Inference of Numerical Relations from Digital Circuits

Enric Rodríguez-Carbonell, Jordi Cortadella

Universitat Politècnica de Catalunya
Jordi Girona, 1-3 08034 Barcelona (Spain)
www.lsi.upc.edu/{~erodri,~jordicf}

Abstract. We propose an approach for reverse-engineering arithmetic circuits, namely for discovering numerical relations from digital circuits. Boolean values and logical functions are abstracted up to a numerical domain with integer values and polynomials. This approach can be used as a pre-processing step to alleviate the computational complexity of formal verification of digital circuits. The method is illustrated with a simple example: binary adders.

1 Introduction

Errors in arithmetic hardware, e.g. the Pentium division bug [1], may cause huge economical disasters, and even the loss of many human lives. However, the verification of arithmetic circuits is still one the most challenging problems in the verification community. In particular, current methods such as equivalence checking do not perform well when applied to integer multipliers at industrial scale. This calls out for new techniques designed to exploit the properties of integer arithmetics, which may be combined with existing logic-oriented methods.

We propose an approach for reverse-engineering arithmetic circuits: given a circuit, our goal is to decide if there exists a numerical relation between some of the input signals and output signals, and if so, to find it. This could be used when verifying circuits with smaller arithmetic parts, e.g. an ALU (Arithmetic Logic Unit), to alleviate the state explosion problem of other techniques.

2 Example: Looking for Addition Relations

We illustrate the approach when looking for addition relations. Let us consider the simplest example of an adder, a full-adder. Fig. 1 shows an implementation of the circuit. It has input signals x, y and c_{in} and output signals s and c_{out}, which are related by the following linear addition equation:

$$s + 2c_{\text{out}} = x + y + c_{\text{in}} \tag{1}$$

The logical expressions of the output signals in terms of the input signals are $s = x \text{ XOR } y \text{ XOR } c_{\text{in}}$, $c_{\text{out}} = (x \text{ AND } y) \text{ OR } (x \text{ AND } c_{\text{in}}) \text{ OR } (y \text{ AND } c_{\text{in}})$. Our idea is to abstract the logical operators by polynomials, interpreting the boolean values *true* and *false* as the integer values 1 and 0 respectively. Namely,

$$x \text{ AND } y = xy \qquad x \text{ XOR } y = x + y - 2xy$$
$$x \text{ OR } y = x + y - xy \qquad \text{NOT } x = 1 - x$$

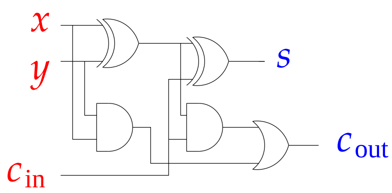

Fig. 1. Full adder

In this example, the output signals can be expressed as follows:

$$s = x + y - 2xy + c_{\text{in}} - 2c_{\text{in}}x - 2c_{\text{in}}y + 4c_{\text{in}}xy$$
$$c_{\text{out}} = xy + c_{\text{in}}x + c_{\text{in}}y - c_{\text{in}}^2xy - x^2yc_{\text{in}} - xy^2c_{\text{in}} + x^2y^2c_{\text{in}}^2$$

Since the values of the signals we are interested in are 0 and 1, we can simplify the expressions above by using that for any signal z, $z^2 = z$. This allows us to have just monomials of degree 1 in each of the variables. In this case:

$$s = x + y - 2xy + c_{\text{in}} - 2c_{\text{in}}x - 2c_{\text{in}}y + 4c_{\text{in}}xy \quad (2)$$
$$c_{\text{out}} = xy + c_{\text{in}}x + c_{\text{in}}y - 2c_{\text{in}}xy \quad (3)$$

As we are interested in getting linear expressions, we can eliminate from these equations the non-linear terms by *Gaussian elimination*. Formally, considering the set of polynomials as a vector space, we project the linear subspace generated by the output equations on the linear monomials. In this case, in order to eliminate the quadratic term xy we can add twice Eq. 3 plus Eq. 2; it turns out that all other non-linear terms also vanish, and finally we get Eq. 1, which is the equation of the full adder. In general, there may be more non-linear terms to eliminate than equations. So it may be the case that some non-linear terms cannot be eliminated, and then we do not obtain any addition relation.

Still, for some circuits the addition equation may not be linear. For instance, for a carry-lookahead adder of n bits with input signals $\mathcal{I} = \{x_0, ..., x_{n-1}, y_0, ..., y_{n-1}, c_{\text{in}}\}$ and output signals $\mathcal{O} = \{s_0, ..., s_{n-1}, G, P\}$ (where G is the *carry-generate* signal, and P is the *carry-propagate* signal), the equation is

$$c_{\text{in}} + \sum_{i=0}^{n-1} 2^i(x_i + y_i) = 2^n \cdot (G + Pc_{\text{in}}) + \sum_{i=0}^{n-1} 2^i s_i\,,$$

which is no longer linear. In this case we can proceed as above expressing the output variables as polynomials in the input variables; the difference is that, when applying Gaussian elimination, the criterium to decide which monomials have to be eliminated has to be different. In the example of the full adder our goal was to keep only linear terms; now we have to keep linear terms and also quadratic terms involving the product of an input variable and an output variable. An area of ongoing work is the study of adequate heuristic(s) for selecting the monomials to be eliminated.

Further, for carry-lookahead adders the output equations are not enough to get the addition equation. The reason is that not all logical consequences of the output equations can be expressed as linear combinations of them. Formally, the vector subspace generated by the output equations misses the algebraic relationships between the different monomials; the *ideal* I [2] generated by the output equations should be considered instead. However, the complexity of the algorithms on ideals would make the approach unfeasible. Thus, it is necessary to work with approximations of I, which may be obtained by multiplying the output equations by monomials. In the example, we need to multiply the equation defining P by the input variable c_{in}. So heuristics have to be employed again to compute the final set of equations from the initial output equations.

Though the complexity of Gaussian elimination is polynomial, the need of adding new equations makes the approach unfeasible for a big number of variables. A solution is to decompose the circuit into black-boxes inductively: once the behaviour of a part of the circuit has been described by addition equations, a bigger part containing the previous one is considered; local signals, i.e. those which are not input or output signals, are eliminated by Gaussian elimination.

The following example illustrates the idea. Consider the carry-ripple adder of 4 bits in Fig. 2, with input signals x_i, y_i, c_0 and output signals s_i, c_4 $(i = 0, 1, 2, 3)$.

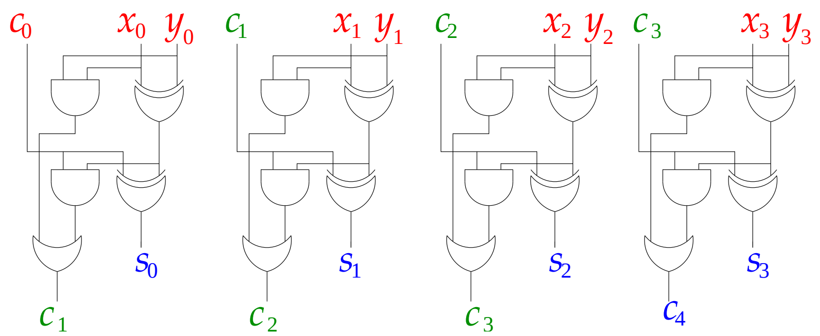

Fig. 2. Carry-ripple adder

Applying the approach proposed above to each of the four modules, we get the equations $c_i + x_i + y_i = 2c_{i+1} + s_i$ $(i = 0, 1, 2, 3)$, which describe the behaviour of the modules as black boxes. By elimination of c_1, c_2 and c_3 we finally obtain:

$$c_{\text{in}} + \sum_{i=0}^{3} 2^i (x_i + y_i) = 2^4 \cdot c_{\text{out}} + \sum_{i=0}^{3} 2^i s_i \, .$$

The proposed method is automatable. We believe it can be extended to more complex arithmetic units and integrated into a verification system. An implementation of the method is in progress.

References

1. M. Blum and H. Wasserman. Reflections on the Pentium Division Bug. *IEEE Transactions on Computers*, 45(4):385–393, 1996.
2. D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties and Algorithms. An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer, 1998.