Higher-order Logic, Types, and Machine Learning:

Several Projects

Supervisor: Luke Ong

The general theme is to use higher-order logic, (refinement) types, allied formal methods (notably SMT solving and model checking), and machine learning for the automatic verification, specification and synthesis of programs.

Several years ago (circa 2012), Bjørner, McMillan and Rybalchenko advocated an approach to automatic program verification based on constrained Horn clauses (CHC). Our recent POPL paper[1] (Burn et al., 2018) extends the CHC approach to higher orders.

We formalise a class of constrained systems, called *higher-order constrained Horn clauses* (HoCHC), and introduce two decision problems relevant to program verification, namely, HoCHC satisfiability, and HoCHC safety problems. Not surprisingly, these problems are not decidable in general. In (Burn et al., 2018), we developed a sound approach to solve the HoCHC safety problem, based on a refinement type system:

(1) *Type Soundness*: given an instance of the HoCHC safety problem, if the associated "logic program" of the problem instance is typable, then it is a yes-instance.

(2) *Reduction*. We show that typability reduces to solvability of a system of 1st-order constrained Horn clauses.

We have developed a prototype "proof of concept" tool implementation of this method, called Horus: see `http://mjolnir.cs.ox.ac.uk/horus`

Members of my research group are now working on other approaches to solving HoCHC:

(i) SLD resolution of higher-order logic programs

(ii) Reynold's defunctionalization.

In the following I briefly describe a number of possible projects.

Project 1. Relativised completeness of refinement types for solving HoCHC

The refinement type approach (i.e. (1) and (2) above) is not complete: the converse of type soundness (1) does not hold. We conjecture that completeness holds for a class of HoCHC satisfying a syntactic constraint called *safety* (Knapik et al., 2002).

[1]See also the slides `http://www.cs.ox.ac.uk/people/luke.ong/personal/IRIF-7Dec17.pdf`, and a video recording of the POPL18 presentation `https://dl.acm.org/citation.cfm?id=3158099&picked=formats`.

We think that there are connections with the notion of relativised completeness studied by Unno et al. (2013), though our focus on safety is new.

This project would suit someone who is interested in (refinement) types, lambda calculus and functional programming.

Project 2. Relatively-complete Hoare logic for higher-order programs

Blass and Gurevich (1987, 2000) characterised *existential fixed point logic* as the underlying logic of Hoare axiom systems. Their result can be read, dually, as a statement of the (relative) completeness of first-order constrained Horn clauses for proving the partial correctness of first-order programs. We conjecture that this theorem about first-order constraints can be lifted to state the relative completeness of HoCHCs for proving partial correctness assertions of higher-order programs.

Previous approaches to the problem of constructing relatively complete systems for higher-order programs have either relied on first-order Gödel encodings (Unno et al., 2013), higher-order expressiveness assumptions (Olderog, 1983), or bespoke higher-order logics (cf. evaluation formulas of Honda et al. (2006)), none of which seems amenable to automation.

By contrast, we have shown (Burn et al., 2018) that HoCHCs are well suited to automation, and can even leverage the existing work in higher-order (and first-order) verification. Our approach will be to design a new Hoare logic for higher-order programs (with and without state) in which the problem of provability is readily seen to reduce to the HoCHC problem.

This project would suit someone who has some background in logic, theory of verification (Hoare logic), and denotational semantics (e.g. Full abstraction problem of PCF (Plotkin, 1977)).

Project 3. Higher-order logic, machine learning, and program synthesis

Automatic program synthesis is an old problem which is still central to programming. There have been interesting developments recently using a variety of approaches:

1. refinement types e.g. (Polikarpova et al., 2016)
2. Angluin-style automata learning e.g. (Vaandrager, 2017)
3. syntax-guided and search / model checking based e.g. (et al., 2015)

A direction which I am interested in developing is synthesis of (higher-order) functional programs by learning from examples. A highly relevant work is an approach based on *higher-order inductive logic programming*, due to Cropper and Muggleton (2016). Their method involves an alternation of higher-order procedures called Abstraction and Invention. HoCHC are syntactically essentially the same as some versions of higher-order logic programs in the literature: many key observations in higher-order inductive logic programing are just as meaningful when transposed to the HoCHC setting. The idea is that Abstractions can benefit from recent advances in the use of refinement types in automatic program specification. Moreover, Inventions, which are concerned with the construction of definitions for the predicate variables used in the Abstractions, can be augmented using syntax-guided model checking.

Explainable AI or *meta-interpretive AI* is a grand challenge. Progress will rely on marrying data-centric AI (statistical machine learning, dominating AI in the last two decade) with symbolic AI (building on symbolic logic, the basis of much of AI up to the 90s, formal methods and programming languages, and other branches of CS). A related challenge in AI is the development of *automatic programming systems* ("writing codes that write codes"). This project may be viewed as a (small) first step in these directions. Comparatively more speculative, this project is also suitable as a DPhil project.

References

Andreas Blass and Yuri Gurevich. Existential fixed-point logic. In *Computation Theory and Logic, In Memory of Dieter Rödding*, pages 20–36, 1987. doi: 10.1007/3-540-18170-9_151. URL https://doi.org/10.1007/3-540-18170-9_151.

Andreas Blass and Yuri Gurevich. The underlying logic of hoare logic. *Bulletin of the EATCS*, 70:82–111, 2000.

Toby Cathcart Burn, C.-H. Luke Ong, and Steven J. Ramsay. Higher-order constrained horn clauses for verification. *PACMPL*, 2(POPL):11:1–11:28, 2018. doi: 10.1145/3158099. URL http://doi.acm.org/10.1145/3158099.

Andrew Cropper and Stephen H. Muggleton. Learning higher-order logic programs through abstraction and invention. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 1418–1424, 2016. URL http://www.ijcai.org/Abstract/16/204.

Rajeev Alur et al. Syntax-guided synthesis. In *Dependable Software Systems Engineering*, pages 1–25. 2015. doi: 10.3233/978-1-61499-495-4-1. URL https://doi.org/10.3233/978-1-61499-495-4-1.

Kohei Honda, Martin Berger, and Nobuko Yoshida. Descriptive and relative completeness of logics for higher-order functions. In *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, pages 360–371, 2006. doi: 10.1007/11787006_31. URL https://doi.org/10.1007/11787006_31.

Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, pages 205–222, 2002. doi: 10.1007/3-540-45931-6_15. URL https://doi.org/10.1007/3-540-45931-6_15.

Ernst-Rüdiger Olderog. A characterization of hoare's logic for programs with pascal-like procedures. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 320–329, 1983. doi: 10.1145/800061.808761. URL http://doi.acm.org/10.1145/800061.808761.

Gordon D. Plotkin. LCF considered as a programming language. *Theor. Comput. Sci.*, 5(3):223–255, 1977. doi: 10.1016/0304-3975(77)90044-5. URL https://doi.org/10.1016/0304-3975(77)90044-5.

Nadia Polikarpova, Ivan Kuraj, and Armando Solar-Lezama. Program synthesis from polymorphic refinement types. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*, pages 522–538, 2016. doi: 10.1145/2908080.2908093. URL http://doi.acm.org/10.1145/2908080.2908093.

Hiroshi Unno, Tachio Terauchi, and Naoki Kobayashi. Automating relatively complete verification of higher-order functional programs. In *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, pages 75–86, 2013. doi: 10.1145/2429069.2429081. URL http://doi.acm.org/10.1145/2429069.2429081.

Frits W. Vaandrager. Model learning. *Commun. ACM*, 60(2):86–95, 2017. doi: 10.1145/2967606. URL http://doi.acm.org/10.1145/2967606.