Formulär WGS 3

WENNER-GREN STIFTELSERNA

Inges (6 ex) till
Vetenskapssekretariatet
Wenner-Gren Center
Sveavägen 166, 23 tr.
113 46 STOCKHOLM

ANSÖKAN OM POSTDOKTORSSTIPENDIUM FÖR UTBILDNING I SVERIGE (utländsk postdoktor)

1. Sökande (svensk huvudman)

Efternamn	Förnamn (tilltalsnamn)		Akademisk titel
Jansson	Patrik		Docent
Institutionstillhörighet		Tel	E-mail
Data- och informationsteknik		031-7725415	patrikj@chalmers.se
Institutionsadress		Postnr	Ort
Chalmers		412 96	Göteborg

2. Utländsk postdoktor

Efternamn	Förnamn (tilltalsnamn)	Akademisk titel	Disputerad (år/månad/dag)
Ionescu	Cezar	PhD	2009/02/09
Institutionsadress i hemlandet			
Potsdam Institute for Climate Impact Research (PIK) P.O. Box 60 12 03, 14412 Postdam			

3. Tidsperiod som ansökan avser

Aug. 2010 – July 2011 (1y)

4. Projekttitel

Type Theory for Sustainability Modelling

5. Sammanfattning av ansökan (använd underrubriker; se anvisningar)

The PostDoc's education and competence

University studies in control engineering and computer science with strong mathematical slant, specialization in computational models of biological inspiration (neural networks, genetic algorithms). Thesis (at the Mathematics and Informatics department of the Freie Universität Berlin), on modeling the concept of "vulnerability" in global change studies, proving ability for interdisciplinary research.

Research project

We propose to use constructive type theory to specify computational models used in sustainability science. We focus on exploratory models, used to improve our understanding of socio-ecological systems and for policy advice, which are similar in structure but smaller and simpler than predictive models, used for forecasting and control. Past experience suggests that much can be achieved by the consequent application of a simple idea: to make explicit the type of every function used in a model. We aim to build on this foundation, and take it one step further through the use of constructive types. We hope that this effort will lead not only to more correct implementations of these models, but also to closer ties between the computer science and the sustainability communities, and to the uncovering and clarification of important concepts of sustainability science.

The PostDoc's importance for Chalmers

The proposed project is perfectly aligned with the vision "Chalmers - for a sustainable future" and Ionescu's background in both Computer Science and Sustainability makes him the perfect candidate for bridging the two disciplines. This application is a natural continuation of the bilateral collaboration established over the last few years between PIK and Chalmers. The CSE department at Chalmers already has world-class groups in Type Theory and Functional Programming and Ionescu's work will help spreading these results within Chalmers and Sweden.

6. Följande handlingar (6 ex) skall bifogas

Postdoktorens	
Meritförteckning (curriculum vitae)	Bil 1
Förteckning över publicerade vetenskapliga arbeten	Bil 2
Forskningsprogram (högst 5 sidor)	Bil 3
Huvudmannens	
Curriculum vitae (högst 2 sidor, inkl. lista med 5 utvalda publikationer)	Bil 4

7. Från annat håll sökt eller erhållet anslag eller stipendium

☐ sökts	☐ erhållits
☒ ej sökts	☐ ej erhållits

Från:

Belopp:

8. Uppgift om avlöningsförhållanden

Uppbär postdoktoren lön från heminstitution ☐ Ja ☒ Nej

Om lön utgår, belopp:

9. Om postdoktoren vid ansökningstillfället är i Sverige,

datum när han/hon kom till Sverige ----

10. Tidigare stipendium från WGS för aktuell gästforskare

Tidsperiod ---

Härmed försäkras att de i ansökan lämnade uppgifterna är med verkligheten överensstämmande och att erhållet bidrag får offentliggöres.

Göteborg

(Ort)

20090930

(Datum)

(Namnteckning)

Type Theory for Sustainability Modelling
Project description

Cezar Ionescu Patrik Jansson

September 30, 2009

1 Abstract

We propose to use constructive type theory to specify computational models used in sustainability science. We hope that this will lead not only to more correct implementations of these models, but also to closer ties between the computer science and the sustainability communities, and to the uncovering and clarification of important concepts of sustainability science.

2 Exploratory models in sustainability science

Sustainability science studies "socio-ecological systems" [8], which are commonly modelled by using interacting dynamical systems representing economical agents and physical entities such as the atmosphere, the oceans, forests, cultivated lands, etc. The interactions between these systems often have consequences which are difficult to predict. Especially for decision making, it is vital that (at least) the space of possible consequences is explored.

Accordingly, we propose to concentrate on the class of models which are used to explore the "first-order effects" of decisions. Such models are meant to help us understand the interactions between the components of socio-ecological systems and to assist policy makers. This is in contrast to the class of predictive models, whose aim is to obtain accurate forecasts or optimal policies. Exploratory models are generally simpler than predictive ones, but they usually have the same structure. Predictive models usually just add more details to that structure, for example additional models to account for albedo effects in models of climate change, or use higher resolutions and more computationally expensive higher accuracy methods.

Although smaller in scale, the explorative models are still a very heterogeneous class: continuous models versus discrete, components with very different time scales (slow for the climate system, fast for the economic one), small number of aggregated agents versus large number of very simple agents, and so on. The same decision problem can be modelled by virtually any combination of these characteristics, which makes the results difficult to compare.

At the same time, there are many similarities between these very different models, owing to the fact that they are, after all, supposed to represent similar entities and processes. One could expect that certain components could be reused between implementations, but this is very rarely the case. On one hand, this is due to the variety of environments used for implementation: these range from tools for formulating and integrating systems of differential

equations, such as Vensim [4] or Simile [3], to platforms for discrete-time multi-agent simulations, such as Swarm [2] or Repast [1]. This makes direct reuse of the code difficult. On the other hand, the design of these components is often described either in an informal way (such as a narrative or diagrams), or using high-level mathematical notation (for example, giving the partial differential equations that were used to implement a small climate model): in any case, the description will miss important parts of the context, which means that the design will also not be easily reusable.

What is missing here are specifications of the computational model, as opposed to the mathematical one or to an informal description of the model. This is not particularly surprising: lack of specifications is quite common in the scientific programming community: for example, a recent reference book, "Writing Scientific Software", with the subtitle "A guide to good style", does not mention the word "specification" even once [13]. The reason for this is perhaps that for many classical numerical problems, for instance linear programming, there exist precise mathematical formulations of the algorithms (such as the simplex method), together with analysis of stability and accuracy results. In such a situation, any other kind of specification is perceived as redundant, overkill.

However, this is not the situation of the vast majority of exploratory models. As we mentioned in the beginning, these models are characterised by interacting dynamical systems, which are often non-linear, implemented in terms of optimisation of non-linear functions (as is the case in many economic models), integration of non-linear systems of partial differential equations (perhaps with stochastic components), or just heuristic "rules of thumb". No numerical analysis of such components is available, and it is difficult to distinguish, for example, errors caused by round-off effects from those caused by a poor choice of an optimisation method. Moreover, we can also not rely on our intuition when deciding whether a simulation has given correct results: exploratory models are likely to give "unexpected" results. After all, if we could characterise the space of possibilities from the beginning, we might not need these models at all. But this means that in the absence of specifications, we cannot be sure that we are seeing the emergent behaviour of the interacting systems we are modelling, instead of the emergent behaviour of our implementation errors.

3 Using constructive type theory for specifications

Functional programming languages, with their clean syntax, staying close to the original mathematical notation from which they borrow their expressive type systems, have often been proposed as the appropriate vehicle for writing specifications (see, for example, [14, 7]). We have successfully used the functional programming language Haskell [6] as a specification language in various contexts, ranging from describing algorithms for relation-based computations in a distributed setting [P3, P9][1], to software components for computational vulnerability assessment [P5, P6]. We have found that Haskell notation, for example, is easily picked up by scientists, even by those who do not usually program. As always, mathematical training helps, but since the Haskell specifications are actually executable, one can actually reach some understanding by just running them.

Haskell specifications tend to have three elements: types, implementations and tests. However the Hindley-Milner type system which Haskell uses does not allow us to express all the properties in types: some of these will necessarily have to be expressed at the level

[1] References of the form "Pn" refer to the n'th paper in Ionescu's publication list."

of the implementation. Implementations, in turn, tend to have much more detail than just needed to express the specification, and moreover are the most fault-prone level of software development. Hence, the usage of tools such as QuickCheck, which allow one to both express some of the properties of the specification and to automatically test the correctness of the implementation. As a simple example: in Haskell, the type of a sorting function is

$$sort :: Ord\ a \Rightarrow List\ a \rightarrow List\ a.$$

This tells us about *sort* that it takes a list as input and returns a list as output, and (from the $Ord\ a \Rightarrow$ clause) that the elements of this list are required to be comparable (so that they can be ordered). The same type is attributed to a function that outputs a list of the maximal elements in the input, and to many other functions as well. In order to distinguish *sort* from these other functions, we would like to express the requirement that it should output an ordered permutation of the input, but we cannot do that at the level of types. The implementation contains this information, of course, at least if it is correct, but it also contains a lot of other details too (for example, the selection of an actual sorting algorithm). This is not what scientists from, say, climate change research want to see. Thus, we write a test function which then carries most of the specification:

$$prop\_sort\ xs = permutation\ xs\ ys \wedge ordered\ ys$$
$$\textbf{where}\ ys = sort\ xs$$

(assuming some implementation of the predicates *permutation* and *ordered*).

In other cases, the tests do not tell the entire story. Consider the case of partial functions, such as addition on vectors, which is only defined for vectors of the same size. The type of this addition in Haskell is, say, $Num\ a \Rightarrow Vector\ a \rightarrow Vector\ a \rightarrow Vector\ a$, which does not express the precondition at all. Therefore, testing the equality of the sizes of the arguments can only be done *within* the implementation of the function.

Finally, there are situations in which we simply cannot express the desired properties at all! We mentioned in the previous section that for the trial-and-error approach of exploratory programming used in sustainability science there are often no guarantees that the round-off errors induced by using floating point representations will not unduly influence the results. It is not always easy to determine whether a surprising result of a simulation is due to a clever trial or an insidious numerical error. What we can sometimes do, however, is to prove that, if the algorithm would act on "real" real numbers, certain desired properties would hold. This is typically the case when using transitivity and monotonicity arguments to ensure that the investments of an agent will not exceed its capital, that wages will not become negative, and so on. However, we cannot express this "what-if" scenario at any of the levels of the Haskell specification.

This is how we came to the idea of using constructive type theory for specifying the kinds of models described in the previous section. Because of the good experiences we have had with Haskell (and the less pleasant ones with the alternatives), we were reluctant to abandon the usage of a functional programming language for specifications. Now, constructive type theory can be seen as a functional programming language [11], and, in fact, in the Chalmers implementation Agda [12], one which is rather similar to Haskell. In many respects, Agda's syntax is cleaner and closer to mathematics. But, most importantly, the type system is strong enough to express the entire specification, and "what-if" scenarios are available as *postulates*. The implementation, then, will be correct simply by virtue of having the correct type, and it

will show that the specification is consistent. We believe that having the specifications in one place, at the level of types, will improve their understandability and will make them easier to share with scientists from different disciplines.

When using Haskell as a specification language for components used in computational vulnerability assessments in the field of climate change, we have been led to a perhaps more valuable result: we were able to formalise and unify the various definitions of the elusive concept of "vulnerability", and share the understanding that we thereby reached with the broader climate change community [P1, P2]. We hope that the usage of Agda will help us find, analyse and better understand other key concepts of sustainability science.

4 Research plan

We will start by analysing typical exploratory models, among which we mention simplified versions of the Lagom model of German economy [10], the Madiams models of Klaus Hasselman et al. [9, 15] and the models of the energy sector currently under development at Chalmers by Kristian Lindgren and Claes Andersson. It is important to note that all the authors of these models have agreed to help in the attempt of finding the right computational descriptions: otherwise, considering the amount of domain knowledge that is incorporated in even the simplest of these models, it is unlikely that the analysis could be usefully done in a reasonable time.

Within this interdisciplinary activity, we will attempt to understand and express the types of the main components of the models. This will lead to a first kind of simple specifications, at the level of Haskell types, which can be represented diagrammatically (similar to the diagrams of category theory, or the graphs in Vensim [4]). This first rough cut will then be polished by adding more significant details, using the additional expressiveness of the dependent type system, until we obtain a number of Agda (re)implementations of the core components of these models.

Special attention will be paid to separating the discrete mathematical aspects from the continuous ones. For example, it is often the case that economic agents can be described at one level as (possibly non-deterministic) finite automata, while at another they are (possibly stochastic) continuous-time dynamical systems. For example, the agents can be in one of a finite number of states, say "producing", "consuming", "engaged in trading", in which they receive queries about "profit" or "wage", while the representations of these states and queries will involve real vector spaces and functions. When dealing with the description and implementation of the latter components, we will study "postulational" solutions (as described in the previous section and implemented in the Coq standard library [5]) versus implementation of exact constructive real numbers. At the moment, there exists no implementation of constructive real numbers which could be used for our task (and for scientific programming in general): it would be a desirable outcome of our project if we could at least identify the most important components of such an implementation.

We hope to ensure the usability of the specifications we develop by making sure that they are understandable to all the various scientists involved, and by implementing them in various programming languages and environments.

Finally, we will collect the various specifications in a library, with links to the implementations available for each specification. In this way, we hope to improve the modelling ability of the scientists involved in sustainability science.

References

[1] Repast: Recursive Porous Agent Simulation Toolkit. http://repast.sourceforge.net/.

[2] The Swarm Development Group. http://www.swarm.org/.

[3] System dynamics and object-based modelling and simulation software. http://www.simulistics.com/.

[4] Vensim, from Ventana Systems, Inc. http://www.vensim.com/.

[5] Y. Bertot and P Castran. *Interactive theorem proving and program development, Coq'art: the calculus of inductive constructions*. Texts in Theoretical Computer Science. Springer, 2004.

[6] S. L. Peyton Jones et al. *Haskell 98 Language and Libraries: the Revised Report*. Cambridge University Press, 2003.

[7] A. Frank and W. Kuhn. Specifying open GIS with functional languages. *Lecture Notes in Computer Science*, 951, 1995.

[8] G. Gallopin. Linkages between vulnerability, resilience, and adaptive capacity. *Global Environmental Change*, 16(3):293–303, 2001.

[9] K. Hasselmann. Simulating human behavior in macroeconomic models applied to climate change. *Accepted for publication in the Proceedings of the 98th Dahlem Workshop of Dahlem Conferences: 'Is There a Mathematics of Social Entities?'*, 2008.

[10] C. Jaeger, A. Mandel, S. Fürst, W. Lass, and F. Meissner. Lagom generic: A minimal description from an economic point of view. *Submitted to the ECF-GSD Modeling Workshop on Agent-Based Modeling for Sustainable Development, Venice 2-4 April 2009*, 2009.

[11] P. Martin-Löf. Constructive mathematics and computer programming. *Philosophical Transactions of the Royal Society of London*, 312(1522):501–518, 1984.

[12] U. Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Chalmers University of Technology, 2007.

[13] S. Oliveira and D. E. Stewart. *Writing Scientific Software: A Guide to Good Style*. Cambridge University Press, 2006.

[14] D. Turner. Functional programs as executable specifications. *Philosophical Transactions of the Royal Society of London*, 312(1522):363–388, 1984.

[15] M. Weber, V. Barth, and K Hasselmann. A multi-actor dynamic integrated assessment model (madiam) of induced technological change and sustainable economic growth. *Ecological Economics*, 54(2-3):306–327, 2005.

Curriculum Vitae

Cezar Ionescu

Personal Details

Name:	Cezar Ionescu
Date of birth:	21.12.1968
Nationality:	Romanian
Address:	Reiterweg 7
	14469 Potsdam, Germany
Email:	ionescu@pik-potsdam.de

Education

1988-1993 "Politehnica" University of Bucharest, Faculty of Automatic Control and Computer Science.

June 1993 Masters Thesis on "Hardware implementation of Neural Networks".

February 2009 PhD in Mathematics at the Freie Universität Berlin with the thesis "Vulnerability Modeling and Monadic Dynamical Systems".

Work Experience

Sep 1993 - Jul 1999 Systems analyst at the Informatics Research Institute in Bucharest. Worked within the Artificial Intelligence laboratory on projects related to optimization and adaptive control using neural networks, fuzzy logic and genetic algorithms. In 1998 worked on a Y2K project for Cap Gemini Nederland where I implemented several tools for analyzing COBOL code and assisting programmers in the removal of Y2K-related bugs.

Aug 1999 - July 2006 IT position at the Potsdam Institute for Climate Impact Research (PIK), project Modenv ("Modeling Environment"), wrote a Fortran code analysis tool for coupling legacy models with Corba, a simulation builder application for remote distributed components ("Graphical Simulation Builder"), participated in the design and implementation of the "Typed

Data Transfer" library and was responsible for the Python version.

Aug 2006 - June 2009 Scientific position at PIK. Worked within the project FAVAIA ("Formal Approaches to Vulnerability, Adaptation and Integrated Assessment"). I formalized the concept of "vulnerability" as used in the global change community and wrote my dissertation on vulnerability, advisor Rupert Klein. I continued to work on IT-related projects, among them the PIK project S ("Software components for distributed adaptive finite volume methods"), where I formulated a mathematical model for a class of parallel programs and assisted in the specification of relation-based algorithms.

since July 2009 Postdoc position at the Potsdam Institute for Climate Impact Research, project "Model Specification and Program Development". Main activity is writing software specifications for developing, testing, extending and re-factoring the actual implementations of PIK models, with focus on Lagom (a model of the German economy) and on REMIND (a multi-regional model of global economy with emphasis on the energy sector).

Other Activities

1994–1998 Part-time work for the *Nemira* publishing house. I have translated three books from English to Romanian, and have co-translated three anthologies from Romanian to English.

2001-present Together with Rupert Klein, initiated and led the "Cartesian Seminar" at PIK, a weekly forum for in-depth exchange of ideas and information among scientists from various disciplines.

Publication list for Cezar Ionescu

Refereed publications

1. C. Ionescu, R. J. T. Klein, J. Hinkel, K. S. Kavi Kumar and R. Klein. Towards a formal framework of vulnerability to climate change. In *Environmental Modelling and Assessment*, volume 14, issue 1, pages 1–16, Springer Netherlands, 2009.

2. C. Ionescu. *Vulnerability Modeling and Monadic Dynamical Systems.* PhD thesis, Fachbereich Mathematik und Informatik, Freie Universität Berlin, Germany, February 2009.

3. N. Botta and C. Ionescu. Relation based computations in a monadic BSP model. In *Parallel Computing*, volume 33, pages 795–821, Elsevier, 2007.

4. C. Ionescu and N. Botta. Modeling Versus Formalization. Accepted for publication in the *Proceedings of the 98th Dahlem Workshop of Dahlem Conferences: 'Is There a Mathematics of Social Entities?', 2008*, to be published in the "Dahlem Workshop Report" series of the MIT Press.

5. D. Lincke, C. Ionescu and N. Botta. A generic library for earth system modeling based on monadic systems. In proceedings of *Digital Earth Summit on Geoinformatics: Tools for Global Change Research*, M. Ehlers, K. Behncke and F. Gerstengarbe Editors, Wichmann, 2008.

6. D. Lincke, P. Jansson, M. Zalewski and C. Ionescu. Generic Libraries in C++ with Concepts from High-Level Domain Descriptions in Haskell. A Domain-Specific Library for Computational Vulnerability Assessment. In *Domain-Specific Languages, Proceedings of the IFIP TC 2 Working Conference DSL 2009, Oxford, UK, July 15-17 2009*, Editor W. M. Taha, 2009.

7. M. Zalewski, A. Priesnitz, C. Ionescu, N. Botta, and S. Schupp. Multi-Language Library Development. From Haskell Type Classes to C++ Concepts. In *Multiparadigm Programming with Object-Oriented Languages, an ECOOP workshop*, 2007.

Edited books

8. P. Flondor and C. Ionescu. *Introduction to Genetic Algorithms.* All Publishing House, Bucharest, 1999. (in Romanian).

Technical Reports

9. N. Botta, C. Ionescu, C. Linstead and R. Klein. *Structuring distributed relation-based computations with SCDRC.* PIK Report No. 103, Potsdam Institute for Climate Impact Research, Potsdam, 2006.

10. C. Ionescu, R. J. T. Klein, J. Hinkel, K. S. Kavi Kumar and R. Klein. *Towards a formal framework of vulnerability to climate change.* NeWater Working Paper 2 and FAVAIA Working Paper 1, Potsdam, 2005.

11. S. Wolf, C. Ionescu, D. Lincke, S. Bisaro, J. Hinkel and D. Reckien. *A Formal Framework of Vulnerability. Deliverable to the ADAM Project*, FAVAIA Working Paper 6, Potsdam Institute for Climate Impact Research, Potsdam, 2008.

Curriculum Vita: Patrik Jansson, 1972-03-11

Higher education degrees and longer research visits:

1995: B.Sc. + M.Sc. degrees in Eng. Physics + Eng. Mathematics. I graduated almost two years before schedule as the best student of my year.

2000: Ph.D. degree in Computing Science, *Functional Polytypic Programming*, Advisor: Johan Jeuring.

2004: Docent (Associate Prof.) degree in Computing Science.

1998, 1998, 2001: Research visits (2 + 2 + 3 months) to Northeastern University, Boston, USA; Oxford University Computing Lab, UK; Dept. of Computer Science, Yale, USA.

Current and Previous Employment:

2004–now: Associate Prof. in Computing Science, Chalmers.

2001–2004: Assistant Prof. in CS, Chalmers.

1991–1992: Military service: National Defence Radio Establishment (FRA). Hand picked (as one out of five per year in Sweden) for 15 months of special education in mathematics, statistics and cryptographic analysis.

Parental leave: With Julia (1999) and Erik (2004) for a total of one full time year.

Supervision experience:

I was PhD advisor of Ulf Norell (PhD 2007) and Nils Anders Danielsson (PhD 2007). With Norell I worked on generic programs and proofs and with Danielsson I worked on software verification and semantics. Both Norell and Danielsson were immediately employed as Post-Docs after their PhD (Norell here at Chalmers and in Danielsson in Nottingham). I have also supervised several MSc students and BSc project students. I currently supervise Jean-Philippe Bernardy (expected PhD 2012).

Awards, grants, etc.:

Main applicant on the project *Generic Functional Programs and Proofs* funded with 1800k SEK by the Swedish Research Council (2003–2005).

Co-applicant on *Cover — Combining Verification Methods in Software Development* funded with 8000k SEK by the Swedish Foundation for Strategic Research (2003–2005).

Obtained travel grants (277k SEK in total) from several private foundations

Received the John Ericsson medal for outstanding scholarship, Chalmers (1996). Represented Sweden in the International Mathematics Olympiad (1991) and the International Physics Olympiads (1991). Winner of the Swedish National Physics Olympiad (1991).

Leadership experience:

2009: Head of the steering group of Chalmers eScience Initiative.

2008–2010: Deputy project leader of the IMPACT project at Chalmers ("Development of Chalmers' New Master's Programmes", 900k EUR/year over three years).

2005–2008: Vice head of department responsible for the BSc and MSc education at the CSE department. The CSE education includes around 120 yearly courses offered to about 20 different programmes at 2 universities (Chalmers and University of Gothenburg).

2002–2005: Director of Studies (DoS) for the BSc and MSc education at the CS department

2002–2008: Member of the steering group of the department.

Selected Publications: Patrik Jansson

Note to non computer scientists Conference articles in computer science are peer reviewed full articles — not 1–2 page abstracts, and are the normal form of refereed publication. The top conferences in each subfield (like *POPL* below) typically have the highest impact factor within that field. All articles listed below are selected for publication by a peer review process.

Most cited publications (Google Scholar)

Jansson's Hirsch-index is 13 and the following papers are the five most cited.

1. P. Jansson and J. Jeuring. PolyP — a polytypic programming language extension. In *POPL'97: Principles of Programming Languages*, pages 470–482. ACM Press, 1997.
   Number of citations: 233.

2. R. Backhouse, P. Jansson, J. Jeuring, and L. Meertens. Generic programming: An introduction. In *Advanced Functional Programming*, volume 1608 of *LNCS*, pages 28–115. Springer-Verlag, 1999.
   Number of citations: 134.

3. J. Jeuring and P. Jansson. Polytypic programming. In J. Launchbury et al., editors, *Advanced Functional Programming '96*, volume 1129 of *LNCS*, pages 68–114. Springer-Verlag, 1996.
   Number of citations: 129.

4. P. Jansson. *Functional Polytypic Programming.* PhD thesis, Computing Science, Chalmers University of Technology and Göteborg University, Sweden, May 2000.
   Number of citations: 38.

5. P. Jansson and J. Jeuring. Functional pearl: Polytypic unification. *Journal of Functional Programming*, 8(5):527–536, 1998.
   Number of citations: 35.