# Flow Analysis as a Higher-order Constraint Problem

Luke Ong Steven Ramsay

January 2018

A central problem in program analysis [5] is to predict, statically, which values may occupy a variable at runtime. In a functional programming language the task, known as *flow analysis*, is particularly difficult since the control flow of the program cannot be accurately determined from the program syntax due to the prescence of higher-order functions. Nevertheless, the problem is such an important and natural one that it has been well studied, with most approaches being equivalent to, or some variation on, Shivers' seminal analysis 0-CFA. This analysis is particularly appealing because it has a constraint-based formulation: the analysis generates a set of constraints (in this case subset inclusions) which capture the essence of the problem and whose satisfaction yields a solution to the original problem.

However, until recently, most approaches to the problem involve approximating higher-order control flow using first-order apparatus (e.g. tree grammars, set constraints or abstract machines). Some attempts have been made to do better using new technology (e.g. pushdown automata, indexed grammars, higher-order model checking [8]) but there is currently no crisp, *constraint-based* formulation of a truely higher-order flow analysis.

This project aims to develop a flow analysis for functional programs (or an applied lambda calculus) formulated as a higher-order constraint problem (the unknowns that are constrained are of higher-type). One starting point is the work on exact flow analysis using higher-order model checking, since we know that higher-order model checking [3, 7, 6] can itself be recast, in a natural way, as a higher-order constraint problem. (See [2] for a higher-order Horn constraint-based approach to the safety verificaiton of higher-order programs; and [4, 1] for first-order examples.) Whatever approach is taken, the analysis must be described precisely, shown to be sound (the results of an analysis do not omit any of the possible values that a variable can take at runtime) and there should be a prototype implementation. Beyond this, the work lends itself well to two possible extensions, depending on the time and inclination of the student:

1. Extending the flow analysis to a real language, such as Haskell, via the Core language of GHC or
2. Characterising classes of constraint and classes of program for which the analysis enjoys a notion of relative completeness.

## Relevant Courses

- 2nd and 3rd-year: Compilers, and Lambda Calculus and Types.

- 4th-year: Computer-aided Formal Verification; Automata, Logic and Games; and Program Analysis.

# References


[1] N. Bjørner, A. Gurfinkel, K. L. McMillan, and A. Rybalchenko. Horn clause solvers for program verification. In *Fields of Logic and Computation II - Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday*, pages 24–51, 2015.

[2] T. C. Burn, C. L. Ong, and S. J. Ramsay. Higher-order constrained horn clauses for verification. *PACMPL*, 2(POPL):11:1–11:28, 2018.

[3] N. Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3):20, 2013.

[4] K. L. McMillan. Lazy annotation revisited. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, pages 243–259, 2014.

[5] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of program analysis*. Springer, 1999.

[6] C.-H. L. Ong. Higher-order model checking: An overview. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 1–15, 2015.

[7] S. J. Ramsay, R. P. Neatherway, and C.-H. L. Ong. A type-directed abstraction refinement approach to higher-order model checking. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 61–72, 2014.

[8] Y. Tobita, T. Tsukada, and N. Kobayashi. Exact flow analysis by higher-order model checking. In *Functional and Logic Programming - 11th International Symposium, FLOPS 2012, Kobe, Japan, May 23-25, 2012. Proceedings*, pages 275–289, 2012.