NATIONAL RESEARCH UNIVERSITY

Retrofitting Implicit Modules for 1ML

Research Proposal

Alexey Trilis, group BPM171
Scientific Advisor: Daniil Berezun, PhD

National Research University Higher School of Economics
St. Petersburg School of Physics, Mathematics, and Computer Science

Saint Petersburg, 2021

Outline

- Background Information
- Motivation
- Research Objective
- Literature Review
- Methodology
- Anticipated Results

Ad hoc polymorphism

- Functions with the same name having different semantics depending on argument types.
- Such as `+` or `print`

Approaches

- *Type classes*, like in Haskell (Wadler and Blott 1989)
- *Implicits*, like in Scala (Oliveira et al. 2010)

Background Information 2/3

ML languages

ML language family

- ML, SML, OCaml, F#, many others
- Powerful type inference
- Advanced module system
- Applications: compiler development, static program analysis, automatic theorem proving, financial systems, web development

No ad hoc polymorphism!

- Need to use `+`, `+.`, `print_int`, `print_string`, etc.
- Undesirable verbosity

Background Information 3/3

1ML

ML modules

- Two languages: core and module
- Module language is more powerful, built on dependent-type machinery (MacQueen 1986)
- Module language is verbose and difficult to integrate with core language

1ML

- Later research showed that dependent types are not actually needed (Rossberg et al. 2010)
- 1ML (Rossberg 2015) is an experimental ML dialect in which core and module languages are united

Motivation

- Extending ML with implicits is highly desirable by users
- However, it is not an easy task
- We limit ourselves to a simpler language
- However, because of some 1ML's design choices, we can achieve new results not currently possible in more mature ML languages
- Also this project can be viewed as test of expressiveness for 1ML

Research Objective

Research objective. Design implicit modules for 1ML.
Subgoals

- Reproduce all results already described for OCaml
- Improve completeness of these results by paying special attention to resolve order and unification
- Compare implementation with OCaml one

Literature Review

- Dreyer et al. 2007 — designing type classes for ML
  - Type classes require *canonicity*
  - In ML languages, canonicity is impossible to achieve
  - Without canonicity, authors impose rather severe restrictions
- White et al. 2015 — designing implicits for OCaml
  - Only prototype implementation, merging in OCaml is years away
  - Both practical and theoretical difficulties
  - Needs *higher-order unification*, which OCaml currently lacks

Methodology

- Resolving of implicits must be done simultaneously with type inference
- Delaying resolve while it is possible
- Recursive search based on a set of type constraints
- Termination check
- Retrying ambiguous resolve with new information

Anticipated Results

- Implementated implicits for 1ML, as part of 1ML prototype compiler
- We test our implementation by comparing its strength against OCaml solution
- Our solution works in all cases where OCaml solution works
- There are cases (mostly related to order and retrying) where our solution works while OCaml solution does not
- We support *implicit functors*, which are listed By OCaml solution's authors as a natural future work
- Results related to unification

References I

Dreyer, D., Harper, R., Chakravarty, M. M. T., & Keller, G. (2007). Modular type classes. In *Proceedings of the 34th annual acm sigplan-sigact symposium on principles of programming languages*, Nice, France, Association for Computing Machinery. https://doi.org/10.1145/1190216.1190229

MacQueen, D. B. (1986). Using dependent types to express modular structure. In *Proceedings of the 13th acm sigact-sigplan symposium on principles of programming languages*, St. Petersburg Beach, Florida, Association for Computing Machinery. https://doi.org/10.1145/512644.512670

References II

Oliveira, B. C., Moors, A., & Odersky, M. (2010). Type classes as objects and implicits. In *Proceedings of the acm international conference on object oriented programming systems languages and applications*, Reno/Tahoe, Nevada, USA, Association for Computing Machinery. https://doi.org/10.1145/1869459.1869489

Rossberg, A. (2015). 1ML – core and modules united (F-ing first-class modules). In *Proceedings of the 20th acm sigplan international conference on functional programming*, Vancouver, BC, Canada, Association for Computing Machinery. https://doi.org/10.1145/2784731.2784738

References III

Rossberg, A., Russo, C. V., & Dreyer, D. (2010). F-ing modules. In *Proceedings of the 5th acm sigplan workshop on types in language design and implementation*, Madrid, Spain, Association for Computing Machinery. https://doi.org/10.1145/1708016.1708028

Wadler, P., & Blott, S. (1989). How to make ad-hoc polymorphism less ad hoc. In *Proceedings of the 16th acm sigplan-sigact symposium on principles of programming languages*, Austin, Texas, USA, Association for Computing Machinery. https://doi.org/10.1145/75277.75283

References IV

White, L., Bour, F., & Yallop, J. (2015). Modular implicits. *Electronic Proceedings in Theoretical Computer Science*, *198*, 22–63. https://doi.org/10.4204/eptcs.198.2

Thank you for your attention!

Retrofitting Implicit Modules for 1ML

Research Proposal

Alexey Trilis, group BPM171
Scientific Advisor: Daniil Berezun, PhD

National Research University Higher School of Economics
St. Petersburg School of Physics, Mathematics, and Computer Science

Saint Petersburg, 2021