Towards type-driven data-science in Idris

Anonymous Author(s)

1 Introduction

The traditional data-analysis pipe-line comprises of several layers, each of which involving nuanced dependencies on the shape and content of the processed data. Inference code depends on the shape of the model. The shapes of the results of tabular operations on data-frames depend on the shape and values stored of the input frames. When cleaning data, the shape of the parse-tree depends on the regex literal constructed to parse it. The legend of a figure depends on the number of graphs it visualises. Modern pipelines are thus developed in dynamically-typed languages, perhaps because simple-type systems are incapable of expressing such dependencies — especially tracking intermediate array dimensions statically. Most recently, languages experiment with dependently, and nearly-dependently, typed techniques that include just enough type-dependency to express such constraints [Henriksen et al. 2017; Paszke et al. 2021].

We take a different approach, and use a fully-fledged dependently typed language to use the full expressive power of type-dependency. Such languages, like Agda [Norell 2007] and Idris [Brady 2011, 2013], also offer interactive development environments that take advantage of the available type information to automatically construct parts of the program without running it. When some data is available statically, features such as dependent type providers [Christiansen 2013] can load some of it during type-checking, further blurring the distinction between dynamic and static without compromising on the robustness guarantees a static type system provides. The recently released Idris 2 language [Brady 2021] offers an additional axis of *erasure*: its unique implementation of Atkey and McBride's quantitative type theory [Atkey 2018] allow programmers to include complex dependencies in their pipeline while retaining fine control over their presence at runtime, providing space and time efficiency guarantees.

We would like to demonstrate some of the data-analysis pipeline components we have been developing in Idris 2 over the last couple of years. Our goal is to solicit feedback and discussion with the LAFI community about the challenges ahead, promising leads, interesting directions, and related work.

2 TyRE: type-driven regex parsing

A core task in data analysis is to extract data, either by mining it from raw data or by analysing existing textual data fields. A key swiss-army knife is the regular expression (regex), which admits efficient recognising and parsing. Since our goal is to extract data, pipelines often use regex *capture groups*. The users write regex literals contain markers for capturing fragments of the matched regexes. The relationship between the regex literal and the shape of the capture group is nuanced, and a simply-typed host language would need native support for regex literals. In a dependently-typed language, we can parse regex literals statically with library code, and implement Radanne's Typed RegEx (TyRE) layer [Radanne 2019] that returns a structured parse-tree instead of an unstructured capture group. Radanne's implementation translates the TyRE layer to an untyped regex with capture groups since his OCaml ecosystem already contains efficient regex engines. Since Idris 2 doesn't yet possess such mature software, we implement a dependently-typed regex parser that is guaranteed to return a parse tree of the correct shape.

3 Tables: statically-typed tabular data

A substantial part of data analysis concerns storing the data and intermediate results in tabular form. These are often accompanied by an efficient implementation, e.g., an interface to a database with data operations translating into optimised queries, although we make no attempts to do so at the moment. The schema of a result-table often depends on the values stored in an input-table, for example, pivoting a collection of rows into a collection of columns.

Using dependent types to represent schema and tables is a well-established idea [Oury and Swierstra 2008]. Our contribution is a conformant API to the recent Brown Benchmark for Table Types (B2T2) [Lu et al. 2022]. When the source schema are available statically, the primitive compute resulting schema statically, maintaining an interactive notebook-like feel to the data exploration.

4 ProbFX and Idris-Bayes: modular statistical modelling and inference

We re-implement recent development in modular statistical modelling [Nguyen et al. 2022] and Bayesian inference [Ścibior et al. 2018]. Here our goal was merely to replicate the state-of-the-art, but we believe dependent types offer much room for more nuanced inference such as trace types [Lew et al. 2020] and similar extensions.

5 Jupyter-Vega: data visualisation binding

We provide bindings to the award-winning Vega-Lite format [Satyanarayan et al. 2017], and a simple Jupyter Notebook kernel for invoking Idris 2 modules and printing their return values and visualisation. We believe there is much

LAFI'23, January 15-21, 2023, Boston, MA
2023. ACM ISBN 00…$00.00
https://doi.org/XXXXXXX.XXXXXXX

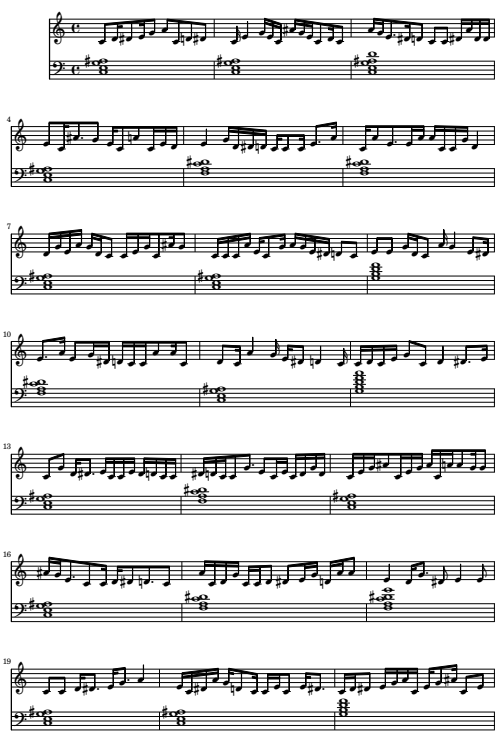

Figure 1. Melocule-generated blues sample

room for type-driven development here, and we are excited to discuss these prospects with workshop participants.

6 Idris-ODF: data reporting binding

Data needs to be retrieved from and output into office-application formats such as ISO OpenDocument Format (ODF, ISO standard ISO/IEC 26300-1:2015). This format include spreadsheets and word-processors, and we have implemented basic bindings for ODF. It is suggestive to develop application-specific dependently-typed layers on top of these bindings to facilitate type-driven reporting and importing.

7 Melocule: generative music

We are currently developing a simple generative music library using these tools. To this end, we developed a MiDi bindings library for Idris 2, and used it to formulate basic concepts in music theory. These include some dependently-typed representation, for example scales are indexed by Major/Minor qualities to ensure scales are well-defined. Fig. 1 shows a sample blues piece we generated. We are currently working on incorporating Bayesian conditioning into our generative models.

8 Conclusion and prospects

To summarise, we have proto-typed core data-analysis components in Idris 2, and are excited by their existing and potential prospects for data-set exploration, statistical modelling, and pipeline deployment. We hope to be able to present them to the LAFI community this year in Boston.

References

Robert Atkey. 2018. Syntax and Semantics of Quantitative Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, Anuj Dawar and Erich Grädel (Eds.). ACM, 56–65. https://doi.org/10.1145/3209108.3209189

Edwin Brady. 2021. Idris 2: Quantitative Type Theory in Practice. In *35th European Conference on Object-Oriented Programming (ECOOP 2021) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 194)*, Anders Møller and Manu Sridharan (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 9:1–9:26. https://doi.org/10.4230/LIPIcs.ECOOP.2021.9

Edwin C. Brady. 2011. IDRIS —: Systems Programming Meets Full Dependent Types. In *Proceedings of the 5th ACM Workshop on Programming Languages Meets Program Verification* (Austin, Texas, USA) *(PLPV '11)*. Association for Computing Machinery, New York, NY, USA, 43–54. https://doi.org/10.1145/1929529.1929536

Edwin C. Brady. 2013. Idris: General Purpose Programming with Dependent Types. In *Proceedings of the 7th Workshop on Programming Languages Meets Program Verification* (Rome, Italy) *(PLPV '13)*. Association for Computing Machinery, New York, NY, USA, 1–2. https://doi.org/10.1145/2428116.2428118

David Raymond Christiansen. 2013. Dependent type providers. In *Proceedings of the 9th ACM SIGPLAN workshop on Generic programming, WGP 2013, Boston, Massachusetts, USA, September 28, 2013*, Jacques Carette and Jeremiah Willcock (Eds.). ACM, 25–34. https://doi.org/10.1145/2502488.2502495

Troels Henriksen, Niels G. W. Serup, Martin Elsman, Fritz Henglein, and Cosmin E. Oancea. 2017. Futhark: Purely Functional GPU-Programming with Nested Parallelism and in-Place Array Updates. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Barcelona, Spain) *(PLDI 2017)*. Association for Computing Machinery, New York, NY, USA, 556–571. https://doi.org/10.1145/3062341.3062354

Alexander K. Lew, Marco F. Cusumano-Towner, Benjamin Sherman, Michael Carbin, and Vikash K. Mansinghka. 2020. Trace types and denotational semantics for sound programmable inference in probabilistic languages. *Proc. ACM Program. Lang.* 4, POPL (2020), 19:1–19:32. https://doi.org/10.1145/3371087

Kuang-Chen Lu, Ben Greenman, , and Shriram Krishnamurthi. 2022. Types for Tables: A Language Design Benchmark. *The Art, Science, and Engineering of Programming* 6, 2 (2022), 26 pages.

Minh Nguyen, Roly Perera, Meng Wang, and Nicolas Wu. 2022. Modular Probabilistic Models via Algebraic Effects. *Proc. ACM Program. Lang.* 6, ICFP, Article 104 (aug 2022), 30 pages. https://doi.org/10.1145/3547635

Ulf Norell. 2007. *Towards a practical programming language based on dependent type theory.* Ph. D. Dissertation. Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden.

Nicolas Oury and Wouter Swierstra. 2008. The Power of Pi. In *Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming* (Victoria, BC, Canada) *(ICFP '08)*. Association for Computing Machinery, New York, NY, USA, 39–50. https://doi.org/10.1145/1411204.1411213

Adam Paszke, Daniel D. Johnson, David Duvenaud, Dimitrios Vytiniotis, Alexey Radul, Matthew J. Johnson, Jonathan Ragan-Kelley, and Dougal Maclaurin. 2021. Getting to the Point: Index Sets and Parallelism-Preserving Autodiff for Pointful Array Programming. *Proc. ACM Program. Lang.* 5, ICFP, Article 88 (aug 2021), 29 pages. https://doi.org/10.1145/3473593

Gabriel Radanne. 2019. Typed parsing and unparsing for untyped regular expression engines. In *PEPM 2019 - Proceedings of the 2019 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, Co-located with POPL 2019*. https://doi.org/10.1145/3294032.3294082

Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* (2017). http://idl.cs.washington.edu/papers/vega-lite

Adam Ścibior, Ohad Kammar, and Zoubin Ghahramani. 2018. Functional programming for modular Bayesian inference. *Proc. ACM Program. Lang.* 2, ICFP (2018), 83:1–83:29. https://doi.org/10.1145/3236778