COSPAN

R. H. Hardin[1], Z. Har'El[2] and R. P. Kurshan[1]

[1] Bell Laboratories, Murray Hill, New Jersey 07974
[2] Dept. of Mathematics, Technion, Haifa, Israel

Abstract. COSPAN (*Co*ordination *Sp*ecification *An*alizer [AKS83]) is an algorithmic computer-aided verification system. It's semantic model [Ku94] is founded on ω-automata: for a *process* P (modelling a system to be verified) and a *task* T which P is intended to perform, *verification* consists of the automata language containment test

$$\mathcal{L}(P) \subset \mathcal{L}(T) \ .$$

If the test fails, COSPAN presents an error track which illustrates the error. Typically, P is not monolithic, but is represented as a ("synchronous") parallel composition $P = P_1 \otimes \cdots \otimes P_k$ of component processes (all modelled as ω-automata). Asynchronous coordination of component processes may be modelled through nondeterminism in the components. The process model can be set either to Mealy or Moore machines.
COSPAN has been used on commercial applications for over a decade, both for software and hardware design verification [HK90]. Recently, it has been implemented as the "verification engine" in the commercial hardware verification tool FormalCheck^{TM}, which is supported for hardware verification by the Bell Labs Design Automation center.
The COSPAN application domains and utilities are enumerated.

1 Methodology

COSPAN supports top-down design development through *successive refinements*, in which one may start by verifying an abstract prototype design, and then successively refine the prototype, verifying new properties in each respective refinement, in such a way that each successive refinement inherits all the properties verified in all previous steps. This is possible because ω-automata define *linear-time* behaviors (in contrast with *branching-time* behaviors as defined by logics such as CTL [CE82]). From the final design model, COSPAN will generate C code as well as input to the Lucent Technologies synthesis tool $BESTMAP^{TM}$ in order to implement automatically its models as software or hardware.

Its analysis algorithms include automated *localization reduction*, *symmetry reduction* and the more general user-defined *homomorphic reduction* [Ku94], used to limit the computational complexity of verification (which in general is intractable in the number of coordinating components k). Localization is an (automatically) generated homomorphic reduction, which reduces the model P relative to the property T which is to be verified. This reduction conservatively

discards parts of the design irrelevant to T, and resizes retained portions relative to the other retained parts. If the design changes, already-verified properties need not be re-verified if the changes lie outside of their respective localizations. Whether this is the case can be checked through a computationally simple CRC map of the localized parse trees before and after the change, for each respective property ("regression verification" [HKRS96]). In homomorphic verification, the user produces two models (possibly with completely different sets of events and system variables) and a language map which relates the two; COSPAN checks that the map in fact defines a (behavior-preserving) homomorphism between the two models. This check may be done component-wise, which is important for circumventing the possibly intractable complexity of performing the check on the entire system at once.

COSPAN supports *real-time* verification, implemented with Rajeev Alur in terms of successive approximations [AIKY93] in order to limit its computational complexity.

In its core routines, COSPAN can use either symbolic- (BDD-based) or explicit-state enumeration algorithms. Both algorithms are "on the fly" in the sense that errors may be detected before exploring the entire state space. The error track that COSPAN produces optionally contains line-number/source-file information for each variable to support *back-referencing*, indicating where in the source the given variable was assigned the value indicated in the error track (at the point where the other variables were assigned their designated respective values). The BDD-based algorithms (based on a new BDD package implemented by David Long) use partitioning, dynamic reordering and a version of the Emerson-Lei algorithm [EL86] for the language-containment test. The explicit-state algorithms optionally invoke several caching and hashing options, a generalized Hopcroft state minimization algorithm [Gr73], and use the Tarjan strongly connected components algorithm [Ta72] for the language-containment test.

2 Language

The *state* of a computer program at an instant of time is the simultaneous value of all its respective variables (assuming that is well-defined). As space complexity is a principal bottleneck in algorithmic verification, the dimension of this state vector is an important factor in the over-all complexity of the verification [Ku94]. If the values of some variables are functions (or relations) of the values of other variables, then the effective dimension of the state vector, as a factor in computing space complexity, is the actual dimension less the number of dependent variables.

If a programming language syntactically draws a distinction between such *state* variables, and the remaining *combinational* variables whose [possible] values at each instant are a function [relation] of the values of the state variables, then this distinction can be exploited in algorithmic verification. Programming languages designed to describe integrated circuits ("hardware description languages" or HDL's) often distinguish between such state variables, used to desig-

nate *latches* or computer memory and combinational variables, used to designate *logic*, for the reason that the performance and fabrication cost of hardware are governed by the same factors as algorithmic verification. For hardware synthesized automatically from HDL code, the syntactic distinction between state and logic in the HDL is reflected in a corresponding physical distinction in the hardware.

An even more fundamental distinction between hardware and software, as it pertains to algorithmic verification, is the use of recursion and pointers, less common in HDL's and more problematic for algorithmic verification.

All of these issues have been significant factors in the generally greater acceptance of algorithmic verification in the commercial hardware design process than in the commercial software design process, and are reflected in the commercial languages for which interfaces to COSPAN have been built. Notwithstanding this, the principles of algorithmic verification and their implementation in COSPAN are applicable equally to hardware and software.

To date, interfaces to COSPAN have been written for the commercial HDL's Verilog and VHDL, the CCITT-standard protocol specification language SDL, and the non-commercial languages HSIS [HSIS94] from UC Berkeley, Holzmann's Promela [Ho91] and Lamport's TLA [La94]. However, it would be feasible to make interfaces to any other languages for which the above language issues pertaining to verification can be adequately addressed.

Conversely, the COSPAN parser can translate its native S/R language into input to the verification tools SMV [Mc93] and SPIN [Ho91] and the automated theorem-provers HOL [Go88] and Larch [GG91] (via TLA).

COSPAN 's native language S/R distinguishes between state variables and combinational variables. The language supports nondeterministic, conditional (*i.e.*, if-then-else) variable assignments; variables of type bounded integer, enumerated, Boolean and pointer (to structures); arrays and records; and integer and bit-vector arithmetic. Modular hierarchy, scoping, parallel and sequential execution, homomorphism declaration and general ω-automaton fairness (acceptance) are supported as well. Property specification also is supported through a library of parameterized automata designed to facilitate the definition of any ω-regular property. Thus, the user need not deal with automata acceptance conditions directly, but may confine coding to conventional programming plus the definition of the properties to be verified and system constraints (if any) specified by assigning propositional expressions to parameters of the library automata. The commercial FormalCheck version of COSPAN provides a graphical interface for this purpose.

3 Utilities

As already described above, the principal COSPAN algorithms include:

- explicit- and symbolic-state enumeration language containment tests
- localization reduction

- homomorphic verification
- timing verification
- state space minimization
- symmetry reduction
- regression verification
- C code generation (model implementation)
- back-referencing from an error track to the source

Additionally, COSPAN supports the following options:

- data-path profiling; manual variable ordering (for BDD's)
- Wolper's [WL93] k-bit generalization of Holzmann's state-vector bit-hashing
- over-writing of states outside of the search path (explicit enumeration)
- randomized property checking
- automated sub-model extraction
- stability checking (for asynchronous models)
- deadlock detection; CTL AGEFp (*NOT* preserved by refinement)
- embedding of C code into S/R (for implementation and test generation)
- listing of the transition graph
- suspension, termination, polling and restarting verification runs
- interactive "check-pointing" (single path exploration)

4 To Obtain

A version of COSPAN is available to universities for research and educational purposes, at no charge: inquire to k@research.bell-labs.com .

References

[AKS83] S. Aggarwal, R. P. Kurshan, D. Sharma, A Language for the Specification and Analysis of Protocols, PSTV III, North-Holland (1983) 35–50.

[AIKY93] R. Alur, A. Itai, R. P. Kurshan, M. Yannakakis, Timing Verification by Successive Approximation, LNCS **663** (1993) 137–150.

[CE82] E. M. Clarke, E. A. Emerson, Design and Synthesis of Synchronization Skeletons for Branching Time Temporal Logic, LNCS **131** (1982) 52–71.

[EL86] E. A. Emerson, C. L. Lei, Efficient Model Checking in Fragments of the Propositional Mu-Calculus, LICS 1986, 267–278.

[GG91] S. J. Garland, J. V. Guttag, A Guide to LP: the Larch Prover, Technical Report 82, Digital Equipment Corporation, 1991.

[Go88] M. Gordon, A Proof-Generating System for Higher-Order Logic, Kluwer SECS **35**, 1988, 73–128.

[Gr73] D. Gries, Describing an Algorithm by Hopcroft, Acta Inf. **2** (1973) 97–109.

[HKRS96] R. H. Hardin, R. P. Kurshan, J. A. Reeds, N. J. A. Sloane, Regression Verification, Bell Laboratories TM11272-960502-14 (1996).

[HK90] Z. Har'El, R. P. Kurshan, Software for Analytical Development of Communications Protocols, *AT&T Tech. J.* **69** (1990) 45–59.

[Ho91] G. J. Holzmann, *Design and Validation of Computer Protocols*, Prentice Hall, 1991.

[HSIS94] A. Aziz, *et. al*, HSIS: A BDD-Based Environment for Formal Verification, Proc. 31st Design Automation Conf., 1994.

[Ku94] R. P. Kurshan, *Computer-aided Verification of Coordinating Processes – The Automata-Theoretic Approach*, Princeton Univ. Press, 1994.

[La94] L. Lamport, The temporal logic of actions, TOPLAS **16** (1994) 872–923.

[Mc93] K. L. McMillan, *Symbolic Model Checking*, Kluwer, 1993.

[Ta72] R. E. Tarjan, Depth-First Search and Linear Graph Algorithms, SIAM J. Comput. **1** (1972) 146–160.

[WL93] P. Wolper, D. Leroy, Reliable Hashing Without Collision Detection, Lecture Notes in Computer Science (LNCS) **697** (1993) 59–70.