The Journal of Open Source Software

fortran-src: Fortran static analysis infrastructure

Mistral Contrastin [1,2], **Raoul Hidalgo Charman** [3], **Matthew Danish** [4], **Benjamin Orchard** [5], **Dominic Orchard** [5,6], **Andrew Rice** [1], **and Jason Xu** [3]

1 Department of Computer Science and Technology, University of Cambridge, UK **2** Meta, London, UK **3** Bloomberg, US **4** Utrecht University, Netherlands **5** School of Computing, University of Kent, UK **6** Institute of Computing for Climate Science, University of Cambridge, UK

DOI: 10.21105/joss.07571

Software

- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:

- @fzhao70
- @filippi
- @tclune

Submitted: 26 September 2024
Published: 07 February 2025

Summary

fortran-src is an open-source Haskell library and command-line application for the lexing, parsing, and static analysis of Fortran source code. It provides an interface to build other tools, e.g., for static analysis, automated refactoring, verification, and compilation. The library supports FORTRAN 66, FORTRAN 77, Fortran 90, Fortran 95, some legacy extensions, and partially Fortran 2003. The library has been deployed in several language tool projects in academia and industry.

Statement of need

As one of the oldest surviving programming languages (Backus, 1978), Fortran underpins a vast amount of software; Fortran is not only a mainstay of legacy software, but is also used to write new software. Fortran remains a popular language in the international scientific community; Vanderbauwhede (2022) reports data from 2016 on the UK's Archer supercomputer, showing the vast majority of use being Fortran code. Fortran is particularly notable for its prevalence in earth sciences, e.g., for implementing climate models that inform international policy decisions (Méndez et al., 2014). In 2024, Fortran re-entered the Top 10 programming languages in the TIOBE Index, showing its enduring popularity. The continued use of Fortran, particularly in scientific contexts, was the catalyst for this software package.

A challenge in writing language tools for Fortran is its long history. There have been several major language standards (FORTRAN I-IV, FORTRAN 66 and 77, Fortran 90, 95, 2003, 2008, etc.) Newer standards often deprecate features that are known to be a ready source of errors, or difficult to specify or understand. However, compilers often support an amalgam of features across standards (Urma et al., 2014), enabling developers to keep using deprecated features and mix language standards. This complicates the development of new tools for manipulating Fortran source code; one must tame the weight of decades of language evolution.

Our package, fortran-src, provides an open-source unified core for statically analysing Fortran code across language standards, with a focus on legacy code over cutting-edge modern Fortran. It is both a standalone tool and a library, providing a suite of standard static analyses as a basis for further programming language tools and systems.

Related software

A variety of other tools exist for analysing Fortran, but most are commercial and closed source, e.g., plusFORT[1] (which includes the SPAG refactoring tool), the SimCon fpt tool[2]

[1] https://polyhedron.com/?product=plusfort
[2] http://simconglobal.com/fpt_summary.html

(which includes further verification features like dimensional analysis), and Forcheck[3]. General commercial static analysis tools, like Coverity[4] and Understand[5], can also handle Fortran. Photran[6] is an open-source plugin for refactoring in Eclipse, but does not provide more general static analysis facilities. More recent work has developed open-source tools for refactoring Fortran (Vanderbauwhede, 2022): RefactorF4Acc[7] is an open-source tool for upgrading FORTRAN 77 code to Fortran 95.

No comprehensive lexing, parsing, and analysis library was available from which to build new tools.

Functionality in brief

- Lexing (of both fixed and free form code) and parsing of Fortran to an expressive unified Abstract Syntax Tree;
- Static analyses, e.g., general data flow analysis including:
  - Reaching definitions;
  - Def-use/use-def;
  - Constant evaluation;
  - Constant propagation;
  - Live variable analysis;
  - Induction variable analysis.
- Type checking;
- Module graph analysis;
- Pretty printing;
- "Reprinting" (patching sections of source code without removing secondary notation such as comments);
- Exporting to JSON.

fortran-src is primarily a Haskell library but it also packages a command-line tool. By exporting parsed code to JSON, the parsing and analyses that fortran-src provides may be utilized by non-Haskell tools.

Functionality and example usage of the tool and library is described in detail on the fortran-src wiki. A demonstration of fortran-src for static analysis is provided by a small demo tool which detects if an allocatable array is used before it has been allocated.[8]

Work building on fortran-src

CamFort

The fortran-src package originated in the CamFort project[9] whose aim was to (1) develop practical tools for scientists to help reduce the accidental complexity of models through evolving a code base, and (2) provide tools for automatically verifying properties of code. The work resulted in the CamFort tool, of which fortran-src is the core infrastructure.

CamFort provides automatic refactoring of deprecated or error-prone programming patterns, with the goal of helping to meet core quality requirements, such as maintainability (D. Orchard

[3] https://codework.com/solutions/developer-tools/forcheck-fortran-analysis/
[4] https://www.synopsys.com/software-integrity/static-analysis-tools-sast/coverity.html
[5] https://scitools.com/
[6] https://projects.eclipse.org/projects/tools.ptp.photran
[7] https://github.com/wimvanderbauwhede/RefactorF4Acc
[8] https://github.com/camfort/allocate-analysis-example
[9] Funded from 2015-18 by the EPSRC under the project title *CamFort: Automated evolution and verification of computational science models* https://gow.epsrc.ukri.org/NGBOViewGrant.aspx?GrantRef=EP/M026124/1

& Rice, 2013). It can rewrite EQUIVALENCE and COMMON blocks (both of which were deprecated in the Fortran 90 standard) into more modern style.

CamFort also provides code analysis and lightweight verification tools (Contrastin et al., 2016). Source-code annotations (comments) provide specifications of certain aspects of a program's meaning or behaviour. CamFort can check that code conforms to these specifications (and for some features can suggest places to insert specifications or infer specifications from existing code). Facilities include: units-of-measure typing (Danish et al., 2024; D. Orchard et al., 2020; D. A. Orchard et al., 2015), array access patterns (for capturing the shape of stencil computations) (D. A. Orchard et al., 2017), deductive reasoning via pre- and post-conditions in Hoare logic style, and various code safety checks.

CamFort has been previously deployed at the Met Office, with its analysis tooling run on the Unified Model (Walters et al., 2017) to ensure internal code quality standards are met.

fortran-vars memory model library

`fortran-vars` is a static analysis library built on top of `fortran-src`. Many static analysis questions depend on knowing the value and type of expressions. `fortran-vars` provides an API to answer this question. It has modules for symbol table construction, constant expression evaluation, and type checking. Additionally, `fortran-vars` provides a memory model to resolve aliases introduced by `equivalence` statements, which are very common in legacy Fortran 77 code. It is possible to construct such a memory model because variables in Fortran 77 are statically allocated by default. Data flow analysis, such as constant propagation analysis, can be conducted based on memory locations instead of variable names.

Nonstandard INTEGER refactoring

fortran-src has been used to build refactoring tools to help migration and improve the quality of large legacy codebases. One example is an effort to fix issues around the use of integers where logical types are expected. This tool uses the typechecker to find integer expressions which are then normalised while flagging anything potentially changing behaviour for further manual inspection. These might be situations in which some code is hard to statically analyse but safe, or it may have uncovered an existing bug. The tool uncovered many such bugs in a particular codebase during this effort, including several in the form of the snippet above.

This effort, along with a number of others, allowed the team working at Bloomberg (a subset of the authors here) to eventually migrate a codebase from a legacy compiler to a modified GFortran, with no change in behaviour.

Project maintenance and documentation

fortran-src may be built and used on Windows, Mac, and Linux systems using a recent version of the Glasgow Haskell Compiler. The project includes an expansive test suite covering various parsing edge cases and behaviours, which is automatically executed for changes to the project (on the above three systems). Bug reports and other contributions are welcomed at the fortran-src GitHub page.

Acknowledgements

The initial work on the fortran-src infrastructure was funded by an EPSRC grant **CamFort: Automated evolution and verification of computational science models** (EP/M026124/1), from 2015-18, and by an EPSRC Impact Acceleration Award and then Knowledge Transfer Partnership grant from 2018-19. Orchard is also supported by the generosity of Eric and Wendy Schmidt by recommendation of the Schmidt Sciences program, through which he carries on his work supporting scientists through programming languages, tools, and systems as part of the Institute of Computing for Climate Science at the University of Cambridge. Furthermore, this work was supported by a grant from Bloomberg.

A number of other people have been associated with the project and have contributed to the development of the package over the years (in alphabetical order of surname):

- Daniel Beer
- Anthony Burzillo
- Harry Clarke
- Aiden Jeffrey
- Lukasz Kolodziejczyk
- Vilem-Benjamin Liepelt
- Darius Makovsky
- Benjamin Moon
- Daniel Ruoso
- Eric Seidel
- Poppy Singleton-Hoare
- Jay Torry

References

Backus, J. (1978). The history of FORTRAN I, II, and III. *SIGPLAN Not.*, *13*(8), 165–180. https://doi.org/10.1145/960118.808380

Contrastin, M., Danish, M., Orchard, D., & Rice, A. (2016). Lightning talk: Supporting software sustainability with lightweight specifications. *Proceedings of the Fourth Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE4), University of Manchester, Manchester, UK, September 12-14*, *1686*.

Danish, M., Orchard, D., & Rice, A. (2024). Incremental units-of-measure verification. In *CoRR* (Vol. abs/2406.02174). https://doi.org/10.48550/ARXIV.2406.02174

Méndez, M., Tinetti, F. G., & Overbey, J. L. (2014). Climate models: Challenges for Fortran development tools. *2014 Second International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*, 6–12. https://doi.org/10.1109/SE-HPCCSE.2014.7

Orchard, D. A., Contrastin, M., Danish, M., & Rice, A. C. (2017). Verifying spatial properties of array computations. *Proc. ACM Program. Lang.*, *1*(OOPSLA), 75:1–75:30. https://doi.org/10.1145/3133899

Orchard, D. A., Rice, A. C., & Oshmyan, O. (2015). Evolving Fortran types with inferred units-of-measure. *J. Comput. Science*, *9*, 156–162. https://doi.org/10.1016/j.jocs.2015.04.018

Orchard, D., Contrastin, M., Danish, M., & Rice, A. (2020). Guiding user annotations for units-of-measure verification. *CoRR*, *abs/2011.06094*. https://arxiv.org/abs/2011.06094

Orchard, D., & Rice, A. (2013). Upgrading Fortran source code using automatic refactoring. *Proceedings of the 2013 ACM Workshop on Refactoring Tools, WRT@SPLASH 2013, Indianapolis, IN, USA, October 27, 2013*, 29–32. https://doi.org/10.1145/2541348.2541356

Urma, R.-G., Orchard, D., & Mycroft, A. (2014). Programming language evolution workshop report. *Proceedings of the 1st Workshop on Programming Language Evolution*, 1–3. https://doi.org/10.1145/2717124.2717125

Vanderbauwhede, W. (2022). Making legacy Fortran code type safe through automated program transformation. *The Journal of Supercomputing*, *78*(2), 2988–3028. https://doi.org/10.1007/S11227-021-03839-9

Walters, D., Boutle, I., Brooks, M., Melvin, T., Stratton, R., Vosper, S., Wells, H., Williams, K., Wood, N., Allen, T., & others. (2017). The Met Office unified model global atmosphere 6.0/6.1 and JULES global land 6.0/6.1 configurations. *Geoscientific Model Development*, *10*(4), 1487–1520. https://doi.org/10.5194/gmd-10-1487-2017