RAT: A Tool for the Formal Analysis of Requirements*
(Tool Paper)

Roderick Bloem[1], Roberto Cavada[2], Ingo Pill[1], Marco Roveri[2], and Andrei Tchaltsev[2]

[1] Graz University of Technology — Inffeldgasse 16b/II - 8010 Graz - Austria
{rbloem, ipill}@ist.tugraz.at
[2] Fondazione Bruno Kessler - irst — Via Sommarive, 18 - 38050 Povo (Trento) - Italy
{cavada, roveri, tchaltsev}@itc.it

Abstract. Formal languages are increasingly used to describe the functional requirements of circuits. Although formal requirements can be hard to understand and subtle, they are seldom the object of verification. In this paper we present our requirement analysis tool, RAT. Our tool supports quality assurance of formal specifications. A designer can interactively explore the requirements' semantics and automatically check the specification against *assertions* (which must be satisfied) and *possibilities* (which describe allowed corner-case behavior). Using RAT, a designer can also investigate the realizability of a specification. RAT was successfully examined in several industrial projects.

1 Introduction

Formal specifications are becoming increasingly important, not only for verification, but also to describe design intent.

Traditionally, the verification effort focuses mainly on the design. A design is verified using either a golden model or a set of properties. This can be done either by simulation or by static verification. Either requires a large amount of effort on behalf of the user and is a time consuming part of the design cycle. Requirements, however, are seldom the *object* of verification. This is somewhat surprising, since industrial data show that about 50 percent of product defects originate in flawed requirements and that around 80 percent of rework effort can be traced back to requirement defects [12].

The use of formal requirements is a first and substantial step towards high quality specifications, but is obviously not enough to ensure the desired quality. RAT, our requirements analysis tool, supports the designer in the crucial task of writing high quality formal requirements of circuits. (We use specifications as a synonym for formal functional requirements.) RAT can be downloaded from `http://rat.itc.it`. It supports PSL [1], and provides a convenient graphical interface for the development, analysis, and management of a specification. Our current version draws from complementary techniques to explore requirement semantics, assure system traits, and check for realizability: *property simulation* [10], *property assurance* [10], and *property realizability*.

In the remainder of this paper we show how these techniques integrate, present a methodological guideline, give technical details, and report feedback from industry.

* Supported by the European Commission under contract 507219 (PROSYD).

W. Damm and H. Hermanns (Eds.): CAV 2007, LNCS 4590, pp. 263–267, 2007.

2 Requirements Analysis

Property Simulation provides the designer with an interactive method to understand the semantics of formal requirements by exploring their behavior one trace at a time. A designer can ask for an example behavior, *constrain* it by fixing the value of any signal for any given time step, and then check whether the altered trace is still allowed by the requirements. If not, the designer can ask for a different trace that is correct and adheres to the user-specified constraints. Although a property does not differentiate between inputs and outputs, the designer may do so. Based on such a classification she can perform a "what-if" analysis by setting inputs and asking to be presented with corresponding outputs. Dually, a "how-can" analysis can be performed by setting output signals and asking how, if at all, these outputs can be achieved. We provide an explanation of derived traces in the form of the property syntax tree plus the truth values of each subformula at every step. This helps the designer to understand how the subformulas and the property itself are evaluated along the trace. In a way, property simulation allows for a reverse-engineering of the property semantics much like a hardware design would be simulated.

Property Assurance provides the designer with a general means to assess whether she has written the right set of properties. First, property assurance can check that the requirements are consistent and do not contain a contradiction. Second, the designer can provide two sets of properties: Φ_A, a set of *assertions* that must be guaranteed, and Φ_P, a set of *possibilities* that describes corner cases that must be allowed by the requirements. Using assertions, a designer can check whether the requirements are strict enough to exclude any undesired behavior. With possibilities, she can check that they are not overly strict, and desirable behavior is allowed.

Property Realizability aims to verify whether there is a system that behaves according to the specification for any provided input sequence. To decide realizability, we split the requirements into assumptions on the environment and guarantees on the system behavior. Then we check for the existence of a system which can provide correct outputs for any inputs that are consistent with the environment assumptions. This problem can be seen as a two player game (the players being the environment and the system), where we have to determine a winning strategy (an implementation) for the system. Realizability is much more demanding than logical consistency. Indeed, there are logical consistent specifications that are unrealizable.

Figure 1 depicts a requirements analysis process that integrates the three proposed techniques. First, the designer comes up with initial approximations of the requirements Γ, assertions Φ_A, and possibilities Φ_P. We propose an iterative approach, checking whether the requirements are consistent, whether they allow for all possibilities stated in Φ_P, and whether they do not contradict any assertion in Φ_A. For any problem, the designer is presented with diagnosis information, and consequently refines Γ, Φ_A, and Φ_P to fix it. After any change, the requirements are verified again for consistency and for adherence to Φ_A and Φ_P. Finally, realizability of the specification is checked. If the requirements are unrealizable, the designer is requested to revise the specification.

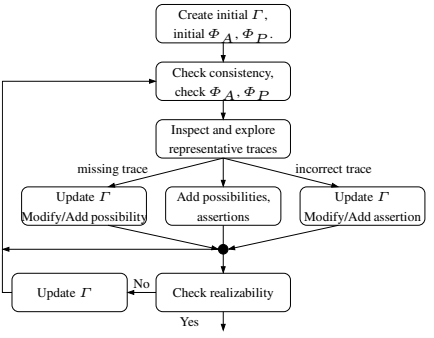

Fig. 1. Guidelines

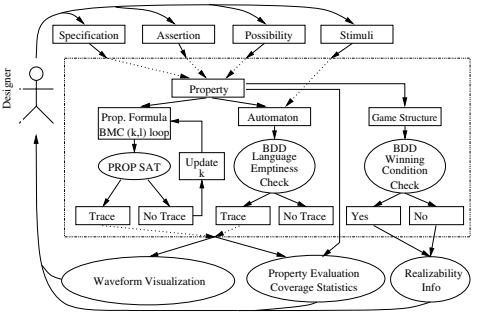

Fig. 2. RAT architecture

3 Technical Aspects

Property simulation and assurance rely on *Automata-based* and *Bounded Model Checking* (BMC) techniques [9,3]. For both approaches we derive an automaton and check its language for emptiness, for a (bounded) witness or counterexample for the task at hand.

To decide the realizability of a specification we construct a two-player game between the system and the environment [11]. The goal of the system is to satisfy the specification by delivering correct outputs considering the so far encountered input, regardless of the input sequence provided by the environment. Realizability can be decided by checking whether this game is winning for the system or the environment. For a winning

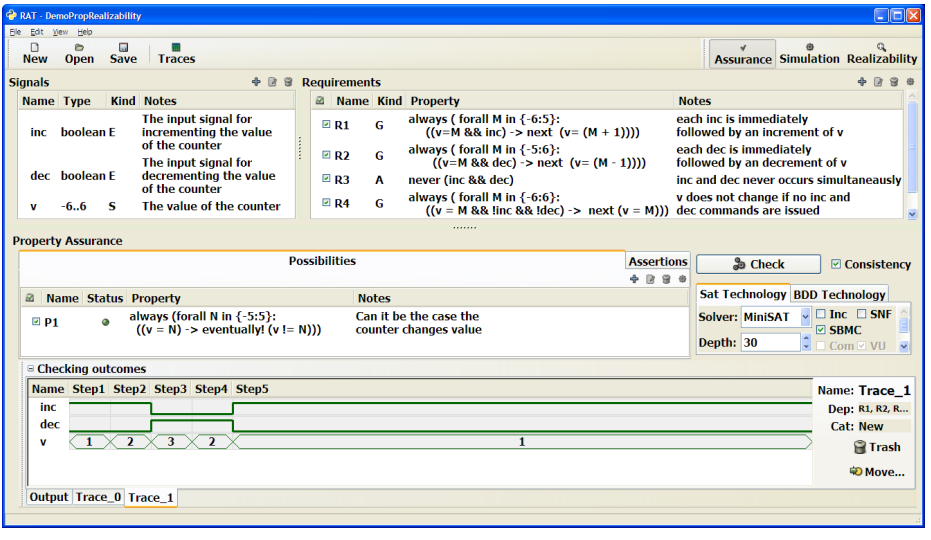

Fig. 3. RAT's GUI

environment, the specification is *unrealizable* and must be modified. If the system wins, the specification is *realizable*: an implementation can be constructed.

The concepts presented in the previous sections have been used in the design and development of RAT [10]. Figure 2 depicts a high level architecture of RAT. An example of the graphical user interface can be found in Figure 3. RAT's verification capabilities currently rely on the NuSMV [7] and VIS [6] model checkers, extended to provide the devised functionalities [4,8]. However, RAT has been designed and implemented for an easy plug in of other verification engines, to support further languages and verification algorithms. Additional information on RAT can be obtained from its web site http://rat.itc.it

4 Experimental Analysis

In [2] IBM, Infineon, STMicroelectronics, and OneSpin examined several new techniques and tools for property-based requirements specification, including a RAT prototype supporting property simulation and assurance. The case studies included transport frontends, protocols, a bridge, SOC interconnects, and other industrial design blocks.

Although our tool was a prototype when the case studies were done, our technology appeals to designers: "We found the concept of property simulation attractive as it allows a developer to debug her/his own PSL code easily, quickly and independently". Several bugs in the properties were found, and property assurance has been "used effectively in specific cases to prove that one set of properties can be substituted by another". The interface and usability features provided by RAT "make the development process easier and provide an enjoyable development experience". IBM's experience with the tool prompted them to start "to design and develop a feature similar to property simulation in RuleBasePE soon after starting this case study". One of the projects also quantified an economical benefit; estimated 1.5 person months for property debugging shrank to 0.5 person months using RAT.

Regarding realizability, in [5] the realizability proof of real system-on-chip designs has been shown using the same algorithms implemented in RAT [8].

Acknowledgements

Special thanks go to Alessandro Cimatti for his support in this work, and to Simone Semprini who contributed to a preliminary version of this tool.

References

1. Accellera. Property specification language — reference manual, version 1.01 (April 2003)
2. Auerbach, G., Benalycherif, L., Fedeli, A., Fisman, D., McIsaac, A., Winkelmann, K.: Case studies in property-based requirements specification, Prosyd Delivarable D1.4/1 (November 2006) `http://www.prosyd.org`
3. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: Cleaveland, W.R. (ed.) ETAPS 1999 and TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999)

4. Bloem, R., Cavada, R., Eisner, C., Pill, I., Roveri, M., Semprini, S.: Manual for property simulation and property assurance tool, Prosyd Delivarable D1.2/4-5 (November 2005) http://www.prosyd.org
5. Bloem, R., Galler, S., Jobstman, B., Weiglhofer, M., Piterman, N., Pnueli, A.: Automatic hardware synthesis from specifications: A case study. In: Proceeding of DATE'07 (to appear)
6. Brayton, R.K., et al.: A system for verification and synthesis. In: Henzinger, T., Alur, R. (eds.) CAV 1996. LNCS, vol. 1102, pp. 428–432. Springer, Heidelberg (1996)
7. Cimatti, A., Clarke, E.M., Giunchiglia, F., Roveri, M.: NuSMV: a new Symbolic Model Verifier. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 495–499. Springer, Heidelberg (1999)
8. Cimatti, A., Roveri, M., Tchaltsev, A.: Manual for property realizability tool, Prosyd Delivarable D1.2/8 (December 2006) http://www.prosyd.org
9. Kurshan, R.: Computer-Aided Verification of Coordinating Processes: the automata theoretic approach. Princeton University Press, Princeton, NJ (1994)
10. Pill, I., Semprini, S., Cavada, R., Roveri, M., Bloem, R., Cimatti, A.: Formal analysis of hardware requirements. In: Design Automation Conference, pp. 821–826 (2006)
11. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. In: VMCAI, pp. 364–380 (2006)
12. Wiegers, K.E.: Inspecting requirements. StickyMinds Weekly Colum (July 2001)