Meta Logical Frameworks and QPQ Position Paper

Carsten Schürmann

Yale University
New Haven, CT, USA
carsten@cs.yale.edu

Deductive software as opposed to numerical software is gaining more and more importance in industrial applications. Numerical software can be found in components prevalent in control systems that are built into cars, trains, space vehicles, and plants, recognition systems for voice, vision, and access control, but also our internet infrastructure such as load balancing servers, and routers. Regarding fault tolerance, safety, security, and correctness, many branches of industry are employing or are thinking of employing deductive software such as model checkers, theorem provers, type checkers, and counter example generators to improve designs cost-efficiently before releasing a product.

Complementary to the functionality commonly associated with deductive software is the question of how to represent derivations, proofs, traces of model checkers, counter examples, and other non-numerical domains if at all. Due to efficiency reasons, many implementations of theorem provers refrain from the explicit creation and maintenance of deductions, making it more difficult, almost impossible to connect and import deductions from one component to another, and consequently render independent third party verification of deductions impossible. For others that do, deductions are merely mathematical objects that can be manipulated, constructed, deconstructed, combined, exchanged, inspected and satisfy far reaching mathematical properties and laws.

The structure of deductions is in general richer then that of numbers. Questions about how to represent axioms, inference rules, hypotheses, conclusions, formulas, terms, quantifiers, and judgments, are often a matter of personal choice and preference, leading to a significant amount of syntactical and semantical ambiguity and prohibiting the free exchange and sharing of deductive objects among software components, or entire software systems. In fact once committed to a particular representation, interfacing a deductive software system to another component can be prohibitively difficult, and the associated cost is often thought to outweigh the expected benefits.

The deductions prevalent in deductive software components range from derivations in various logics, such as, temporal, first-order, or modal logics, to witness traces of model checkers. Ultimately, deductions should have adequate representations in software and should not only be described by the properties they satisfy. Of course, all of this complexity is mirrored in the deductions as well as the deductive software components whose implementations are tedious and often difficult to get right. Model checkers, for example, exploit symmetry properties to prune the state space, and theorem provers employ various optimization tech-

niques to make proof search feasible. The soundness of an implementation can be difficult to show, especially in the model checking case where it hinges on a complete traversal of the state space.

A source code repository such as QPQ in form of a passive digital library will clearly be of interest to many contemporary industrial applications. However, if it were also active in that it offers source code related functionality, as for example, operators that combine a heterogeneous set of software components into one single component that share one common data structure by means of partial evaluators and semantics preserving source code transformers, we can hope for a tighter integration of components, efficiency benefits, portability, and simplified maintainability. Combining deductive software components that are written in different programming languages but share the same core data structures can be a real challenge and requires semantic modeling of each programming language, with sound transformations into and from the semantic domain.

The interaction of heterogeneous software components adds another level of complexity to the repository of deductive software. While many safety properties can be inferred directly from a software component (given an appropriate semantic model of the programming language involved), there are others, often relevant for security, privacy, and secrecy, that can only be inferred from communication infrastructure or the protocols that are used to communicate between software component. CORBA or COM and DCOM, for example, are popular software architectures used in practice that interconnect remote applications and facilitate the exchange of data. Their meta-theoretic properties, however, remain informal and can in general not be used when reasoning about the behavior of a software system.

We therefore conjecture that a source code repository such as QPQ should distinguish between the correctness of data representations, the correctness of the functionality of a software component, and the correctness of the the communication infrastructure. To tackle these challenges, we advocate the use of meta-logical frameworks that provide

1. A rich, uniform, expressive, representation language to represent deductions. Logical frameworks, in particular LF [HHP93], are designed as a meta-language for representing logics, and therefore lend themselves as prime candidates for encoding deductions. More research is necessary to scale the present techniques to capture the semantics of real world programming languages, including Java, C#, ML, Haskell or Mathematica, and model the underlying communication infrastructure.
2. Tools to help analyze deductions, deductive software and their properties. Special-purpose reasoning tools that are custom-made for the underlying logical framework ensure that the source code of deductive software entered into the QPQ satisfies the accompanying meta theoretical claims. One example is the special-purpose theorem prover implemented in Twelf [Sch00] for LF.
3. Tools that can convert between different deductions, mapping them from one logical formalism into another. Different software components employ

different deductive systems and often one can be converted into another, for example, by subsumption or because two logics have different but equivalent formulations. One could also speculate that ultimately these conversion tools are useful for faithfully converting source programs from one programming language into another.

Even without automated reasoning techniques, a digital library based on QPQ must make source code available at a "suitable" level of abstraction. Programming language idiosyncrasies, global state, and other programming language related particularities make it often difficult to distinguish the essential from the unimportant, and consequently leave informal human software verification time consuming, tricky, and error prone. Consider, for example, the LINUX approach to open source. Although the source code can be inspected and analyzed by everybody, buffer overflow vulnerabilities in servers and exploits of faulty protocols persist to be hard problems. On the other hand, if its source code could be made accessible on higher levels of abstraction, automatic inspection and certification processes may arguably have better chances to success.

And finally, a digital library based on QPQ will have the most significant impact, if it also offers configuration management capabilities. Updates of deductive software should progress as autonomously as possible, while preserving the prescribed properties by the client side. This task includes again reasoning about deductive software and their properties. A sophisticated and reliable configuration management system seems to play a key role in the deployment and acceptance of such a repository.

In conclusion, deductions are becoming the new primitive data objects of coming generations of industrial software applications. I therefore advocate that a part of the research effort on digital libraries for deductive software should focus on the underlying theories, principles, and meta logical frameworks that model, analyze, reason, and verify deductions, programming languages, and communication infrastructures.

References

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[Sch00] Carsten Schürmann. *Automating the Meta-Theory of Deductive Systems*. PhD thesis, Carnegie Mellon University, 2000. CMU-CS-00-146.