Local Temporal Reasoning

Submitted by brad_martin on Tue, 04/22/2014 - 11:58am. Contributors:
Eric Koskinen Tachio Terauchi

Abstract:
Programming languages that use higher-order functionality (e.g. Java, C#, F#, Haskell, Ocaml, Perl, Python, Ruby) have become commonplace. Higher-order language features such as map, grep, Google's Map/Reduce, are used widely and applauded for their simplicity and modularity. Meanwhile, in the past couple of decades, algorithms and tools have emerged that have enabled automatic verification of some industrial software systems. Symbolic analysis techniques such as abstraction refinement and interpolation have given rise to interprocedural program analysis tools for safety such as SLAM and Blast, while termination argument refinement has lead to tools for liveness such as Terminator. Some of our recent work has led to tools and algorithms for verifying properties expressed in temporal logic: more elaborate specifications that combine safety and liveness. These verification techniques have been mostly limited to imperative first-order software and cannot be applied to higher-order programming languages. In recent years, researchers have developed some techniques for verifying higher-order programs. Some showed how to verifying temporal properties of higher-order programs that are restricted to finite data. Others have also developed methods of verifying purely properties or purely termination of infinite-data higher-order programs. Despite the efforts discussed above, at present there are no methods for verifying safety/liveness properties (i.e. temporal logic formulae) of programs written in higher-order languages.

Compositional reasoning. We present the first technique for verifying temporal logic properties of higher-order, infinite-data programs. The crux of our work is to decompose the problem, not only by dividing the program up into individual expressions via a type-and-effect system, but also, for every expression, to track the behavior of finite traces separate from the behavior of infinite traces. Our type rules permit verification oracles (including the type system itself) to reason about the conditional safety and liveness (i.e. temporal) behavior of program parts, and compose these facts together to prove the overall target property of the program. Moreover, we show that existing tools can be used as oracles to introduce liveness proofs into the type system's effects. While it is a commonly held belief that type systems cannot be used for liveness properties, we show how they can, nonetheless, be used to carry some liveness information and soundly combine reasoning about program parts together to prove overall safety and liveness for a wide variety of examples.

Contributions. To our knowledge, this work marks the first method for reasoning about temporal logic properties for higher-order programs that have infinite data. We believe our work provides the theoretical foundation toward several areas of practical significance. To this end, we have devised general rules so there are many instantiations and applications of them, including: (1) Instantiation to a wide variety of specification logics: able to support any logic that is closed under intersection, union, and composition (over finite and infinite traces) such as Bu chi specifications. (2) Instantiation to arbitrary type environments. Often the type system alone is strong enough to derive safety properties. For example, when using refinement types in the absence of a termination oracle, our rules can be thought of as a novel extension to dependent types, where temporal behaviors are carried as effects. (3) Instantiation of oracles to any fragment of program expressions or any subset of the specification logic. (4) Instantiation to a modular reasoning system for temporal behaviors of first-order programs. We have devised our methodology to be based on local reasoning, employing a type system. This stands in contrast to many existing verification works for higher-order programs that operate by extracting a transition system and then performing standard model checking techniques. Such existing techniques suffer from the inability to refine the abstraction during the verification process and, moreover, require input programs to be given in CPS form (or converted thereto), a further barrier to adoption in industry.
Presenter Bio:
I am a research scientist at the Courant Institute of Mathematical Sciences at New York University. Previously I was a Ph.D student at the University of Cambridge (UK) Computer Laboratory, jointly supervised by Byron Cook and Mike Gordon. I am funded by the Gates Cambridge Scholarship. I received my Sc.M from Brown University, where I worked with Maurice Herlihy. In my research, I develop new mathematical methods for making software safer and more efficient, and apply those methods to realistic computer systems. In recent years I have accomplished this in two ways. First, I have improved programming languages by introducing new language features which are, by design both safer and more efficient. In particular, I am concerned with language advances which enable engineers to safely produce software which consists of parallel computation (e.g. my work on Transactional Boosting and Coarse-Grained Transactions). Second, I have developed static and dynamic analysis techniques in order to better understand the performance (e.g. discovering symbolic complexity bounds), understand the behavior (e.g. request tracing in BorderPatrol), discover bugs and formally prove correctness of programs (e.g. proving Linear Temporal Logic properties via a reduction to a program analysis task and symbolic partial determinization).

Other available formats:

Local Temporal Reasoning
Switch to normal viewerSwitch to experimental viewer

Eric Koskinen | Tachio Terauchi
License: Creative Commons 2.5

Nagoya University New York University National HCSS Conference 2014 Academia Presentation HCSS14 Specifications and Requirements [HCSST14]