Property Verification for Arithmetic Logical Circuit by using Proof Checker

A-30

Katsumi Wasaki† Shin'nosuke Yamaguchi† Grzegorz Bancerek‡

1. Introduction

In this paper, we describe the calculation models of a logical arithmetic unit based on the many-sorted algebra and verify the circuit's design by using a proof checker [1][2]. The stability of circuits (full adder/subtracter and 2's complementer examples) is proved based on the definitions and theorems about the logic operator, the hardware gates and the connection of signal lines. The insistence of this research is to establish the mathematical principle technique concerning calculation circuits with high reliability. All definitions and theorems in follow are already formalized and proved by a proof checker "MIZAR" developed by a research project on University of Bialystok, Poland [1].

2. Formalization of the Circuit Structure

The formal definition of a circuit structure is based on the concept of synthesizing many sorted signatures (MSS) [2]. We present the necessary definitions below beginning with the simple circuit structure (1GateCircStr) [3][4].

Definition 2.1. The quadruple S = (c, o, a, r) is said to be a many sorted signature when c is a non-empty set, o is a set, a is a mapping o→c*, and r is a mapping o→c. We call c the carrier of S, o the OperationSymbols of S, a the Arity of S, and r the ResultSort of S, respectively.

Definition 2.2. Let S be a many-sorted signature. S is called void if OperationSymbols of S is empty.

Definition 2.3. Let S be a non-void many-sorted signature. Let o be an element of OperationSymbols of S. Arity(o) is defined as the function (the arity of S)(o), and the result sort of o is defined as the function (the result sort of S)(o).

The above concepts are used to define a one-gate circuit structure, which has only a single item of OperationSymbols below.

Definition 2.4. Let f be a set. Let p be a finite sequence of a set. Let S be a non-void many-sorted signature. 1GateCircStr(p, f) is a structure defined on S, where:
The carrier of 1GateCircStr(p, f) = rng(p) ∪ { 〈 p , f 〉 },
the operation symbols of 1GateCircStr(p, f) = { 〈 p , f 〉 },
(the arity of 1GateCircStr(p, f))(〈 p , f 〉) = p , and
(the result sort of 1GateCircStr(p, f))(〈 p , f 〉) = 〈 p , f 〉 .

We can define the function of InputVertices and Inner Vertices to describe the status of a many-sorted signature. The function of InputVertices/InnerVertices is called a Relation, if there exist the sets y, z such that x = [y, z] where x is a pair set of it. Otherwise, if no such pair set of it exists, it is called without pair.

Definition 2.5. Let G be a non-void many-sorted signature. InputVertices and InnerVertices generate the following subsets of the carrier of G: InputVertices (G) = (The carrier of G) \ rng(the result sort of G) , InnerVertices (G) = rng(the result sort of G) .

Theorem 2.1. Let f be a set. Let p be a finite sequence of a set. Let S be a non-void many sorted signature. The following propositions are true if 1GateCircStr(p, f) is a one-gate circuit defined on S.
(∀ f)(∀ p)(InputVertices (1GateCircStr(p, f)) = rng(p)) ,
(∀ f)(∀ p)(InnerVertices (1GateCircStr(p, f)) = { 〈 p , f 〉 }) .

Theorem 2.2. Let f be a set. Let p be a finite sequence of a set. Let S be a non-void many-sorted signature. The following propositions are true if 1GateCircStr(p, f) is a one-gate circuit defined on S.
(∀ f)(∀ p)(InputVertices (1GateCircStr(p, f)) is without pair) ,
(∀ f)(∀ p)(InnerVertices (1GateCircStr(p, f)) is Relation) .

Now, by defining the union of many sorted sets, we formulate the complex circuit as a combined many sorted signature [5].

Definition 2.6. Let S1, S2 be non empty many sorted signatures. The functor S1+• S2 yields a strict non empty many sorted signature and is defined by the following conditions.
The carrier of S1+• S2 = (the carrier of S1) ∪ (the carrier of S2) , the operation symbols of S1+• S2 = (the operation symbols of S1) ∪ (the operation symbols of S2) , the arity of S1+• S2 = (the arity of S1) +• (the arity of S2), and the result sort of S1+• S2 = (the result sort of S1) +• (the result sort of S2) .

Theorem 2.3. Let S1, S2 be non empty many sorted signatures such that S1 S2 . The following propositions are true:
(∀ S1)(∀ S2)(InnerVertices (S1+• S2) = InnerVertices (S1) ∪ InnerVertices (S2)) ,
(∀ S1)(∀ S2)(InputVertices (S1+• S2) ⊆ InputVertices (S1) ∪ InputVertices (S2)) .

Theorem 2.4. Let S1, S2 be non empty many sorted signatures such that S1 S2, InputVertices (S1) and InputVertices (S2) are without pairs. The following proposition is true:
(∀ S1)(∀ S2)(InputVertices (S1+• S2) is without pair) .

Theorem 2.5. Let S1, S2 be non empty many sorted signatures such that S1 S2, InnerVertices (S1) and InnerVertices (S2) are Relations. The following proposition is true :
(∀ S1)(∀ S2)(InnerVertices (S1+• S2) is Relation) .

Theorem 2.6. Let S1, S2 be non empty many sorted signatures such that S1 S2, InputVertices (S1) and InputVertices (S2) are without pairs, InnerVertices (S1) and InnerVertices (S2) are Relations. The following proposition is true:

† Faculty of Engineering, Shinshu University, Nagano, Japan
‡ Institute of Computer Science, University of Bialystok, Poland

(∀S1)(∀S2)(InputVertices (S1+• S2) = InputVertices (S1) ∪ InputVertices (S2)) .

We conclude this section by compiling the concepts of the structural stability of a complex circuit in the following theorem. Let us define the functor "Following" to describe the state of signals in a circuit after computation. Now, let s be a state of a non empty circuit of non empty many-sorted signatures, the functor of Following(s) is a state of a non empty circuit which means that for every v being a Vertex of non empty many sorted signatures holds Following(s).u = s.u when u is an element of InputVertices, and Following(s).v = ((the Charact of non empty circuit of non empty many sorted signatures). action_at v) . (action_at v depends_on_in s)).v if v when an element of InnerVertices (e.g., see [5]) .

Theorem 2.7. We adopt the following rules: x, y, z, c are sets and f is a function from Boolean 2 into Boolean. We now state four propositions :

(i) Let X be a finite non empty set, f be a function from X^2 into X, and s be a state of 1GateCircuit(〈 x, y 〉 , f). Then, (Following(s))(〈 〈 x, y 〉 , f 〉) = f (〈 s(x), s(y) 〉) , (Following(s))(x) = s(x) and (Following(s))(y) = s(y) .

(ii) Let X be a finite non empty set, f be a function from X^2 into X, and s be a state of 1GateCircuit(〈 x , y 〉 , f). Then, Following(s) is stable.

(iii) For every state s of 1GateCircuit(〈 x , y 〉 , f) holds (Following(s))(〈 〈 x , y 〉 , f 〉) = f (〈 s(x), s(y) 〉) , and (Following(s))(x) = s(x) and (Following(s))(y) = s(y) .

(iv) For every state s of 1GateCircuit(〈 x , y 〉 , f) holds Following(s) is stable.

3. Property Verification Example

In this section, we introduce the mathematical definition and proof of the stability of the circuit referred to as the "full-subtracter" circuit. We can reduce this circuit to the full-adder circuit by taking the complements of the inputs. By using various mathematical concepts on the circuit explained in sec.2, we define an arithmetic unit (see Fig. 1).

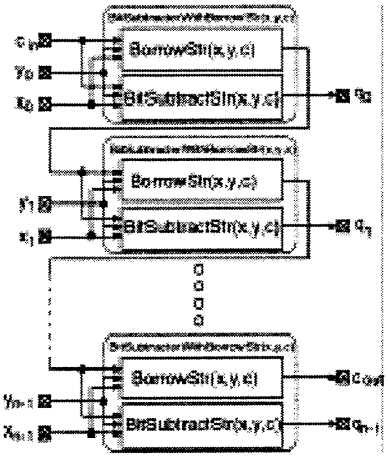

Figure 1. BitSubtracterWithBorrowStr configuration.

To prove the stability of this circuit (Theorem 4.4), we already show the proof of the definitions, theorems and its lemma (Definition 4.1-4.3, Theorem 4.1-4.3 and Lemma 4.1). For more details, see [1]-[5]. All definitions and theorems to prove the stability are already formalized and proved by "MIZAR" proof checking system.

Theorem 4.4. For all non pair sets x, y, c and for every state s of BitSubtracterWithBorrowCirc(x, y, c) holds Following(s, 2) is stable.

[Proof of Theorem 4.4] Let x, y, c be non pair sets. Let S be non empty many sorted signatures of BitSubtracterWithBorrowStr(x, y, c) and s be a state of S. Suppose s1 = Following(s,2), and s2 = Following(Following(s,2)). Under this assumption, we may prove the relation s1=s2. [A] the carrier of S is dom(s1) and dom(s2) by [3](Th4). [B] the carrier of S=the carrier of BitSubtractStr(x,y,c) ∪ the carrier of BorrowStr(x,y,c) by Def2.6. Consider a the carrier of S, then a { x, y, c } or a { [〈 x , y 〉 , xor2] , [〈 [〈 x , y 〉 , xor2] , c 〉 , xor2] } or a { [〈 x , y 〉 , and2a] , [〈 y , c 〉 , and2] , [〈 x , c 〉 , and2a] } or a { [〈 [〈 x , y 〉 , and2a] , 〈 y , c 〉 , and2 〉 , 〈 x , c 〉 , and2a] }, or3 }} by [B],[BOOLE](Th8). Then, [C] a = x or a = y or a = z or a = [〈 x , y 〉 , xor2] or a = [〈 [〈 x , y 〉 , xor2] , c 〉 , xor2] or a = [〈 x , y 〉 , and2a] or a = [〈 y , c 〉 , and2] or a = [〈 x , c 〉 , and2a] or or a = [〈 [〈 x , y 〉 , and2a] , 〈 y , c 〉 , and2 〉 , 〈 x , c 〉 , and2a]] , or3] by [ENUMSET](Th8).

On the input signal lines {x, y, c}, we now state the proposition [D] s2(x)=s1(x), s2(y)=s1(y), s2(c)=s1(c) and, s1(x)=s(x), s1(y)=s(y), s1(c)=s(c) by Th2.7(iii). Next, [E] s1([〈 [〈 x , y 〉 , xor2] , c 〉 , xor2]) = s1 (BitSubtractOutput(x, y, c)) by Def4.1, Th2.7(iii). Then, s1([〈 [〈 x , y 〉 , xor2] , c 〉 , xor2]) = (s1 (x) ⊕ s1 (y)) ⊕ s1 (c) by Lemma 4.1. We state s2([〈 [〈 x , y 〉 , xor2] , c 〉 , xor2]) = (s1 (x) ⊕ s1 (y)) ⊕ s1 (c) by the application of Following repeatedly.

Similarly, [F] s1([〈 [〈 x , y 〉 , and2a] , 〈 y , c 〉 , and2] , 〈 x , c 〉 , and2a] 〉 , or3]) = s1 (BorrowOutput(x, y, c)) by Def4.2, Th2.7(iii). Then, s1([〈 [〈 x , y 〉 , and2a] , 〈 y , c 〉 , and2] , 〈 x , c 〉 , and2a] 〉 , or3]) = (s1 (x) ∧ s1 (y)) ∨ (s1 (y) ∧ s1 (c)) ∨ (s1 (x) ∧ s1 (c)) by Lemma 4.1. We state s2([〈 [〈 x , y 〉 , and2a] , 〈 y , c 〉 , and2] , 〈 x , c 〉 , and2a] 〉 , or3]) = (s1 (x) ∧ s1 (y)) ∨ (s1 (y) ∧ s1 (c)) ∨ (s1 (x) ∧ s1 (c)) .

Therefore, s2(a) = s1(a) for all a the carrier of S by [C]~[F]. Hence, s2 = s1 by [A], [FUNCT_1](Th9). [Q.E.D.]

References

[1] Mizar project homepage: http://www.mizar.org/

[2] A.Trybulec, Many-sorted Algebras, Formalized Mathematics, 4, (1), 14-18, 1995.

[3] Y.Nakamura, P.Rudnicki, A.Trybulec and P.N.Kawamoto, Preliminaries to Circuits (I), Formalized Mathematics, 5, (2), 167-172, 1996.

[4] Y.Nakamura, P.Rudnicki, A.Trybulec and P.N.Kawamoto, Preliminaries to Circuits (II), Formalized Mathematics, 5, (2), 215-220, 1996.

[5] Y.Nakamura and G.Bancerek, Combining of Circuits, Formalized Mathematics, 5, (2), 283-295, 1996.