# A structured modelling framework to capture symmetry in circuits

Ashish Darbari
Oxford University Computing Lab
Wolfson Building
Parks Road
Oxford, OX1 3QD
ashish@comlab.ox.ac.uk

## 1 Introduction

Symbolic trajectory evaluation (Seger and Bryant, 1995) or STE in short has been successfully used in verification of large industrial sized circuit designs. The data abstraction in STE via Xs is often insufficient to bring about any reasonable reduction in the size of the verification problem. For circuits with large number of state holding elements like random access memories (RAM), content addressable memories (CAM) and caches, these reduction techniques are not adequate. It is also the case that several such circuits exhibit symmetry in their structure, and intuition suggests that exploiting the presence of these symmetries can render significant reductions in the size of the verification problem (C.N. Ip and D.L. Dill, 1993; Pandey and Bryant, 1997; McMillan, 2000; Clarke et al., 1998; Sistla and Godefroid, 2004; Calder and Miller, 2002). Our motivation is to be able to perform symmetry based property reduction for STE model checking. In order to do so, we need to be able to have a method of discovering symmetries in circuits. For circuits represented as graphs, a significant proportion of verification effort using symmetry based reduction, goes in *detecting symmetries* in models (Pandey and Bryant, 1997) using subgraph isomorphism. We believe that the problem of symmetry detection is best addressed when symmetries are discovered by using special types in a language (C.N. Ip and D.L. Dill, 1993). In this case the problem of symmetry detection reduces to a problem of type checking.

We have worked on the design and implementation of a type system that is used for designing a structurally symmetric class of circuits. The type system is a way of defining an *abstract data type of models*, rather than attempt of designing an HDL like Verilog or VHDL. We have used this to design several symmetric circuits starting from basic logic gates to multiplexers, comparators, registers, RAMs, CAMs and circuits that involve multiple CAMs. These circuits appear regularly in all modern microprocessor designs.

## 2 Design of the framework

Our methodology of designing the framework is based on identifying the following key issues:

1. Propose a sufficiently generic type for circuits.
2. Define the functions that will be used for constructing symmetric circuit blocks.
3. Provide the type judgement rules.
4. Articulate the mathematical definition of structural symmetry.
5. Prove a type soundness theorem, which says that every circuit definition that is well behaved with respect to the typing rules, will have structural symmetry.

A circuit is said to have structural symmetry with respect to a specific set of inputs and outputs when the circuit's behaviour remains unchanged under permutation of those specific set of inputs and outputs. The specific inputs in this case can be termed as symmetric inputs. Every circuit can have some inputs which will be symmetric[1]. There would be some inputs that would not have any role in the symmetry of the circuit, and they are referred to as non-symmetric inputs. Thus a circuit can have the following type:

$$circ : (bool\ list)\ list \rightarrow (bool\ list)\ list \rightarrow (bool\ list)\ list$$

[1]In case the circuit has no symmetry the set of symmetric inputs will be empty.

where the type ($bool\ list$) denotes the set of values of any one group of wires, and the list of Boolean lists (($bool\ list$) $list$) denotes the collection of several sets of inputs. The first argument of the circuit type denotes the values of non-symmetric inputs, the second argument denotes the values for symmetric inputs and the third argument the outputs of the circuit.

We use several well known functions on lists, from functional programming, like $head$, $tail$, $append$, $map$, $map2$, $fold$ and so on, as building blocks for writing higher-order combinators that are used to define symmetric circuit blocks. The type of symmetric circuit blocks is

$$circ : (bool\ list)\ list \rightarrow (bool\ list)\ list$$

The type judgement rules are given by defining a typing predicate $SS$ (symmetry-safe) on the circuit construction combinators, which is defined inductively using nine rules. Arbitrary permutations on lists can be obtained by composing the $swap$ function, which swaps two elements of the list. Symmetry is thus defined in terms of a $swap$ function on lists. A necessary constraint on the symmetric inputs, is that all inputs that are contained in the list of symmetric inputs, should have equal length. This is expressed by the function $CheckLength$.

$$\begin{aligned} Sym\ \ c \ \ &\triangleq \ \ \forall sym.\ CheckLength\ \ sym\ \ \supset \\ &\qquad\qquad \forall i\,j.\ \ map(swap(i,j))(c\ \ sym)\ \ =\ \ c\,(map(swap(i,j))\ \ sym) \end{aligned}$$

Our last step in the design is to prove the type soundness theorem, which in our case states that every circuit block that is well-behaved with respect to the type judgement rules, exhibits symmetry. Thus we prove the following type soundness theorem:

$$\vdash \forall circ.\ SS\ \ circ\ \ \supset\ \ Sym\ \ circ$$

The proof of the soundness theorem relies on proving several properties about lists, and permutations on lists. We used some of the properties about lists, that appear in the list theory in HOL 4, and extended that library for our use, with many other properties that talk about permutations on lists, and functions such as $map$, $map2$, and $fold$. Once we prove that the type system is correct, we can use the type rules to infer whether or not a given circuit is symmetry safe ($SS$). We have written tactics that can automate the process of type checking, which takes typically a few seconds. Structured circuit models validated by type checking, are compiled by a combination of HOL functions and an ML program, to a flat netlist usable for simulations in Intel's symbolic simulator Forte.

## Acknowledgements

Thanks to my supervisor Tom Melham for providing continual support in this work. This work also benefited in its early stages, by discussions with John O' Leary and Robert Jones at Intel, and Joe Hurd, Tom Newcomb and Ed Smith at Oxford. Hasan Amjad provided useful comments on the abstract.

## References

M. Calder and A. Miller. Five ways to use induction and symmetry in the verification of networks of processes by model-checking, 2002.

Edmund M. Clarke, E. Allen Emerson, Somesh Jha, and A. Prasad Sistla. Symmetry Reductions in Model Checking. In *CAV '98: Proceedings of the 10th International Conference on Computer Aided Verification*, pages 147–158, London, UK, 1998. Springer-Verlag. ISBN 3-540-64608-6.

C.N. Ip and D.L. Dill. Better Verification through Symmetry. In D. Agnew, L. Claesen, and R. Camposano, editors, *Computer Hardware Description Languages and their Applications*, pages 87–100, Ottawa, Canada, 1993. Elsevier Science Publishers B.V., Amsterdam, Netherland.

K L. McMillan. A Methodology for Hardware Verification using Compositional Model checking. *Science of Computer Programming*, 37(1-3):279–309, 2000.

Manish Pandey and Randal E. Bryant. Exploiting Symmetry When Verifying Transitor-Level Circuits by symbolic trajectory evaluation. In *CAV*, pages 244–255, 1997.

Carl-Johan H. Seger and Randal E. Bryant. Formal Verification by Symbolic Evaluation of Partially-Ordered Trajectories. *Formal Methods in System Design: An International Journal*, 6(2):147–189, March 1995.

A. Prasad Sistla and Patrice Godefroid. Symmetry and reduced symmetry in model checking. *ACM Trans. Program. Lang. Syst.*, 26(4):702–734, 2004.