Formal Methods in an Industrial Environment

Jorge R. Cuéllar

Siemens AG
Corporate Technology ZT SE 4
Otto-Hahn-Ring 6
D-81739 Munich, Germany
Jorge.Cuellar@mchp.siemens.de

Industrial applications of formal techniques may be divided roughly in two types:

- *Consultant Service.* A formal method team inside or outside the company acts as a consultant to the engineering group responsible for the design. This is the most common type of application of formal methods in industry found in the literature and to a certain extent it is usual at Siemens.
- *Ready to use tools.* The formal method group only *provides* tools, training and support, while the design or verification task itself is planned and realized by the group of engineers responsible for the design. They use the formal method tools increasingly as a routine part of their development process, in very the same way as they use editors, compilers, simulation tools or data bases. This is turning to be the most common situation within Siemens, with about two thirds of the formal methods group dedicated to the development of tools and interfaces to the different design languages and to the established development tools in the application domains.

The formal methods group has successfully developed the tools CVE (Circuit Verification Environment) and SVE (System Verification Environment), which are used within the business divisions of Siemens as *ready to use tools.* They provide interfaces to quite a few languages, in particular to SDL (used at the divisions public networks, private networks, and others), HiGraph and AWL (both used in particular at the automation divisions), and hardware description languages such as VHDL, Verilog or EDIF. Existing programs (not: abstract versions of them) are used directly as input to application specific compilers that generate automatically finite state machine representations. The specified properties and assumptions about the environment (physical process, other system components) are formulated in customised, user-friendly specification languages and are translated internally into temporal logic and/or finite state machines. The model checking algorithms verify whether the given properties hold or, if not, generate a counter example trace. This counter example is transformed to a trace at the application level and presented to the user in their usual environment. For instance, in HiGraph (a new programming tool for the Siemens Automation Systems Simatic), the user can animate the counter example sequence in a graphical debugging environment or view it in a textual form.

Let us briefly look at concrete application examples and present some figures. The tools (see [8]) have been applied for instance to programs developed

by Siemens customers controlling components in several production lines, for example drilling stations for the production of automobile engines. These HiGraph program sizes range between 1000 and 3000 LOCs and contain several hundred boolean variables. Also we would like to mention the verification of protocol software in mobile phones produced by Siemens (see [12]). Each mobile phone contains an implementation of the highly complex GSM protocol. Typically, the number of reachable states of the models generated range between 10^7 and 10^{13}. Other examples include ISDN Protocols (network signaling for public switching networks or call processing for the private branch exchange HICOM), of 600 pages and 110 kLOCs of SDL Code, respectively. Applications in HW design include the areas of consumer electronics, industrial automation, telecommunication, and multiprocessor systems. There, ASICS with up to a million gates are routinely automatically compared.

The rest of the formal methods team is on the one hand acting as a consultant group within the corporation and, on the other hand, developing tools that, we hope, will be eventually used by the business divisions on their own. Their activities center around DES (supervising control of discrete event system, [2, 11]), synchronous languages (in particular SCSL, synchronous control specification language, a new language closely related to TLT [5], [6] and CSL [9]), abstract state machines (formerly evolving algebras, see [1]), and theorem proving, see [13] and [3, 4]. This last activity is being currently reduced, due mostly to the poor acceptance of the methods by the engineers. The DES Tools could be used by the engineers to synthesise controllers for automation systems, or to check their consistency (for instance, the "non-blocking" property). Synchronous Languages ([7]) are very interesting because they provide a higher abstraction level that in general is easier to verify than a conventional, sequential solution. There are quite efficient compilation techniques for synchronous languages. Within the ESPRIT project SACRES ([10]), we are applying the SVE model checker to synchronous programming languages such as SIGNAL and Statemate, and supporting several industrial case studies in the aerospace and automotive industries. ASMs are currently successfully used for the *documentation* of the design process.

Formal Methods for the design or verification of HW or SW systems is an emerging technique in industry. As with *any other* new methodology, the decision to use it is driven by considerations of benefits, costs, efficiency and risks: for instance,

- *Are the benefits of the new methodology clear to managers and engineers?* For most formal methods the answer is *yes*. Indeed, the benefits of using formal methods to validate/verify the correctness of the product are easy to explain: Existing software products may fail and the costs due to their malfunctioning is high. Formal methods have demonstrated success in specifying, designing and verifying industrial systems.
- *Do the benefits of the new method exceed the costs of converting to it?* This is somewhat difficult to determine. If the new method implies a drastic change in the established design flow, the costs will probably be high or unpredictable. For instance, a design or verification methodology based on estab-

lishing a sequence of design descriptions at different abstraction levels, linked together by a formal refinement notion, is found to be difficult. This is even true in the case of a finite-state setting with Model Checking and worse for the theorem-proving approach. The situation is that high level descriptions are not used further in the current design flow. The construction and maintenance of the consistency mapping of abstract descriptions with the actual system is a not trivial task, poorly supported with tools and costly in time and effort.
- *Can you estimate the risks?* For most formal methods, that require a change in the design process: the answer is *no*.

In order for a formal method to be used in a given, typical industrial environment, the following criteria should be met:

- the method must handle examples of industrial size, at reasonably low levels of abstraction,
- the costs in terms of effort, time, etc. should be less or comparable to the ones in the existing design process,
- the method should be accepted by engineers who do not have a strong background in logic, and
- the technique must be easily integrated into the existing design flow.

Most of these criteria favour the Model Checking approach.

References

1. ASM-Bibliography. http://www.eecs.umich.edu/gasm/. WWW page.
2. B. A. Brandin, "The Real Time Supervisory Control of an Experimental Manufacturing Cell", IEEE Transactions on Robotics and Automation, Vol. 12, No. 1, February 1996, pp. 329-342.
3. H. Busch. A Practical Method for Reasoning About Distributed Systems in a Theorem Prover. In *Higher Order Logic Theorem Proving and its Applications - 8th International Workshop, Aspen Grove, UT, USA, Proceedings*, pages 106–121. Springer-Verlag, LNCS 971, September 1995.
4. H. Busch. Proving Liveness of Fair Transition Systems. In J. v. Wright, J. Grundy, and J. Harrison, editors, *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOL'96*, volume 1125 of *LNCS*, pages 77–92. Springer-Verlag, August 1996.
5. J. R. Cuéllar and I. Wildgruber. A TLT Solution. In J.-R. Abrial, E. Börger, and H. Langmaak, editors, *Formal Methods for Industrial Applications. Specifying and Programming the Steam Boiler*, volume 1165 of *LNCS*, pages 165–183. Springer-Verlag, 1996.
6. Jorge Cuéllar, Dieter Barnard, and Martin Huber. Rapid Prototyping for an Assertional Specification Language. *TACAS'96, LNCS 1055*, March 1996.
7. N. Halbwachs. *Synchronous Programming of Reactive Systems.* Kluwer, 1993.
8. M. Hölzlein, Th. Filkorn, P. Warkentin, and M. Weiss. Eine Verifikationskomponente für HiGraph. Volume 1397 of *VDI-Berichte.* VDI-Verlag, Düsseldorf, 1998.

9. Klaus Nökel and Klaus Winkelmann. The FZI Production Cell Case Study: A distributed solution using TLT. In *Formal Development of Reactive Systems: Case Study Production Cell*, volume 891 of *LNCS*. Springer-Verlag, 1995.
10. Sacres Esprit Project. http://www.ilogix.co.uk/ilogix/sacres.html. WWW home page.
11. P. J. Ramadge and W. M. Wonham, "The Control of Discrete-Event Systems", IEEE Proceedings, Vol. 77, No. 1, January 1989, pp. 81-98.
12. Franz Regensburger and Aenne Barnard. Formal Verification of SDL Systems at the Siemens Mobile Phone Department. In *TACAS 1998*, Lecture Notes in Computer Science. Springer-Verlag, 1998.
13. Karl Stroetmann. SEDUCT — a proof compiler for first order logic. In Manfred Broy and Stefan Jänichen, editors, KORSO*: Methods, Languages, and Tools for the Construction of Correct Software*, volume 1009 of *Lecture Notes in Computer Science*, pages 299–316. Springer Verlag, 1995.