen, I. Fischer (Eds.), Computational Life Sciences II. XIII, 269 pages. 2006. (Sublibrary LNBI).

Vol. 4215: D.W. Embley, A. Olivé, S. Ram (Eds.), Conceptual Modeling - ER 2006. XVI, 590 pages. 2006.

Vol. 4213: J. Fürnkranz, T. Scheffer, M. Spiliopoulou (Eds.), Knowledge Discovery in Databases: PKDD 2006. XXII, 660 pages. 2006. (Sublibrary LNAI).

Vol. 4212: J. Fürnkranz, T. Scheffer, M. Spiliopoulou (Eds.), Machine Learning: ECML 2006. XXIII, 851 pages. 2006. (Sublibrary LNAI).

Vol. 4211: P. Vogt, Y. Sugita, E. Tuci, C. Nehaniv (Eds.), Symbol Grounding and Beyond. VIII, 237 pages. 2006. (Sublibrary LNAI).

Vol. 4210: C. Priami (Ed.), Computational Methods in Systems Biology. X, 323 pages. 2006. (Sublibrary LNBI).

Vol. 4209: F. Crestani, P. Ferragina, M. Sanderson (Eds.), String Processing and Information Retrieval. XIV, 367 pages. 2006.

Vol. 4208: M. Gerndt, D. Kranzlmüller (Eds.), High Performance Computing and Communications. XXII, 938 pages. 2006.

Vol. 4207: Z. Ésik (Ed.), Computer Science Logic. XII, 627 pages. 2006.

Vol. 4206: P. Dourish, A. Friday (Eds.), UbiComp 2006: Ubiquitous Computing. XIX, 526 pages. 2006.

Vol. 4205: G. Bourque, N. El-Mabrouk (Eds.), Comparative Genomics. X, 231 pages. 2006. (Sublibrary LNBI).

Vol. 4204: F. Benhamou (Ed.), Principles and Practice of Constraint Programming - CP 2006. XVIII, 774 pages. 2006.

Vol. 4203: F. Esposito, Z.W. Raś, D. Malerba, G. Semeraro (Eds.), Foundations of Intelligent Systems. XVIII, 767 pages. 2006. (Sublibrary LNAI).

Vol. 4202: E. Asarin, P. Bouyer (Eds.), Formal Modeling and Analysis of Timed Systems. XI, 369 pages. 2006.

Vol. 4201: Y. Sakakibara, S. Kobayashi, K. Sato, T. Nishino, E. Tomita (Eds.), Grammatical Inference: Algorithms and Applications. XII, 359 pages. 2006. (Sublibrary LNAI).

Preface

This volume contains the proceedings of the 4th Asian Symposium on Programming Languages and Systems (APLAS 2006), which took place in Sydney, Japan, November 8-10, 2006. The symposium was sponsored by the Asian Association for Foundation of Software.

In response to the call for papers, 70 full submissions were received. Each submission was reviewed by at least three Program Committee members with the help of external reviewers. The Program Committee meeting was conducted electronically over a 2-week period. After careful discussion, the Program Committee selected 22 papers. I would like to sincerely thank all the members of the APLAS 2006 Program Committee for their excellent job, and all the external reviewers for their invaluable contribution. The submission and review process was managed using the CyberChair system.

In addition to the 22 contributed papers, the symposium also included two invited talks by Jens Palsberg (UCLA, Los Angeles, USA) and Peter Stuckey (University of Melbourne, Melbourne, Australia), and one tutorial by Matthew Flatt (University of Utah, USA).

Many people helped to promote APLAS as a high-quality forum in Asia to serve programming language researchers worldwide. Following a series of well-attended workshops that were held in Singapore (2000), Daejeon (2001), and Shanghai (2002), the first three formal symposiums were held in Beijing (2003), Taipei (2004) and Tsukuba (2005).

I am grateful to the General Co-chairs, Manuel Chakravarty and Gabriele Keller, for their invaluable support and guidance that made our symposium in Sydney possible. I would like to thank the AAFS Chair Tetsuo Ida and the Program Chairs of the past APLAS symposiums, Atsushi Ohori, Wei-Ngan Chin, and Kwangkeun Yi, for their advice. I am also thankful to Eijiro Sumii for serving as the Poster Chair. Last but not least, I thank Kohei Suenaga for his help in handling the CyberChair system and other administrative matters.

September 2006 Naoki Kobayashi

Organization

General Co-chairs

Manuel Chakravarty (University of New South Wales, Australia)
Gabriele Keller (University of New South Wales, Australia)

Program Chair

Naoki Kobayashi (Tohoku University)

Program Committee

Kung Chen (National Chengchi University, Taiwan)
Wei-Ngan Chin (National University of Singapore, Singapore)
Patrick Cousot (ENS, France)
Masahito Hasegawa (Kyoto University, Japan)
Jifeng He (United Nations University, Macau)
Haruo Hosoya (University of Tokyo, Japan)
Bo Huang (Intel China Software Center, China)
Oege de Moor (Oxford University, UK)
George Necula (University of California at Berkeley, USA)
Martin Odersky (EPFL, Switzerland)
Tamiya Onodera (IBM Research, Tokyo Research Laboratory, Japan)
Yunheung Paek (Seoul National University, Korea)
Sriram Rajamani (Microsoft Research, India)
Andrei Sabelfeld (Chalmers University of Technology, Sweden)
Zhong Shao (Yale University, USA)
Harald Sondergaard (University of Melbourne, Australia)
Nobuko Yoshida (Imperial College London, UK)

Poster Chair

Eijiro Sumii (Tohoku University)

External Referees

Amal Ahmed
Minwook Ahn
Hugh Anderson
Stefan Andrei
Puri Arenas
Aslan Askarov
Benjamin Aziz
Nick Benton
Martin Berger

Julien Bertrane
Bruno Blanchet
Frederic Blanqui
Matthias Blume
Hans Boehm
Iovka Boneva
Mihai Budiu
Cristiano Calcagno
Sagar Chaki
Avik Chaudhuri
Siau Cheng Khoo
Shigeru Chiba
Adam Chlipala
Doosan Cho
Jeremy Condit
Florin Craciun
Jason Dai
Cristina David
Xinyu Feng
Jérôme Feret
Cédric Fournet
Stephen Freund
Alexey Gotsman
Dan Grossman
Joshua Guttman
Huu Hai Nguyen
Matthew Harren
Aaron Harwood
Martin Hofmann
Kohei Honda
Hans Huttel
Atsushi Igarashi
Kazuhiro Inaba
Seokgyo Jung
Shin-ya Katsumata
Wonseok Kim
Yongjoo Kim
Yue-Sun Kuo
Akash Lal
Peeter Laud
Jooyeon Lee
Benjamin Leperchey
Francesco Logozzo
Youngmo Lyang
Sergio Maffeis
Laurent Mauborgne
Antoine Miné
Yasuhiko Minamide
David Monniaux
Shin-Cheng Mu
Lee Naish
Aleks Nanevski
Aditya Nori
Atsushi Ohori
Sanghyun Park
Andrew Phllips
Henrik Pilegaard
Bernie Pope
Corneliu Popeea
Shengchao Qin
Xavier Rival
Alejandro Russo
Sriram Sankaranarayanan
Jatin Shah
Alex Simpson
Lex Spoon
Tadahiro Suda
Eijiro Sumii
Yuanhao Sun
Michiaki Tatsubori
Kenjiro Taura
Peter Thiemann
Tayssir Touili
Yoshihito Toyama
Akihiko Tozawa
Daniele Varacca
Manik Varma
Jérôme Vouillon
Keith Wansbrough
Tony Wirth
Wuu Yang
Jonghee W. Yoon
Poonna Yospanya
Jonghee M. Youn
Zhiqiang Yu

Sponsoring Institutions

Asian Association for Foundation of Software (AAFS)
The University of New South Wales

Table of Contents

Invited Talk 1

Session 1

Session 2

Session 3

Invited Talk 2

Session 4

Session 5

Invited Tutorial

Session 6

Session 7

Session 8

Type Processing by Constraint Reasoning

Peter J. Stuckey[1,2], Martin Sulzmann[3], and Jeremy Wazny[2]

[1] NICTA Victoria Laboratory
[2] Department of Computer Science and Software Engineering
University of Melbourne, 3010 Australia
{pjs, jeremyrw}@cs.mu.oz.au
[3] School of Computing, National University of Singapore
S16 Level 5, 3 Science Drive 2, Singapore 117543
sulzmann@comp.nus.edu.sg

Abstract. Herbrand constraint solving or unification has long been understood as an efficient mechanism for type checking and inference for programs using Hindley/Milner types. If we step back from the particular solving mechanisms used for Hindley/Milner types, and understand type operations in terms of constraints we not only give a basis for handling Hindley/Milner extensions, but also gain insight into type reasoning even on pure Hindley/Milner types, particularly for type errors. In this paper we consider typing problems as constraint problems and show which constraint algorithms are required to support various typing questions. We use a light weight constraint reasoning formalism, Constraint Handling Rules, to generate suitable algorithms for many popular extensions to Hindley/Milner types. The algorithms we discuss are all implemented as part of the freely available Chameleon system.

1 Introduction

Hindley/Milner type checking and inference has long been understood as a process of solving Herbrand constraints, but typically the typing problem is not first mapped to a constraint problem and solved, instead a fixed algorithm, such as algorithm $\mathcal{W}$ using unification, is used to infer and check types. We argue that understanding a typing problem by first mapping it to a constraint problem gives us greater insight into the typing in the first place, in particular:

- Type inference corresponds to collecting the type constraints arising from an expression. An expression has no type if the resulting constraints are *unsatisfiable.*
- Type checking corresponds to checking that the declared type, considered as constraints, *implies* (that is has more information than) the inferred type (constraints collected from the definition).
- Type errors of various classes: ambiguity, subsumption errors; can all be explained better by reasoning on the type constraints.

Strongly typed languages provide the user with the convenience to significantly reduce the number of errors in a program. Well-typed programs can be guaranteed not to "go wrong" [22], with respect to a large number of potential problems.

N. Kobayashi (Ed.): APLAS 2006, LNCS 4279, pp. 1–25, 2006.

Typically type processing of a program either checks that types declared for each program construct are correct, or, better, infers the types for each program construct and checks that these inferred types are compatible with any declared types. If the checks succeed, the program is type correct and cannot "go wrong".

However, programs are often not well-typed, and therefore must be modified before they can be accepted. Another important role of the type processor is to help the author determine why a program has been rejected, what changes need to be made to the program for it to be type correct.

Traditional type inference algorithms depend on a particular traversal of the syntax tree. Therefore, inference frequently reports errors at locations which are far away from the actual source of the problem. The programmer is forced to tackle the problem of correcting his program unaided. This can be a daunting task for even experienced programmers; beginners are often left bewildered.

Our thesis is that by mapping the entire typing problem to a set of constraints, we can use constraint reasoning to (a) concisely and efficiently implement the type processor and (b) accurately determine where errors may occur, and aid the programmer in correcting them. The Chameleon [32] system implements this for rich Hindley/Milner based type languages.

We demonstrate our approach via three examples. Note that throughout the paper we will adopt Haskell [11] style syntax in examples.

Example 1. Consider the following ill-typed program:

```
f 'a' b    True = error "'a'"
f c    True z    = error "'b'"
f x    y    z    = if z then x else y
f x    y    z    = error "last"
```

Here `error` is the standard Haskell function with type $\forall a.[Char] \rightarrow a$. GHC reports:

```
mdef.hs:4:
    Couldn't match 'Char' against 'Bool'
        Expected type: Char
        Inferred type: Bool
    In the definition of 'f': f x y z = if z then x else y
```

What's confusing here is that GHC combines type information from a number of clauses in a non-obvious way. In particular, in a more complex program, it may not be clear at all where the *Char* and *Bool* types it complains about come from. Indeed, it isn't even obvious where the conflict in the above program is. Is it complaining about the two branches of the if-then-else (if so, which is *Char* and which *Bool*?), or about `z` which might be a *Char*, but as the conditional must be a *Bool*?

The Chameleon system reports:[1]

[1] The currently available Chameleon system (July 2005) no longer supports these more detailed error messages, after extensions to other parts of the system. The feature will be re-enabled in the future. The results are given from an earlier version.

```
multi.hs:1: ERROR: Type error - one error found
Problem : Definition clauses not unifiable
Types   : Char -> a -> b -> c
          d -> Bool -> e -> f
          g -> g -> h -> i
Conflict: f 'a' b True = error "'a'"
          f c True z = error "'b'"
          f x y z = if z then x else y
```

Note we do not mention the last definition equation which is irrelevant to the error.

If we assume the actual error is that the `True` in the second definition should be a `'b'` through some copy-and-paste error, then it is clear that the GHC error message provides little help in discovering it. The Chameleon error certainly implicates the `True` in the problem and gives type information that should direct the programmer to the problem quickly.

As part of the diagnosis the system "colours" both the conflicting types and certain program locations. A program location which contributes to any of the reported conflicting types is highlighted in the same style as that type. Locations which contribute to multiple reported types are highlighted in a combination of the styles of the types they contribute to. (There are no such locations in the case above.)

The above example illustrates the fundamental problems with any traditional Hindley/Milner type inference like algorithms $\mathcal{W}$ [22]. The algorithms suffer from a bias derived from the way they traverse the abstract syntax tree (AST). The second problem is that being tied to unification, which is only one particular implementation of a constraint solving algorithm for tree constraints, they do not treat the problem solely as a constraint satisfaction problem.

The problems of explaining type errors are exacerbated when the type system becomes more complex. Type classes [34] are an important extension to Hindley/Milner types, allowing principled (non-parametric) overloading. But the extension introduces new classes of errors and complicates typing questions. Type classes are predicates over types, and now we have to admit that type processing is a form of reasoning over first order formulae about types.

Example 2. Consider the following program which is typical of the sort of mistake that beginners make. The base case `sum [] = []` should read `sum [] = 0`. The complexity of the reported error is compounded by Haskell's overloading of numbers.

```
sum [] = []
sum (x:xs) = x + sum xs

sumLists = sum . map sum
```

GHC does not report the error in `sum` until a monomorphic instance is required, at which point it discovers that no instance of `Num [a]` exists. This means that unfortunately such errors may not be found through type checking

alone – it may remain undiscovered until someone attempts to run the program. The function `sumLists` forces that here, and GHC reports:

```
sum.hs:4:
No instance for (Num [a]) arising from use of 'sum' at sum.hs:3
Possible cause: the monomorphism restriction applied to the following:
  sumLists :: [[[a]]] -> [a] (bound at sum.hs:3)
Probable fix: give these definition(s) an explicit type signature
In the first argument of '(.)', namely 'sum'
In the definition of 'sumLists': sumLists = sum . (map sum)
```

The error message is completely misleading, except for the fact that the problem is there is no instance of `Num [a]`. The probable fix will not help.

For this program Chameleon reports the following:

```
sum.hs:4: ERROR: Missing instance
Instance:Num [a]: sum [] = []
                  sum (x:xs) = x + sum xs
```

This indicates that the demand for this instance arises from the interaction between `[]` on the first line of `sum` and `(+)` on the second. The actual source of the error is highlighted.

The advantages of using constraint reasoning extend as the type system becomes even more complex. Generalized Algebraic Data Types (GADTs) [3,36] are one of the latest extensions of the concept of algebraic data types. They have attracted a lot of attention recently [24,25,26]. The novelty of GADTs is that the (result) types of constructor may differ. Thus, we may make use of additional type equality assumptions while typing the body of a pattern clause.

Example 3. Consider the following example of a GADT, using GHC style notation, where `List a n` represents a list of `a`s of length `n`. Type constructors `Z` and `S` are used to represent numbers on the level of types.

```
data Z -- zero
data S n -- successor
data List a n where
  Nil :: List a Z
  Cons :: a -> List a m -> List a (S m)
```

We can now express much more complex behaviour of our functions, for example

```
map :: (a -> b) -> List a n -> List b n
map f Nil = Nil
map f (Cons a l) = Cons (f a) (map f l)
```

which guarantees that the `map` function returns a list of the same length as its input.

GADTs introduce more complicated typing problems because different bodies of the same function can have different types, since they act under different assumptions. This makes the job of reporting type errors much more difficult.

Example 4. Consider defining another GADT to encode addition among our (type) number representation.

```
data Sum l m n where
  Base :: Sum Z n n
  Step :: Sum l m n -> Sum (S l) m (S n)
```

We make use of the `Sum` type class to refine the type of the `append` function. Thus, we can state the desired property that the length of the output list equals the sum of the length of the two input lists.

```
append2 :: Sum l m n -> List a l -> List a m -> List a n
append2 Base Nil ys = Nil -- wrong!! should be ys
append2 (Step p) (Cons x xs) ys = Cons x (append p xs ys)
```

For this program GHC reports

```
append.hs:17:22:
    Couldn't match the rigid variable 'n' against 'Z'
      'n' is bound by the type signature for 'append2'
      Expected type: List a n
      Inferred type: List a Z
    In the definition of 'append2': append2 Base Nil ys = Nil
```

For this program Chameleon currently reports:

```
ERROR: Polymorphic type variable 'n' (from line 13, col. 56) instantiated by
append2 :: Sum l m n -> List a l -> List a m -> List a n
append2 Base Nil ys = Nil -- wrong!! should be ys
```

Here we can determine the actual locations that cause the subsumption error to occur. We could also give information on the assumptions made, though presently Chameleon does not. We aim in the future to produce something like:

```
append.hs:10: ERROR:  Inferred type does not subsume declared type
Problem: The variable 'm' makes the declared type too polymorphic
       Under the assumptions [l = Z] and [m = n] arising from
            append2 [Base] Nil ys = Nil
       Declared: Sum Z m m -> List a Z -> List a m -> List a m
       Inferred: Sum Z m m -> List a Z -> List a m -> List a Z
append2 Base Nil ys = Nil -- wrong!! should be ys
```

Our advantage is that we use a constraint-based system where we maintain information which constraints arise from which program parts. GHC effectively performs unification under a mixed prefix, hence, GHC only knows which 'branch' failed but not exactly where.

As the examples illustrate, by translating type information to constraints with locations attached we can use constraint reasoning on the remaining constraint problem. The constraint reasoning maintains which locations caused any inferences it makes, and we can then use these locations to help report error messages much more precisely. In this paper we show how to translate complex typing problems to constraints and reason about the resulting typing problems.