Practical Program Analysis and Synthesis

Eran Yahav
The Technion, Haifa, Israel

Software is becoming increasingly complex. For *system-level programmers*, the transition of hardware providers to multi-core architectures exposes new sources of complexity. Additional complexity is introduced by systems using heterogenous concurrency and massively data-parallel architectures such as GPUs. For *application-level programmers*, the proliferation of libraries and frameworks, intended to reduce complexity, often requires programmers to be aware of intricate library internals for effective and correct usage of the library. Furthermore, despite the ability to hide some aspects of concurrency in the library, even application- level programmers might still need to reason about atomicity (e.g.,[8]).

Despite significant progress in automatic checking and verification tools (e.g., [2, 1]), such tools can only be applied after the code is written and may be broken in a fundamental manner. This motivates us to explore *software synthesis* techniques that assist a programmer *during* the development process.

In this lecture series, we will focus on synthesis techniques that use *abstract interpretation* [3]. Abstract interpretation provides a framework for sound static (compile-time) reasoning about realistic programs by over-approximating their runtime behaviors. We will first survey basic ideas from abstract interpretation in the context of practical program analysis, and then employ these ideas in program synthesis.

We will present a framework for synthesizing efficient synchronization in concurrent programs, a task known to be difficult and error-prone when done manually. The framework is based on abstract interpretation and can infer synchronization for infinite state programs. Given a program, a specification, and an abstraction, we infer synchronization that avoids all (abstract) interleavings that may violate the specification, but permits as many valid interleavings as possible. We will show application of this general idea for automatic inference of atomic sections [9] and memory fences [4-6] in programs running over relaxed memory models. Then, we will present a framework for synthesis of code using libraries, based on combination of results from semantic code search [7]. Finally, we will discuss some interesting directions for future research.

References

[1] T.Ball, R.Majumdar, T.Millstein, S.K.Rajamani. *Automatic Predicate Abstraction of C Programs.* In: Procs. of the ACM SIGPLAN Conference PLDI '01, pp. 203–213, ACM, 2001.

[2] B.Blanchet, P.Cousot, R.Cousot, J. Feret, L. Mauborgne, A. Mine, D. Monniaux, X.Rival. *A Static Analyzer for Large Safety-critical Software.* In: Procs. of the ACM SIGPLAN Conference PLDI'03, pp. 196–207, ACM, 2003.

[3] P.Cousot, R.Cousot. *Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints.* In: Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 238–252, ACM, 1977.

[4] M.Kuperstein, M.Vechev, E.Yahav. *Automatic Fence Inference.* In: Procs of the FMCAD'10, pp. 111–119, 2010.

[5] M.Kuperstein, M.Vechev, E.Yahav. *Partial-coherence Abstractions for Relaxed Memory Models.* In: Procs. of the PLDI'11. pp. 187–198, 2011.

[6] F.Liu, N.Nevev, N.Prisadnikov, M.Vechev, E.Yahav. *Dynamic Synthesis for Relaxed Memory Models.* In: Procs. of the PLDI'12, pp. 429–440, 2012.

[7] A.Mishne, S.Shoham, E.Yahav. *Typestate-based Semantic Code Search over Partial Programs.* In: Procs. of the OOPSLA'12, pp. 997–1016, 2012.

[8] O.Shacham, N.Bronson, A.Aiken, M.Sagiv, M.Vechev, E.Yahav. *Testing Atomicity of Composed Concurrent Operations.* In: Procs. of the OOPSLA '11, pp. 51–64, 2011.

[9] M.Vechev, E.Yahav, G.Yorsh. *Abstraction-guided Synthesis of Synchronization.* In: Procs. of the POPL '10, pp. 327–338, 2010.