Proof-relevant structural resolution for type-inference

František Farka <ff32@st-andrews.ac.uk> Supervisor: Kevin Hammond
School of Computer Science, University of St Andrews

Motivation

The importance of formal verification and modern verification tools in software development is widely acknowledged. There are two major approaches to verification: in the first approach verification problems are specified in an automated prover, e.g. SMT solver or logic programming system. An alternative is software verification driven by types, the interesting properties are recorded in types of functions in the program. This approach is more trustworthy as the function, thanks to Curry-Howard isomorphism, represents a checkable proof of the property encoded by a type whereas there is no such proof in the automated-prover approach. However, the automated-prover approach seems to be more readily-integrabel to existing languages as is demonstrated by *e.g.* Liquid Haskell and F*.

We propose to use a recent extension to standard resolution in logic programming—a proof-relevant structural resolution—to replace an SMT solver that is used by a static verification tool Liquid Haskell [3, 4] with a proof-relevant logic programming. Not only we obtain a system that produces machine-checkable proofs but due to properties of structural resolution we can also easily reason about inductive and coinductive properties of programs adapting techniques of coinductive logic programming (CoALP) [2].

Proof-relevant structural resolution

Structural resolution is a newly proposed alternative to SLD-resolution in first-order Horn-clause logic. Structural resolution allows for not only traditional inductive proof search but also for coinductive proof search. It can be extended such that, besides a resulting substitution, it also captures a proof object. The signature of a program is extended with a set of function symbols Θ and there is one symbol assigned to each clause in a program. The big-step semantics accounts for these symbols:

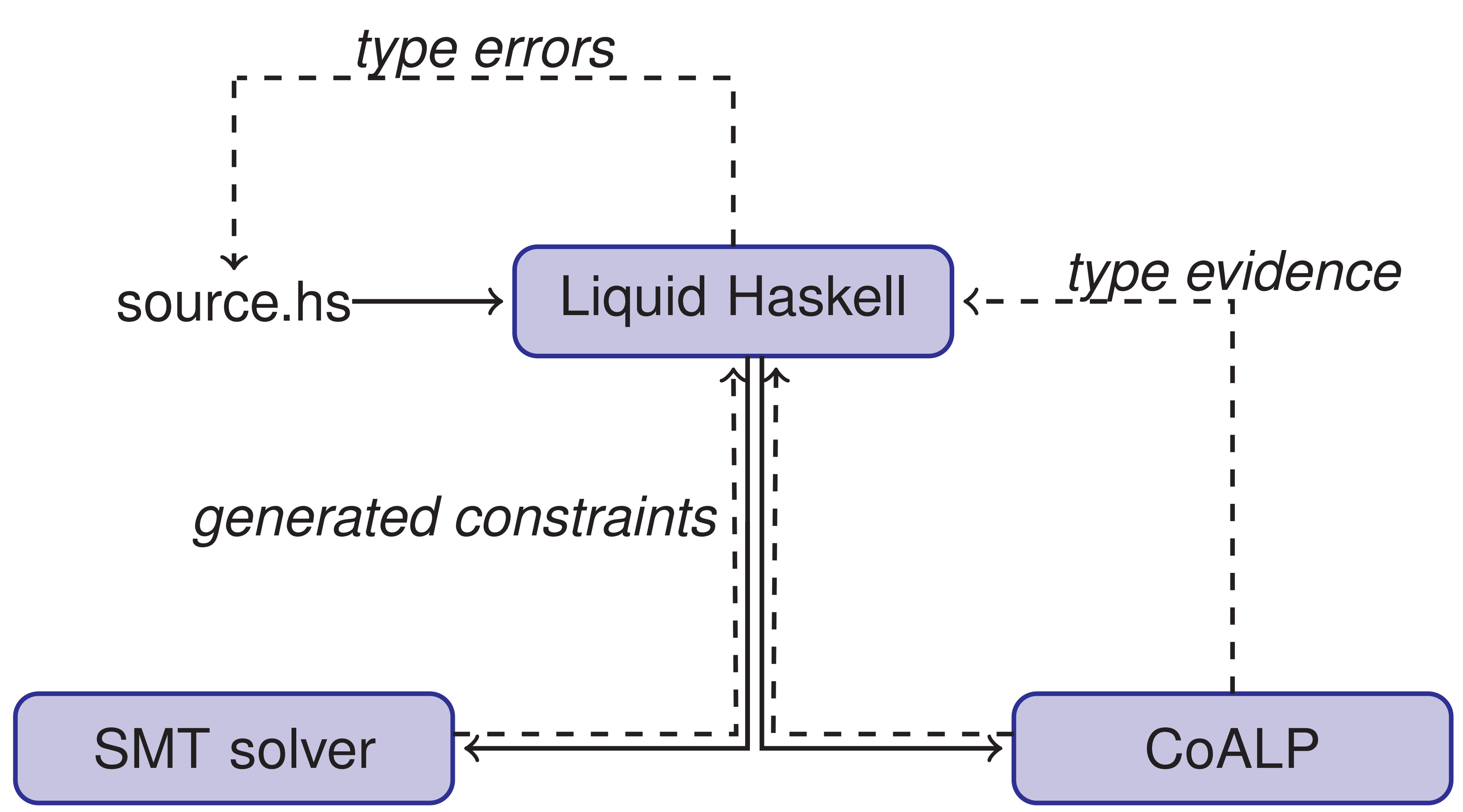
$$\frac{P \vDash x_1 : \sigma B_1, \dots P \vDash x_n : \sigma B_n}{P \vDash s(x_1, \dots, x_n) : \sigma A}$$

where P is a logic program such that $s \in \Theta$ and $s : A \leftarrow B_1, \dots, B_n \in P$. For proof objects (terms) $x_1, \dots, x_n$ the term $s(x_1, \dots, x_n)$ proves σA in P. This explicit handling of proof also allows for better handling of coinductive hypothesis.

Liquid Haskell

Liquid Haskell is a static verifier for the Haskell programming language based on liquid types [3]. The current implementation takes an annotated Haskell program and generates a set of constraints that is solved by an SMT solver. This approach suffers from the above described problem—the SMT solver does not provide any checkable proof that the property that is being verified indeed holds. We propose to replace an SMT solver step with the proof-relevant structural resolution. This will bring the following improvements over the current approach:

- The verification tools will provide a machine checkable proof of properties that are verified,
- CoALP resolution is done in parallel and promises improvement in performance, and
- the generated evidence can be further used for type-inference when integrated with a compiler, *e.g.* for construction of type class dictionaries[1].



References

[1] Peng Fu et al. 'Proof Relevant Corecursive Resolution'. In: *Functional and Logic Programming - 13th International Symposium, FLOPS 2016, Kochi, Japan, March 4-6, 2016, Proceedings*. Ed. by Oleg Kiselyov et al. Vol. 9613. Lecture Notes in Computer Science. Springer, 2016, pp. 126–143.

[2] Ekaterina Komendantskaya et al. 'Structural Resolution: a Framework for Coinductive Proof Search and Proof Construction in Horn Clause Logic'. In: *ACM Transcations in Computational Logic* submitted (2015).

[3] Patrick Maxim Rondon et al. 'Liquid types'. In: *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation, Tucson, AZ, USA, June 7-13, 2008*. Ed. by Rajiv Gupta et al. ACM, 2008, pp. 159–169.

[4] Niki Vazou et al. 'Abstract Refinement Types'. In: *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*. Ed. by Matthias Felleisen et al. Vol. 7792. Lecture Notes in Computer Science. Springer, 2013, pp. 209–228.