Is Feature-Oriented Verification Useful for Hardware?

Kathi Fisler and Shriram Krishnamurthi

November 12, 2005

The structure of designs too often fails verification. Isolating fragments of designs that impact a particular property can make verification significantly more tractable. Because performing this isolation is challenging, verifiers often rely on the modular structure of the design for guidance. Unfortunately, the portions of designs that impact properties often span several modules. As a result, the verification engineer either has to decompose the properties around the design modules or apply more sophisticated decomposition methods that do not directly exploit the modular structure. This seems a lost opportunity, and raises a question: are there modularizations that enable designers to naturally express more of their knowledge that matters for verification?

Modules in hardware description languages generally correspond to physical subcomponents of a system (such as the CPU or RAM). Aligning design modules with physical components seems natural. Over the last decade, however, many researchers in software engineering and programming languages have explored modules that encapsulate user-defined *features* rather than fragments of implementations [1, 5, 8]. Intuitively, a feature is a piece of system functionality that is meaningful to an end user (an identifiable piece of functionality that an end user would pay for). A single system is a composition of the features that the end user wants. The large number of systems definable from a common set of features form a *product-line*. Feature-based constructs are uncommon, but not new, in hardware specification languages. The **extend** construct in Verisity's *e* language was motivated by the work on features from the software community [4]. To the best of our knowledge, however, they have not exploited this modularization for verification.

Feature-oriented modules are attractive in verification because properties often describe user-identifiable traits of a system. Many properties align with small sets of features. This alignment reduces, and often eliminates, the need for property decomposition. Features also support incremental reasoning about designs as they evolve. They suggest a two-stage verification methodology in which properties are first checked against individual features to determine constraints that the feature places on the rest of the system. As features are composed into products, the constraints are checked using lightweight analysis techniques. The constraints enable incremental verification and amortize verification costs over many products built from the same core features.

Over the last 5 years, we have been developing theories of incremental and modular model checking for feature-oriented systems expressed as state machines [6, 7]. Our work has shown that features induce a form of module composition that lies between purely sequential and purely parallel composition [3]. Furthermore, modular verification in this framework is best viewed as a combination of constraint generation and constraint solving, rather than as compositions of results from straightforward model checking [2]. Our work to date is largely theoretical but has been prototyped (with implementation) against some actual software designs.

The talk has two goals: first, to give an overview of the benefits, assumptions, and challenges of feature-oriented modeling and verification; second, to spark discussion as to whether this style has a meaningful role in hardware design. Key questions include (1) the extent to which features are useful for large-scale organization of hardware designs, (2) how hardware design flows might exploit the opportunities for incremental verification that features enable, and (3) how well feature-based decompositions align with challenging hardware verification tasks.

References

[1] Don Batory, Clay Johnson, Bob MacDonald, and Dale von Heeder. Achieving extensibility through product-lines and domain-specific languages: A case study. *ACM Transactions on Software Engineering and Methodology*, April 2002.

[2] Colin Blundell, Kathi Fisler, Shriram Krishnamurthi, and Pascal Van Hentenryck. Parameterized interfaces for open system verification of product lines. In *IEEE International Conference on Automated Software Engineering*, 2004.

[3] Kathi Fisler and Shriram Krishnamurthi. Modular verification of collaboration-based software designs. In *Symposium on the Foundations of Software Engineering*, pages 152–163. ACM Press, September 2001.

[4] Yoav Hollander, Matthew Morley, and Amos Noy. The *e* language: A fresh separation of concerns. In *Proceedings of TOOLS Europe*, March 2001.

[5] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In *European Conference on Object-Oriented Programming*, pages 220–242, 1997.

[6] Shriram Krishnamurthi, Kathi Fisler, and Michael Greenberg. Verifying aspect advice modularly. In *Symposium on the Foundations of Software Engineering*, November 2004.

[7] Harry Li, Shriram Krishnamurthi, and Kathi Fisler. Modular verification of open features through three-valued model checking. *Journal of Automated Software Engineering*, 12(3):349–382, July 2005.

[8] H. Ossher and P. Tarr. Multi-dimensional separation of concerns in hyperspace. Technical Report RC 21452(96717), IBM, April 1999.