Complete Specification Mining

Gianluca Martino
TUHH
Hamburg, Germany

Heinz Riener
EPFL
Lausanne, Switzerland

Görschwin Fey
TUHH
Hamburg, Germany

ABSTRACT

In this paper, we start to define a framework for *Complete Specification Mining*. We give a definition of functional completeness for property generation methods. We then propose a naïve technique for the generation of functionally complete sets of properties and demonstrate empirically the feasibility of the method.

KEYWORDS

design undestanding, specification mining, reactive systems

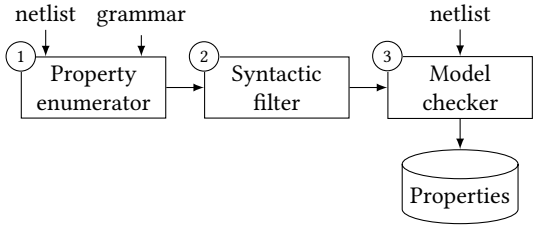

Figure 1: Overview of the property generation process

1 INTRODUCTION

Specification mining is a matter of interest for industry, e.g. JasperGold Property Synthesis or BugScope, and for academia [9, 11] because having tools which are able to generate sets of properties can greatly help in the work of an engineer. For example in tasks such as debugging of complex systems, update of legacy systems, or reverse engineering of unknown designs. Moreover when only a netlist representation of the circuit is available, the design intent may be hard to regain if some optimizations are introduced. In this case simulating the design is the only way for understanding the behavior. Properties are able to encode an higher view, the design intents, as a set of independent assertions.

For certain applications, a complete description of all the behaviors is required, i.e. high-level synthesis (HLS) requires a precise behavioral specification and the design constraints in order to obtain an RTL description. A possible use case for a property mining tool which is able to produce a complete specification could be IP-reuse. Starting from an existing design, using such a tool, it would be possible to get a complete specification. Such specification could be used to deduce parameters of the system. Similarly to HLS, modifying the desired parameters and reapplying synthesis techniques would realize a different design, but with the same guaranties.

The problem of having functionally complete sets of properties has been addressed with the goal of obtaining a full coverage when verifying a design [5].

In our paper, we give a definition of functionally complete sets of properties which applies specifically to property generation methods. Our definition uses the idea of reactive synthesis for the creation of a design, using the generated properties, which is sequentially equivalent to the original one.

Most of the currently used property mining techniques rely on simulation traces [9, 11]. This is a limitation because using traces it is practically impossible to generate a functionally complete set of properties, unless for very specific designs. This is due to the fact that the amount of information needed to encode all possible behaviors of a design in simulation traces grows very fast with the size of the design.

We show that Syntax-Guided Property Enumeration [4] can generate functionally complete sets of properties. In our experiments we explain how we obtained the properties and demonstrate that the set is functionally complete.

Our contributions are:

- the introduction of a notion of completeness which can be used as a quality metric for the sets of properties generated by specification mining tools;
- the proof that complete specification mining is feasible.

2 FUNCTIONAL COMPLETENESS OF TEMPORAL LOGIC PROPERTIES

Informally, a set T of temporal logic properties is a specification which describes the set of states that a system can go through. Such a set is *consistent* if it contains only non-contradicting properties. A *reactive system* is a system which maintains a continuous interaction with its environment. A consistent set of temporal logic properties T_c is *realizable* if there exists a reactive system which implements it [1]. *Reactive synthesis tools* [7] use a set of consistent properties T_c to synthesize a reactive system which satisfies the properties by construction.

Given two designs D_1, D_2 with the same input and output space, it is possible to combine D_1 and D_2 into a product machine which contains the two designs, running concurrently on the same inputs. The output function $\Lambda : S \times X \rightarrow B$, where S is the state space, X is the input space and $B = \{0, 1\}$ denotes the set of Boolean values, is 1 for a given state and input vector, if all pairwise corresponding outputs of the two circuits assume the same value.
Sequential equivalence checking proves that Λ is 1 for every reachable state and input vector. [10]

Definition 1. Given a set T of temporal logic properties and a design which models a Kripke structure S. A set T is *functionally complete* with respect to S iff any realization R_T of T is sequentially equivalent to S.

3 SYNTAX-GUIDED PROPERTY ENUMERATION

We generalize the idea of SyGuS [2] in the context of temporal logic and model checking. For a given Kripke structure S, we generate a

set $T = \varphi_1, \varphi_2, \ldots, \varphi_n$ of temporal logic formulæ that are satisfied by S, i.e., $S \models \varphi_i$ for $1 \leq i \leq n$, and obey to certain syntactic rules. The formulæ are generated by unwinding a context-free grammar G and model checked on S. Satisfied formulæ are kept and reported to a user, failing formulæ are discarded.

Given a Kripke structure S, a context-free grammar G and some termination criterion, the method for property generation works as illustrated in Fig. 1:

(1) The process starts from the property enumerator. The enumeration requires as inputs the grammar G and the netlist which encodes the Kripke structure S. As result, the property enumerator block outputs formulæ until either all the search space obtained from the current grammar is explored or the termination criterion is reached.
(2) Every formula generated passes through the syntactic filter. This step uses syntactic rules to detect redundant formulæ generated by the enumeration. If the formula does not comply with the syntactic rules, it is discarded.
(3) We finally employ the model checker to obtain the set T of properties that hold on S, which is realizable by construction.

Trace evaluation and vacuity detection [3] can be used in order to discard properties not holding in the design without employing the model checker (runtime improvement) and to remove all the properties which contain redundant information (improvement on the quality of the results), respectively.

An additional termination criterion, e.g., in form of a time limit or an upper bound on the maximum length of formulæ, is required to guarantee termination.

3.1 Grammar for completeness

A set of logic connectives is *adequate* iff all other connectives can be expressed in terms of it. As a first approximation, using any adequate set of logic connectives of a temporal logic which is *expressively complete* [8], it is possible to generate a functionally complete set of properties. Using Syntax-Guided Property Enumeration, it is possible to obtain a functionally complete set of properties by using a grammar which ensures that all possible states are represented.

In Fig. 2, we show the grammar used as input for this instantiation of Syntax-Guided Property Enumeration for complete specification mining: the grammar is composed of a subset of the logic connectives of CTL and of the set $\mathcal{AP}$ of atomic predicates containing the signals of the design. This grammar has been chosen because it permits to simplify the synthesis procedure.

In Section 4, we show that it is possible to obtain a functionally complete set of properties using this grammar, which contains a set of logic connectives that is not adequate.

4 EXPERIMENTS

The experiments have been done in order to empirically prove that complete specification mining is feasible. This has been achieved by using the grammar defined in Section 3.1 and enumerating a set of properties using Syntax-Guided Property Enumeration. The synthesis of the design has been done by grouping all obtained properties by the signal in P and using all the properties of each group to create a truth table. Using the truth tables generated, a design has been synthesized. Finally, using state-of-the-art tools,

```
1 S ::= p, p ∈ 𝒜𝒫
2 P ::= (S) | (¬S)
3 L ::= (P ∧ P)
4 L ::= (P ∧ L)
5 F ::= AG(L → AX P)
```

Figure 2: CTL grammar used

the original design and the synthesized design have been checked for sequential equivalence. As expected, the sequential equivalence check has been successful.

The experiment has been done using an AIGER file, modeling a counter, as a starting design. The design contains no inputs, 3 latches and one output. Using the procedure described, we have obtained a functionally complete set of properties, composed of 22 formulæ, in less than 10 seconds.

5 CONCLUSION

In this paper we defined a criterion for deciding functional completeness of sets of properties obtained through specification mining methodologies. This notion of completeness is not related to any of the previous metrics based on fault coverage [6].

We performed preliminary experiments in order to prove that a method which does complete specification mining is feasible. We demonstrated also that it is possible to design a grammar such that, using Syntax-Guided Property Enumeration, a complete set of properties is obtained.

In future work, more extensive experiments will be performed using larger designs and different approaches for complete specification mining.

REFERENCES

[1] M. Abadi, L. Lamport, and P. Wolper. 1989. Realizable and unrealizable specifications of reactive systems. In *International Colloquium on Automata, Languages, and Programming, (ICALP)*. Springer, 1–17.
[2] R. Alur, R. Bodík, E. Dallal, D. Fisman, P. Garg, G. Juniwal, H. Kress-Gazit, P. Madhusudan, M. M. K. Martin, M. Raghothaman, S. Saha, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, and A. Udupa. 2015. Syntax-Guided Synthesis. In *Dependable Software Systems Engineering*. 1–25.
[3] D. Bustan, A. Flaisher, O. Grumberg, O. Kupferman, and M. Y. Vardi. 2005. Regular Vacuity. In *Correct Hardware Design and Verification Methods, (CHARME)*. 191–206.
[4] G. Fey, T. Ghasempouri, S. Jacobs, G. Martino, J. Raik, and H. Riener. 2018. Design Understanding: From Logic to Specification. In *International Conference on Very Large Scale Integration, (VLSI-SoC)*. IFIP/IEEE.
[5] D. Große, U. Kühne, and R. Drechsler. 2008. Analyzing Functional Coverage in Bounded Model Checking. *Transactions on Computer Aided Design of Circuits and Systems (TCAD)* 27, 7 (2008), 1305–1314.
[6] Y. Hoskote, T. Kam, P.-H. Ho, and X. Zhao. 1999. Coverage estimation for symbolic model checking. In *Design Automation Conference (DAC)*. 300–305.
[7] S. Jacobs, N. Basset, R. Bloem, R. Brenguier, M. Colange, P. Faymonville, B. Finkbeiner, A. Khalimov, F. Klein, T. Michaud, G. A. Pérez, J.F. Raskin, O. Sankur, and L. Tentrup. 2017. The 4th Reactive Synthesis Competition (SYNTCOMP 2017): Benchmarks, Participants & Results. In *Sixth Workshop on Synthesis, (SYNT@CAV)*. 116–143.
[8] H. Kamp. 1968. *Tense Logic and the Theory of Linear Order*. Ph.D. Dissertation. University of California, Los Angeles.
[9] D. Neider and I. Gavran. 2018. Learning Linear Temporal Properties. arXiv:cs.LO/1806.03953
[10] C. A. J. van Eijk. 1998. Sequential equivalence checking without state space traversal. In *Design, Automation & Test in Europe, (DATE)*. IEEE, 618–623.
[11] S. Vasudevan, D. Sheridan, S. Patel, D. Tcheng, B. Tuohy, and D. Johnson. 2010. Goldmine: Automatic assertion generation using data mining and static analysis. In *Design, Automation & Test in Europe, (DATE)*. IEEE, 626–629.