International Journal of Innovative Computing, Information and Control
Volume **12**, Number **3**, June **2016**

ICIC International ©2016 ISSN 1349-4198
pp. 761–778

THEOREM-PROVING VERIFICATION FOR ASYNCHRONOUS CIRCUITS

Shunji Nishimura, Motoki Amagasaki, Morihiro Kuga
Masahiro Iida and Toshinori Sueyoshi

Graduate School of Science and Technology
Kumamoto University
2-39-1, Kurokami, Chuo-ku, Kumamoto 860-8555, Japan
nishimura@arch.cs.kumamoto-u.ac.jp
{ amagasaki; kuga; iida; sueyoshi }@cs.kumamoto-u.ac.jp

Received November 2015; revised March 2016

Abstract. *This paper presents a formal verification method for asynchronous circuits. Previously, formal verification methods for asynchronous circuits were based on state exploration, and have weaknesses of "state explosion problem". When a state explosion occurs, the verification result is not provided. In contrast, our method is based on theorem-proving which guarantees correctness of a circuit by given formal proofs. Those formal proofs require consistency; that is undertaken by a proof-checking algorithm. This algorithm will complete in linear time relative to the size of a proof. Thus, proof-checking will terminate in any case, and the verification result is always available on our method.*
Keywords: Formal methods, Proof theory, Modal logic, State explosion

1. Introduction.

Asynchronous circuits demonstrate several advantages over synchronous circuits as follows. First, their performance speeds become faster. The speed of a synchronous circuit is determined by the worst delay between flip-flops among the circuit; thus some elements in the circuit have to wait for completions of the others. Meanwhile, elements do not need to wait for the others in an asynchronous circuit as the name suggests. Calculations by asynchronous circuits can conclude faster than those by synchronous circuits. Secondly, their power consumption becomes lower. In a synchronous circuit, flip-flops operate at every clock edges, and each time they latch the data signals even if the signals are unchanged. On the contrary, an element operates only when its input signals change in an asynchronous circuit. Therefore, in general, asynchronous circuits consume less than synchronous circuits. In addition, five more advantages are pointed out in Chapter 15 of [1].

In spite of the advantages above, asynchronous circuits are currently recessive compared to synchronous circuits in the industry. One of the central reasons is a difficulty in an early stage of a design process. In a synchronous circuit, a designer just determines logical functions between flip-flops. In contrast, a designer of asynchronous circuits has to consider not only logical functions but also the operation order of the functions. This brings the difficulty on early designs of asynchronous circuit; thus the designs have been only allowed to the experts who can manage both functions and their order. One proposed solution to this problem is to employ formal verification methods to check demanded properties of asynchronous circuits. A formal verification reduces the designers' burden because they ease correctness checking of a circuit. Even if a designer has no confidence in the design, a formal verification guarantees correctness instead. For this purpose, a number of formal verification methods have been proposed, but all of them have a defect of "state explosion problem" as follows. The methods are based on state exploration, a kind of enumerate method; thus they might give up reaching the result when all computational resources were spent. In other words, if the size of a design under verification (DUV) is relatively larger than power of computational resources for verification tools, a designer cannot get the verification result. This problem prevents taking advantage of asynchronous circuits; thus we need a scale-robust method, which means ready for variously-sized DUV, of formal verification to conquer the problem. With a scale-robust verification method, it compensates the difficulty of asynchronous circuit designing, and then we will gain efficiency of performance speed and power consumption as mentioned above.

This paper presents a verification method, or a property checking method, for asynchronous circuits, and the method enables formal verification regardless of DUV size and power of computational resources. The foundation of our method is *theorem-proving*, in which a proof certifies correctness of a DUV. A proof had to be given by hand, and the size of a proof is linear to the size of a DUV. Then proof-checking algorithm will complete in linear time relative to the size of a proof. This leads the algorithm is scale-robust, and therefore the verification result is always available in a theorem-proving manner, in contrast to conventional methods having the problem of state explosion. It became possible to apply theorem-proving to asynchronous circuit verification by reconstructing a circuit model in *modal logic* [2], a kind of logical system. Modal logic involves temporal logic, which is well known as a basis theory of assertion languages for synchronous circuits, and modal logic is described by using *possible worlds* and relations between the worlds. From this point of view, we regard a signal at a time point as a world, and the relations between the worlds represent abstract passages of time; this makes available to verify circuits even if actual delay values have not been determined.

The rest of the paper is organized as follows. Section 2 presents related works. Section 3 provides theoretical preliminaries, with which the basis theory of our verification method is built in Section 4. Section 5 expresses discussions, which includes an issue of validation of the theory. Section 6 demonstrates a verification example on our method: the DUV is an asynchronous FIFO (First-In, First-Out) circuit, as a small but practical one. Although we have not built a whole verification tool of the method, an experimental implementation on a theorem proving language is demonstrated in Section 7. The conclusions are provided in Section 8.

2. Related Works.

Asynchronous circuits are often classified according to conditions in the delay; for example, *delay-insensitive* circuits are independent of the delay of wires or gates. However, we do not assume any conditions on the subjects in this paper. For this kind of non-specific class circuits, a number of formal verification methods were proposed: some representative examples of them are for *Timed Petri Nets* [3, 4], for *Transition Systems* [5], and for *Process Spaces* [6]. All of them regard circuits in a perspective of state representations, and an exploration of the state space corresponds to a verification. State spaces grow exponentially with the increase of the DUV size; thus these methods are obliged to fail when the computation environment lacks resources. The problem is known as "state explosion". In order to conquer the difficulty of asynchronous design, it is required to obtain formal verification methods without the state explosion problem.

Theorem-proving has been applied to circuit verifications since the 1980s [7, 8, 9, 10]. These studies are only for synchronous circuits because discrete time is able to manipulate easily in a formal manner. In contrast, asynchronous circuits essentially require real number as background time, and real number is hard to treat in a rigorous manner; it is well known that real number is represented by approximated values in almost all computer systems. This paper introduces a logical system to represent abstract time, a substitute for real number, with which a behavior of an asynchronous circuit is able to be described. The logical system, named *modal circuit logic*, is provided in Section 4.

3. Preliminaries.

In order to define our logical system in the next section, this section introduces two notions: *categories* from category theory [11], and *frames* for modal logic [2]. Then we also define graph generated categories, which will play an important role in our theory.

Definition 3.1. *(Categories)*

A category (O, M) consists of a set of objects O[1]*, a set of morphisms M, and implicit four functions: the first and second function assign to each $f \in M$ its domain $dom\, f \in O$ and its codomain $cod\, f \in O$, respectively. If $dom\, f = a$, $cod\, f = b$, then we write*

$$f : a \to b \quad or \quad a \xrightarrow{f} b$$

to indicate morphism f. The third function assigns to each $a \in O$ a morphism $id_a \in M$ called the identity morphism of a, and the last function makes a composition with assigning to any f and g of morphisms such that $cod\, f = dom\, g$, $f \cdot g : domf \to cod\, g$ (notice order of that notation). These functions are required to satisfy the following axioms:

$$id_a \cdot f = f, \quad f \cdot id_a = f$$
$$(f \cdot g) \cdot h = f \cdot (g \cdot h).$$

Definition 3.2. *(Frames)*

A pair $(W, \{R_i\}_{i \in I})$, here I is an index set, of following is called a frame.

- *W is a non-empty set of possible worlds.*
- *Each R_i is a binary relation on W, called accessibility relation.*

Such a frame $(W, \{R_i\}_{i \in I})$ introduces a multi-modal logic: based on W and modal operators are determined by $\{R_i\}_{i \in I}$ [2].

A netlist circuit, that combines components and their connectivity information, is also considered as a directed graph: a vertex/arc of the graph is a component/signal respectively. However, we point out another interpretation of circuits into directed graph: a vertex/arc is regarded as a signal/component respectively, by dividing a component into the number of its paths. We call the result of this interpretation “signal-vertex graph”. Figure 1 illustrates two kinds of graph description: in the signal-vertex graph, AND_{a-c} denotes the path on AND from a to c. We employ signal-vertex graphs for our purpose, and we can assume that graphs have connectivity without loss of generality.

We consider a given circuit as signal-vertex graph $G = (V, A)$, a pair of vertex set and arc set. The relation between V and A is represented by domain and codomain function: for $a \in A$, $dom\, a$ gives its domain vertex, and $cod\, a$ also gives its codomain vertex.

Definition 3.3. *(Generated category)*

For a signal-vertex graph $G = (V, A)$, corresponding free category (O, M) is generated by the recursive definition below.

1. *A vertex $v \in V$ is also a member of O.*
2. *An arc $a \in A$ is also a member of M.*
3. *For $a_0, a_1 \in M$, a composition $a_0 \cdot a_1 \in M$ is defined when $cod\, a_0 = dom\, a_1$. Then the domain/codomain of $a_0 \cdot a_1$ is defined as $dom\, a_0/cod\, a_1$ respectively, and the composition operator follows the condition:*

$$(f \cdot g) \cdot h = f \cdot (g \cdot h).$$

[1] A category in this paper is equivalent to the *small category* in [11].

4. *For each $v \in O$, an identity morphism id_v exists in M, and follows the conditions:*

$$id_v \cdot a = a, \quad a \cdot id_v = a \quad (a \in M).$$

5. *For each $a \in M$, an inverse morphism a^{-1} is also included in M, defining $dom\, a^{-1} := cod\, a$ and $cod\, a^{-1} := dom\, a$, and follows the conditions:*

$$a \cdot a^{-1} = id_{cod\, a}, \quad a^{-1} \cdot a = id_{dom\, a}.$$

The last clause is not usually included in commonly-used definition of free category [11].

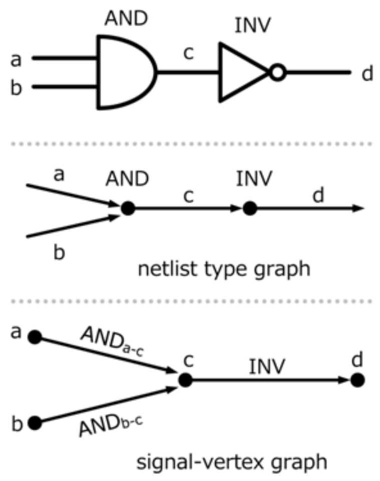

FIGURE 1. Graph interpretations

4. **Modal Circuit Logic.** This section describes a logical system, named *Modal Circuit Logic*, in which we can express circuit behaviors. The definitions of modal circuit logic require the notions of frames and graph generated category which are given in the previous section.

4.1. **Frames for modal circuit logic.** The next definition determines a logical system as a type of modal logic.

Definition 4.1. *(Frames for modal circuit logic)*

For signal-vertex graph (V, A), its generated category (O, M) and an object $o \in O$, we build a frame $(W, \{R_i\}_{i \in I})$ as follows.

- *Let possible worlds W be a set of morphisms $\{f \in M | dom\, f = o\}$.*
- *Let index set I be a set of Arcs A.*
- *For $i \in I$, let accessibility relation $R_i := \{(w, w \cdot i) | w \in W,\ cod\, w = dom\, i\}$.*

We call this "frame at o".

Let us take a look at two examples of frames.

Example 4.1. *(Graph, generated category and frame)*

An example of concepts above is given in Figure 2. At the top of the figure, a circuit C_0 is shown, and the next $Graph(C_0)$ denotes the signal-vertex graph of C_0. $Graph(C_0)$ has arcs of ${}^*AND$, ${}_*AND$ and INV: ${}^*AND$ denotes the upside path on AND from a to c and also ${}_*AND$ denotes the downside path from b to c. Then free category $Cat(C_0)$ is generated as at the middle of the figure: each object a, b, c and d has identity morphism respectively, and there are compositions and inverse morphisms of original arcs. The frames determined by circuit C_0 are at the bottom of the figure, displayed as $Frame(C_0, x)$

for each frame at x (where x is a signal on C_0): the nodes correspond possible worlds and the arrows correspond accessibility relation. In detail, on $Frame(C_0, a)$ in particular, from world id_a, it is accessible to world $^*AND$ by relation $R_{^*AND}$ because $R_{^*AND}$ includes $(id_a, {}^*AND)$ by its definition, and so on. However, every $Frame(C_0, x)$ of four conforms to $Graph(C_0)$; thus no additional structures are found by category generation and frame formulation. To conclude, this is a trivial example from our perspective.

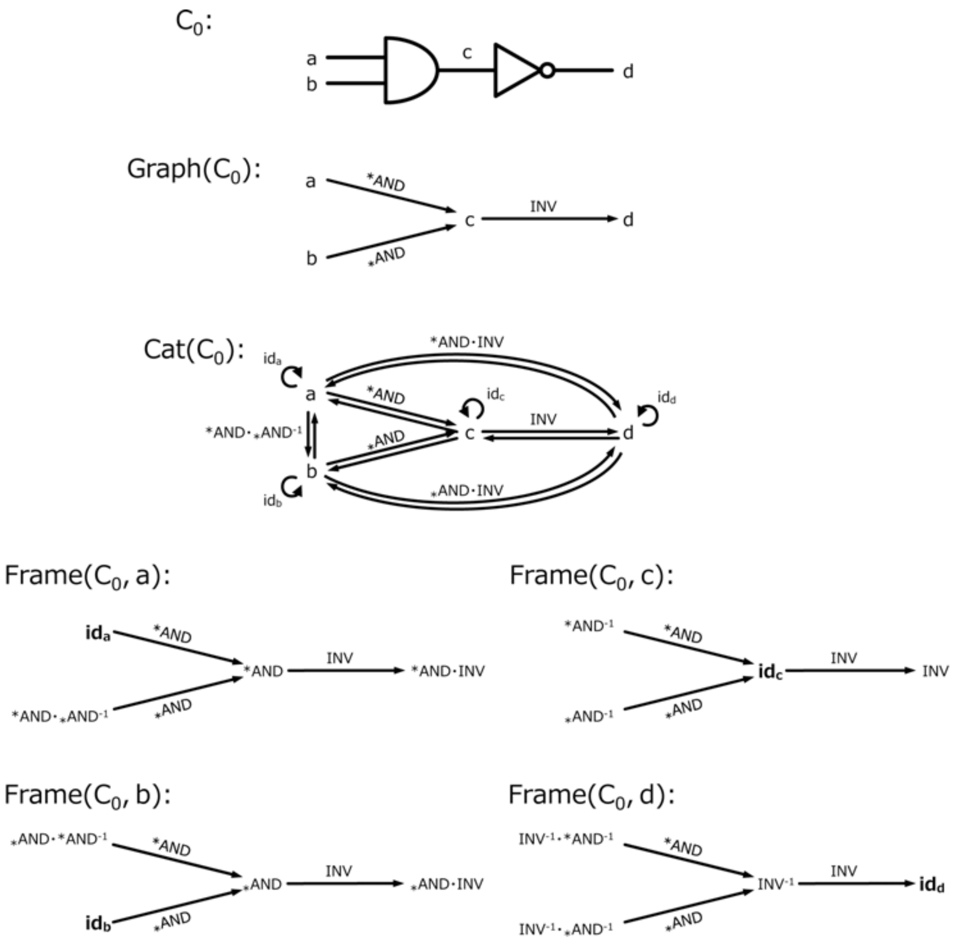

FIGURE 2. Example of graph, generated category and frame

We have non-trivial example below.

Example 4.2. *(Non-trivial frame)*

The second example of graph and generated category is given in Figure 3. Circuit C_1 and $Graph(C_1)$ are illustrated in a similar manner to Example 4.1. Free category $Cat(C_1)$ is generated as the middle of the figure: each object a, b, c has identity morphism respectively, and between any two objects, there are infinite morphisms. Focusing on a to c in particular, the lower part of the figure shows some of basic morphisms.

The determined frame $Frame(C_1, a)$ is shown as in Figure 4. From world id_a, it is accessible to world INV by relation R_{INV}, and so on. As a whole, $Frame(C_1, a)$ has additional structure when it is compared to $Graph(C_1)$. We will apply such a frame to analyzing circuit properties.

4.2. **Inference rules.** To investigate a behavior of circuits, we have to consider propositions on a signal and their satisfiability.

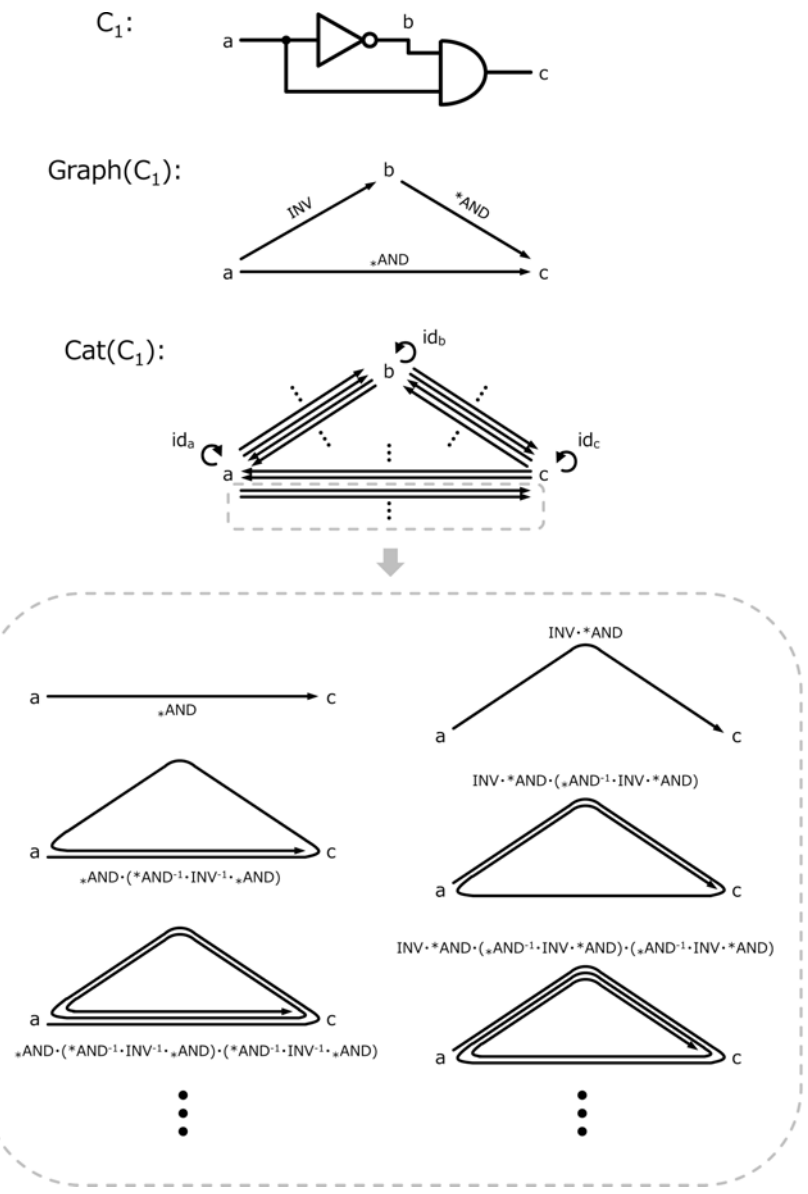
C1:
a
b
c
Graph(C1):
INV
*AND
⁎AND
Cat(C1):
idb
ida
idc
⁎AND
INV·*AND
⁎AND·(*AND-1·INV-1·⁎AND)
INV·*AND·(⁎AND-1·INV·*AND)
⁎AND·(*AND-1·INV-1·⁎AND)·(*AND-1·INV-1·⁎AND)
INV·*AND·(⁎AND-1·INV·*AND)·(⁎AND-1·INV·*AND)

FIGURE 3. Second example of graph and generated category

Definition 4.2. *(Satisfaction)*
For given frame $F = (W, \{R_i\}_{i\in I})$, world w in W and proposition P at w, we introduce following denotation:

$$F,\ w \Vdash P.$$

which means P holds at w under F. Frames might be omitted when it is clear from its contexts.

Back to circuit C_1 in Figure 3. When one considers the circuit under the notion of zero delay, it holds that "if $a = 0$ then $b = 1$ because of a function of INV". Taking this idea into our frame, we could state that "if the signal value is equal to 0 at world id_a, then the signal value is equal to 1 at world INV caused by relation R_{INV}". We denote this as follows:

$$\frac{id_a \Vdash 0}{INV \Vdash 1}.$$

The upper part indicates a premise, and in which 0 means proposition "is equal to 0". The lower part is a conclusion. At this point, we propose a point of view: world INV is

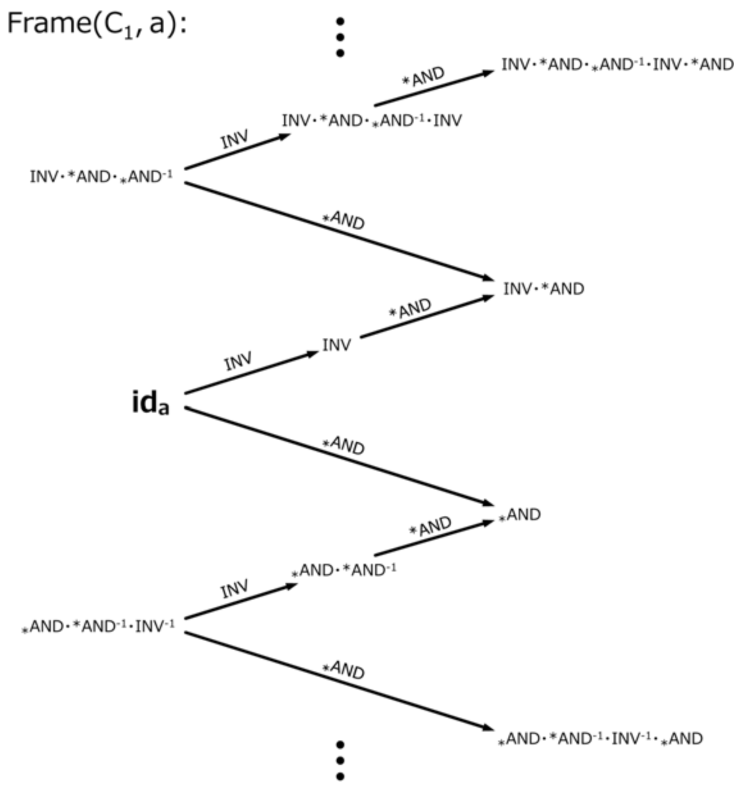

FIGURE 4. Frame of C_1

TABLE 1. Inference rules for INV and AND

$\dfrac{w \Vdash 0}{w \cdot INV \Vdash 1}\ {\scriptstyle INV_0}$	$\dfrac{w \Vdash 0}{w \cdot {}^{*}AND \Vdash 0}\ {\scriptstyle {}^{*}AND}$
$\dfrac{w \Vdash 1}{w \cdot INV \Vdash 0}\ {\scriptstyle INV_1}$	$\dfrac{w \Vdash 0}{w \cdot {}_{*}AND \Vdash 0}\ {\scriptstyle {}_{*}AND}$
	$\dfrac{w \cdot {}^{*}AND^{-1} \Vdash 1 \quad w \cdot {}_{*}AND^{-1} \Vdash 1}{w \Vdash 1}\ {\scriptstyle AND}$

accessible from id_a with R_{INV}, and then in the real circuit, it is regarded as "signal b after passing through INV gate is determined by signal a at the moment". With this idea, all inference rules about C_1 are introduced as shown in Table 1, where at the right of each rule, an identity of the rule is given. Notice that rules are applied only in the worlds; thus an inference rule does not make sense when it does not fit the worlds. For example, rule INV_0 cannot be applied for $w = {}_{*}AND$ because $*AND \cdot INV$ does not make sense.

Example 4.3. *(Property inference)*
Consider circuit C_0 in Figure 2 with condition $a = 0$, following inference is introduced.

$$\cfrac{\cfrac{id_a \Vdash 0}{{}^{*}AND \Vdash 0}\ {\scriptstyle {}^{*}AND}}{{}^{*}AND \cdot INV \Vdash 1}\ {\scriptstyle INV_0}$$

The last world ${}^{*}AND \cdot INV$ represents signal d, and thus this concludes "d becomes 1". It is helpful to illustrate such an inference in a diagram as shown in Figure 5. The vertexes denote properties and arcs denote inference rules.

4.3. **Connection to an environment.** To express properties on a circuit, we need another worlds as an environment of it.

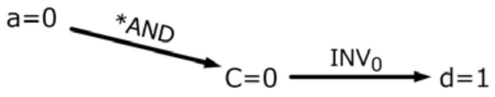

FIGURE 5. Property inference on C_0

Definition 4.3. *(Environments and connections)*

An environment is a partially ordered set (E, R): a pair of a set E and partial order R. For given frame $(W, \{R_i\}_{i\in I})$ at $w_0 \in W$ and such an environment (E, R), corresponding connected frame is defined as follows:

- *Let possible worlds be a Cartesian product $E \times W$,*
- *Let accessibility relations be a direct sum of R and $\{R_i\}_{i\in I}$.*

In symbolic expression, the connected frame is $(E \times W, R \oplus \{R_i\}_{i\in I})$.

If a frame is at w, the connecting point between an environment and the circuit becomes w. Figure 6 illustrates an example of an environment and connection. Consider natural numbers $\mathbb{N}$ and its normal order as an environment, and the frame of C_0 at a, then the connected frame is shown on the right side. Notice that one cannot connect more than one world because two points connection might result in a contradiction of timing relations.

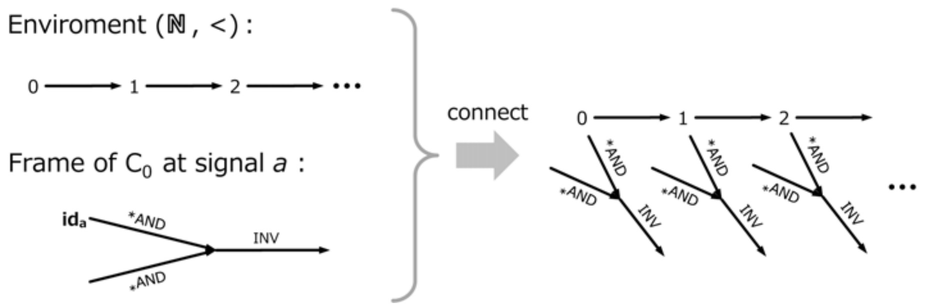

FIGURE 6. Example of environment and connection

4.4. **Additional conditions.** In closing of our definitions, we need two additional conditions. First, generally, arbitrary accessibility relation determines partial order. For our frames, we assume that for each signal a, the set $\{w \in \text{Worlds} | cod\, w = a\}$ has the nature of total order in particular. Henceforth, we call it *locally total order.*

Secondly, for relation R of a frame, we also assume following *linear condition.*

Definition 4.4. *(Linear condition)*

For given frame $(W, \{R_i\}_{i\in I}, R)$ and $i \in I$,

$$a_0 \xrightarrow{R_i} a_1,\ a_0 \xrightarrow{R} b_0,\ b_0 \xrightarrow{R_i} b_1 \ \Rightarrow\ a_1 \xrightarrow{R} b_1,$$
$$a_0 \xrightarrow{R_i} a_1,\ a_1 \xrightarrow{R} b_1,\ b_0 \xrightarrow{R_i} b_1 \ \Rightarrow\ a_0 \xrightarrow{R} b_0.$$

where $a_0,\ a_1,\ b_0,\ b_1 \in W$ and $cod\, a_0 = cod\, b_0 = dom\, i$, $cod\, a_1 = cod\, b_1 = cod\, i$.

Figure 7 illustrates this condition. These conditions, locally total and linearity, make whole relations $(\{R_i\}_{i\in I},\ R)$ imitate a concept of time, as will be illustrated in 5.2.

5. **Discussions.** The previous section provided only the definition of modal circuit logic, and therefore we have to discuss the validity and related issues in this section. There are three subsections here as follows. First, we provide the relation to conventional delay model, which validates modal circuit logic. Secondly, as a related issue, we propose an

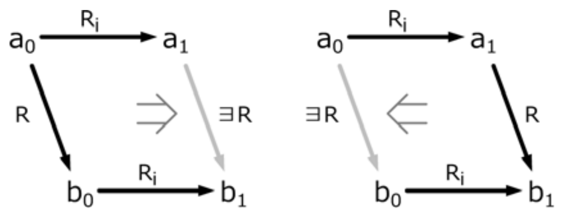

FIGURE 7. Linear condition

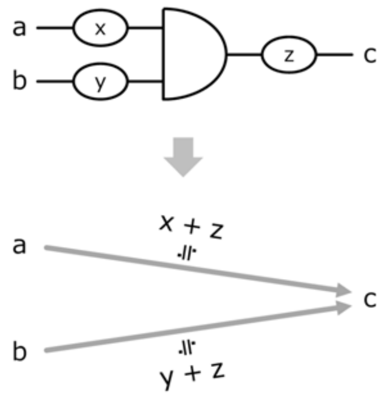

FIGURE 8. Gate- and wire-delay model and accessibility

approach to express timing constraints in modal circuit logic. Finally, computational complexities of our method are also provided as a related issue.

5.1. **Delay model.** Conventional gate- and wire-delay models assume virtual delay elements at each gate and wire (see Chapter three of [1]). On the semantics of our method, both of gate- and wire-delay are included in an accessibility relation as shown in Figure 8. The sum of wire-delay x and gate-delay z are regarded as the accessibility from a to c. Thus our notion is broader than that of gate- and wire-delay model.

5.2. **Representation of timing constraints.** Back to circuit C_1 in Figure 3 again, when "$a = 0$" and followed by "$a = 1$" are given to the input, generally it is unknown whether output c keeps 0 or a short pulse appears. In order to treat that kind of behavior, we introduce *relative timing constraints*. Relative timing constraints have been adopted on abstract circuit model in a metric-free manner such as [12] for *Petri net* and [13, 14] for *process space*. For our method, relative timing constraints are defined as follows.

As we mentioned at Section 4.4, relations $(\{R_i\}_{i\in I}, R)$ of our frame assume locally total ordered and linear. Remarking on $Frame(C_1)$ in Figure 4, there is a relation either $id_a \to w$ or $w \to id_a$, where $w := {}_*AND \cdot {}^*AND^{-1} \cdot INV^{-1}$, since locally total. Resulting from linearity, we obtain either $\{\cdots, w^{-1} \overset{R}{\to} w^0,\ w^0 \overset{R}{\to} w^1, \cdots\}$, or, $\{\cdots, w^{-1} \overset{R}{\leftarrow} w^0,\ w^0 \overset{R}{\leftarrow} w^1, \cdots\}$, where w^0 denotes id_a. Such a coherent order of $\{w^n\}_{n\in\mathbb{Z}}$ is regarded as a relational timing constraint for the signal path a to c of circuit C_1 in Figure 3; $\{\cdots, w^{-1} \overset{R}{\to} w^0, w^0 \overset{R}{\to} w^1, \cdots\}$ indicates that the signal path from a through INV to c is faster than the another wire path, and vice versa.

Example 5.1. *(Analysis along with timing constraint)*

About circuit C_1 in Figure 3, when the signal path from a through INV to c is faster than another one, the behavior is deduced as shown in Figure 9. Here, dotted arrow $P \dashrightarrow w$ denotes "P holds until just before w." In formal, P at $w_0 \dashrightarrow w_1$ implies that for arbitrary x in the world, if there exists $w_0 \to x$ and $x \to w_1$ in the accessibility relations, then P

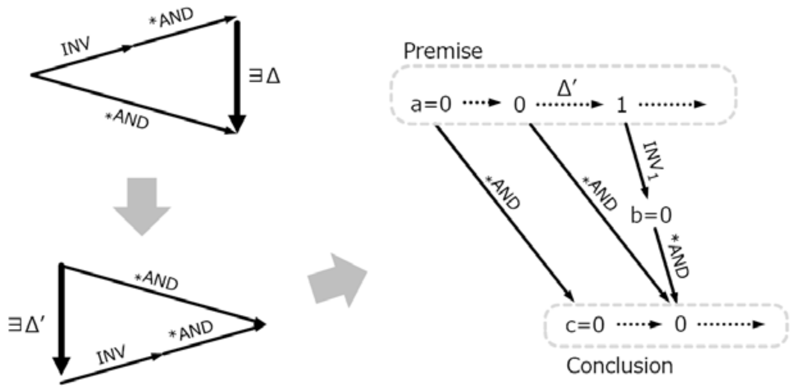

FIGURE 9. C_1 analysis when INV is faster

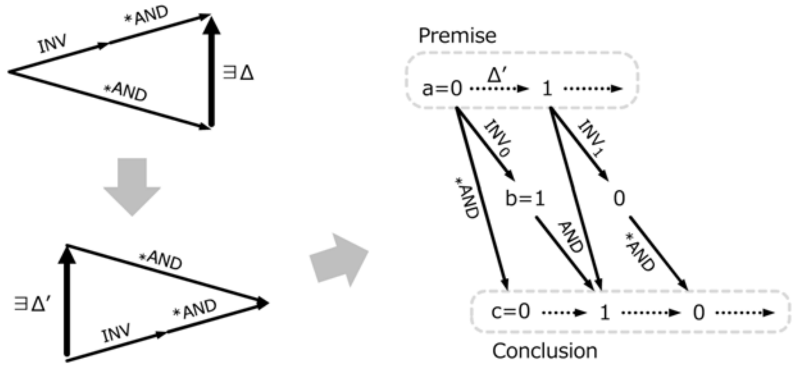

FIGURE 10. C_1 analysis when INV is slower

also holds at x. From the condition, we have Δ at the upper left of the figure, and we also have Δ' at the lower left by linearity. Then the input hypotheses, that indicate a wave changes its value from 0 to 1, lead c is always 0 as the conclusion. Meanwhile, when the signal path through INV is slower than the other, the behavior is deduced as shown in Figure 10. The existence of Δ' is introduced, and the conclusion has a short pulse.

We will describe verification example for a more practical circuit in the next section.

5.3. **Computational complexities.** Since our aim is to avoid state explosion problem, we have to investigate the computational complexity of our method. Assuming a proof has given, the issue is about the complexity of the proof-checking process. For example on circuit C_1 in Figure 3, each step of a given proof has to fit to one of the rules in Table 1. In general, the proof-checking is an iteration of matching check against rules, and that iteration goes through the whole steps of a proof. The computation amount of each matching check depends on a number of rules, a constant for the circuit. Therefore, proof-checking will complete in linear time relative to the size of a proof; in other words, our method has a linear complexity.

6. Verification of Asynchronous FIFO.

This section presents a verification example of asynchronous FIFO (First-In, First-Out) circuit, by using modal circuit logic defined in Section 4. Our target circuit is asynchronous FIFO by Molner et al. [15], demonstrating throughput of 930 million data items per second using 0.6 micron process. In [15], they adopted an analog simulator to verify the circuit, and thus they have not done any exhaustive verifications. As far as timing constraints are concerned, they were built according to an outcome of the simulator in an ad hoc manner. In contrast, using our modal circuit logic, we can verify the FIFO along with some timing constraints without any test patterns. This means an exhaustive verification becomes available.

6.1. **The FIFO under verification.** Molner's asynchronous FIFO is shown in Figure 11. The whole circuit consists of a series connection of the Units; each Unit contains *D-Latch*, *RS-FF* (Reset-Set Flip-Flop) and $ANDN$ (one-side-negated AND gate). Though the FIFO was represented with transistors and gates originally in [15], we adopted a relatively-coarse interpretation by using latch and FF. These are minimum elements of our verification. Signal D indicates data, RQ indicates a request for latching the data, and ACK indicates an acknowledgment of completion of latching.

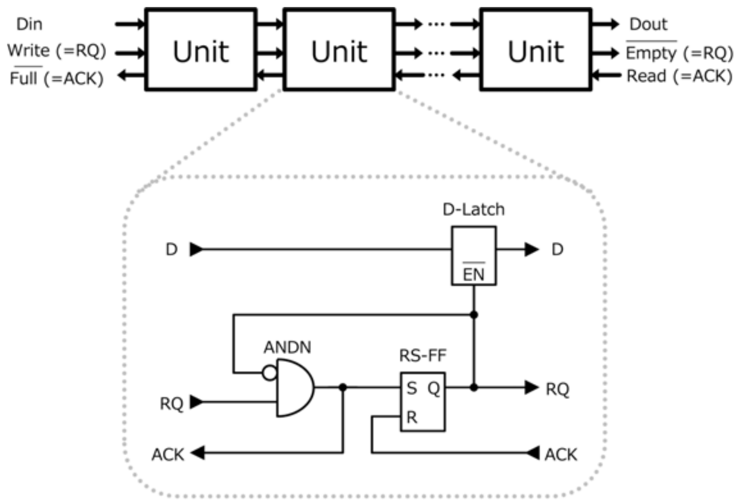

FIGURE 11. Molnar's asynchronous FIFO

Its behavior is described as follows. We assume that all RQs and ACKs are stabilized at 0 at the beginning. It follows that *D-Latch* is transparent. When $RQ = 1$ comes from left-hand unit, $ANDN$ sends $ACK = 1$ to left-hand unit. Meanwhile *RS-FF* is set, then $RQ = 1$ is sent to right-hand unit and *D-Latch* holds the current data. Later, when $ACK = 1$ is reached from right-hand unit, *D-Latch* turns back to transparency. To wrap up, this sequence forms that latching data from the left and passing data to the right.

Next, we should express local properties of each element on modal circuit logic. Properties of $ANDN$ are shown as in Figure 12, where $_L$ and $_R$ denote left- and right-hand side, $^*ANDN$ denotes the path from inverted input to the output, and $_*ANDN$ denotes the other path. For *RS-FF*, properties are shown as in Figure 13. The lowest part represents holding state, in which, reg denotes a feedback-loop that is supposed to included in *RS-FF*, and it works as a value holder. We assume a delay value of reg is sufficiently-smaller than the others. In the case of *D-Latch*, its properties are shown as in Figure 14: the upper part represents transparent state and the lower part represents holding state. Note that we use same name reg for *RS-FF* and *D-Latch*, but they are different; and one can distinguish them by the world of its domain or codomain.

Finally, we will give a whole image of a frame of the FIFO, as in Figure 15. It describes relations on three consecutive units and each dotted line separates them. (We substituted dotted lines for previous $_L/_R$ notation.) These D and RQ have self-referential arrows that denote reg, feedback-loops.

6.2. **Timing constraints required.** This subsection describes timing constraints which is required for proving a desired property. (The "desired property" will be discussed in the next subsection.) These constraints have been built during a course of deductions we

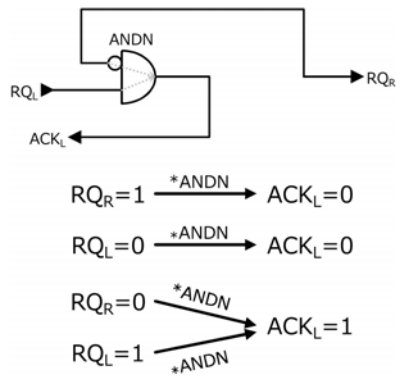

FIGURE 12. Properties of *ANDN*

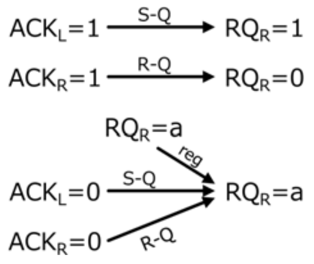

FIGURE 13. Properties of *RS-FF*

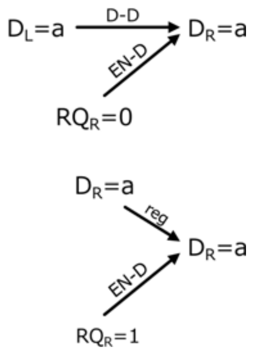

FIGURE 14. Properties of *D-Latch*

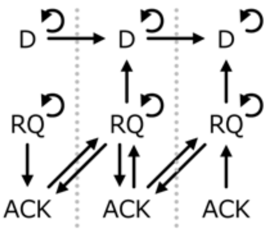

FIGURE 15. Whole image of a frame

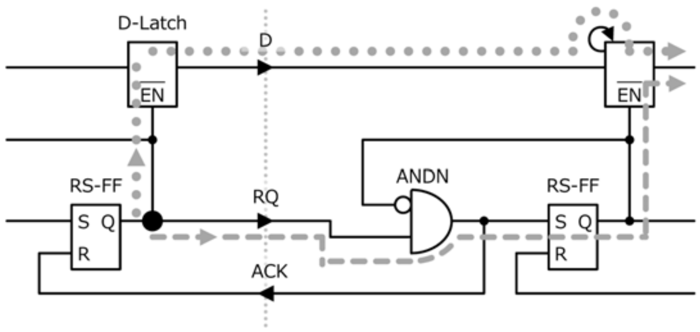

FIGURE 16. Timing constraint A

had; when we felt impossible to reach the target property due to an absence of hypothesis, we suggested that some timing constraints were needed.

Timing constraint A: One of required constraints is described with Figure 16. From the branch point of *RQ*, the dotted path which goes up, then goes through the left *D-Latch*, and finally gets to the right *D-Latch*'s output, must be faster than the dashed line which goes right and also gets to *D-Latch*'s output. This constraint provides that the right *D-Latch* should take data after being delivered valid ones from the left *D-Latch*.

Timing constraint B: The other constraint is described with Figure 17. From the branch point of *ACK*, the dotted path which goes right and gets to the right *D-Latch*'s output, must be faster than the dashed line which goes down, goes through

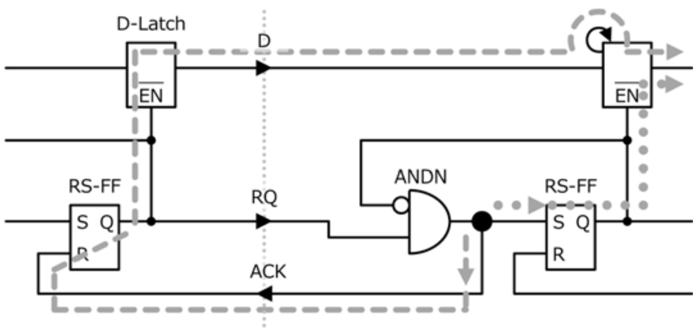

FIGURE 17. Timing constraint B

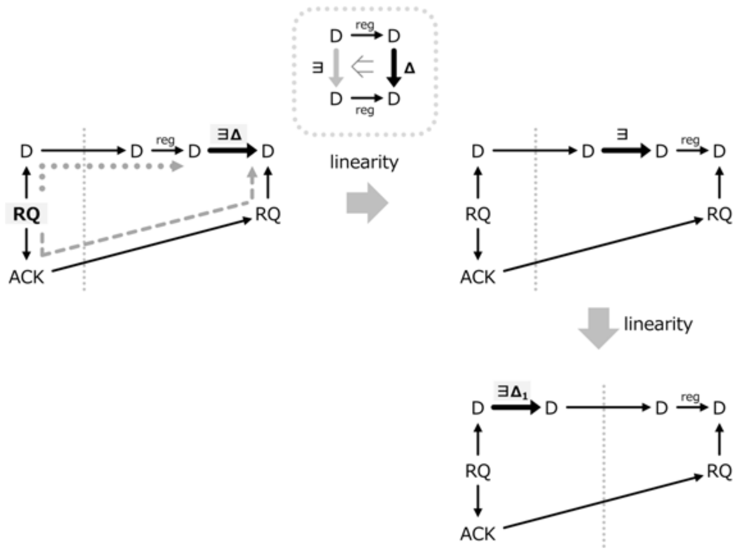

FIGURE 18. Arrangement of constraint A

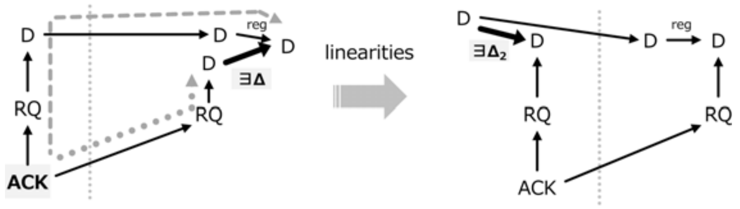

FIGURE 19. Arrangement of constraint B

a few elements, then also gets to the *D-Latch*'s output. It provides that the right *D-Latch* should take data before finishing holding state of the left *D-Latch*.

In order to prove in the next subsection, we need to arrange these constraints. We can redraw constraint A on a frame as in the upper left part of Figure 18. There exists Δ because the clockwise path from RQ is faster than the other. Then, we can transform the existence as from left to right in the figure. As in the upper right of the figure, there also exists an accessibility relation before reg by linear condition we assumed: the top part of the figure explains it in detail. At last, we obtain Δ_1 as in the lower part of Figure 18. We will use Δ_1 in a process of an inference in the next subsection.

Meanwhile, about constraint B, an original expression is shown in the left part of Figure 19. As in the right of the figure, we definitely obtain Δ_2 which is also used for a verification of the FIFO in the forthcoming subsection.

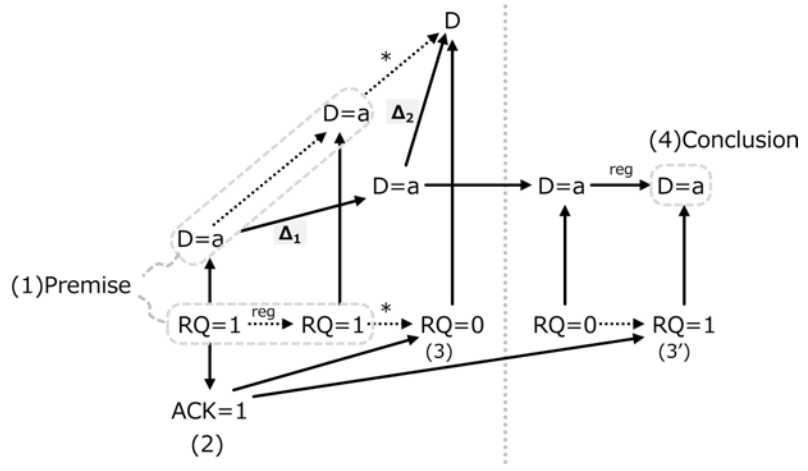

Figure 20. Proof outline

6.3. **Reasoning as a verification.** We describe a verification example on the asynchronous FIFO. In our logical system, modal circuit logic, reasoning corresponds to a verification; and a proof corresponds to an evidence of correctness.

One of required properties for the FIFO is that "under the circumstances of stability, when data and $RQ = 1$ are given, the data will be taken by the *D-Latch* of the next unit after a reasonable period of time". Here "stability" is defined as "$RQ = 0$ and $ACK = 0$ everywhere"; in the condition, all *D-Latch* have transparent states. We will prove the claim as discussed below.

The proof outline is shown as in Figure 20. There are the premise "data and $RQ = 1$ are given" at (1), as $D = a$ and $RQ = 1$ during each appropriate period. $RQ = 1$ leads (2) $ACK = 1$ by the lowest property in Figure 12 and stability assumption. Then it turns (3) RQ back to 0 and activates (3') $RQ = 1$ in the next unit. Where the relations at asterisks(*), their existences come from a shortness hypothesis for reg. Next we use Δ_1 and Δ_2 from the previous subsection, as if pinching the principal $D = a$, which leads up to the (4) conclusion. In fact, it is deduced by the upper and lower property in Figure 14 in sequence. The conclusion is placed over (3') $RQ = 1$, which means *D-Latch* holds and also implicates "the data are taken". Thus the claim is proved along with timing constraints A and B.

7. **Implementation in a Theorem Proving Language.**

This section describes an example implementation of the system of modal circuit logic. In the previous section, we demonstrated a verification and its inferences are developed in a natural language. Since our aim is formal and rigorous verification, any obscurity should be excluded; for example, any leaps in logic should be avoided. Here we propose a solution: building our logical system in theorem proving language, in which inference processes will be confirmed accurately. Our implementation is in *Agda* [16], a theorem proving language and also a functional programming language.

7.1. **Implementation of definitions.** In order to represent circuits and frames, we adopt a concept of *Arrow* [17] as a basis of our implementation. Arrow is a class of functional programming language *Haskell*, or a framework for expressing some sort of abstract calculations. On it, a calculation is expressed in a combination of lesser calculations with the operators shown as in Figure 21. We use these operators to represent circuits and frames.

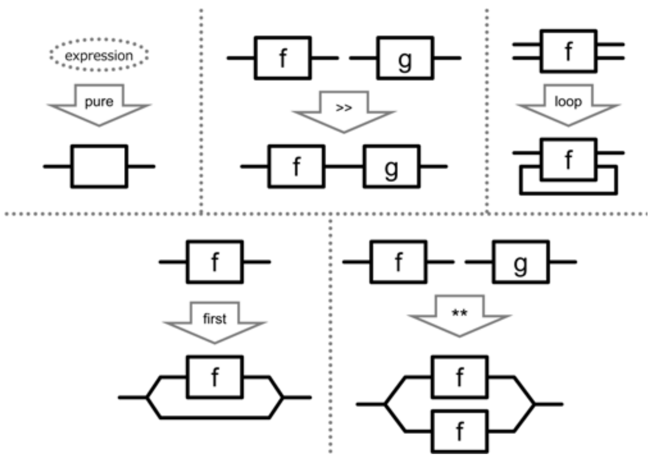

FIGURE 21. Operators in Arrows

In Arrows manner, circuit definition in Agda is as follows.

```
data Arrow where
  pure : (A → B) → Arrow A B
  _>>_ : Arrow A B → Arrow B C
    → Arrow A C
  _**_ : Arrow A₀  B₀  → Arrow A₁  B₁
    → Arrow (A₀ × A₁ ) (B₀ × B₁ )
  first : Arrow A₀  B₀
    → Arrow (A₀ × A₁ ) (B₀ × B₁ )
  loop : Arrow (A × C) (B × C)
    → Arrow A B
```

Next, frames are defined as an extension of `Arrow`: by adding following to

```
id : ExArrow A A
L| : ExArrow (A₀ × A₁ ) B → ExArrow A₀  B
R| : ExArrow (A₀ × A₁ ) B → ExArrow A₀  B
|L : ExArrow A (B₀ × B₁ ) → ExArrow A B₀
|R : ExArrow A (B₀ × B₁ ) → ExArrow A B₀
-  : ExArrow A B → ExArrow B A
```

`ExArrow` represents frames: for example, `ExArrow A B` represents a path, or accessibility relation, from world `A` to world `B`. `id` is a path to itself. When `f : Arrow (A₀ × A₁ ) B`, `L|f` denotes the path from `A₀` to `B`, qualified its input to left-hand. The denotation "`-`" makes it inside out.

In addition, we introduce an equation of `ExArrow` as follows.

```
data _≡_ : {A B : Set} →
 ExArrow A B → ExArrow A B → Set₁  where
  refl : {A B : Set}{f : ExArrow A B} →
     f ≡ f
  symm : {A B : Set}{f g : ExArrow A B} →
    f ≡ g → g ≡ f
  trans : {A B : Set}{f g h : ExArrow A B} →
    f ≡ g → g ≡ h → f ≡ h
  assoc : {A B C D : Set}{f : ExArrow A B}
    {g : ExArrow B C}{h : ExArrow C D} →
     (f >> g) >> h ≡ f >> (g >> h)
   :
```

where expressions enclosed by "`{}`" denotes implicit arguments in Agda. The equation ≡ is defined as in the nature of reflectivity(`refl`), symmetry(`symm`) and transitivity(`trans`). `assoc` determines associativity of `>>`.

In fact, the definition of `ExArrow` contains the opposites of accessibility relations to form worlds of frames, but we have to distinguish them after all as follows.

```
data Positive : {A B : Set} →
    ExArrow A B → Set₁  where
   :
   pure :
    {A B : Set} →
     (f : A → B) →
      Positive (pure f)
   :
```

`pure` indicates that all paths from the circuit under verification should be positive.

Finally, we define `PROP` for express properties, or propositions.

```
data PROP : {A B : Set} →
 Time → ExArrow A B → B → Set₁  where
   :
  pure : {t : Time}{A B C : Set} →
   {f : ExArrow A B}{g : B → C}{b : B} →
    PROP t f b → PROP t (f >> pure g) (g b)
   :
```

`PROP` takes time, accessibility relation and output value, then returns a set. For given `t : Time`, `r : ExArrow A B` and `v : B`, if `PROP t r v` has any elements, it means that proposition `v` holds at the world where reached from `t` through `r`. `pure` expresses that when `PROP t f b` holds, then `g b` also holds after `g`.

7.2. **Verification example.** We demonstrate a verification example here. The DUV is C_1 in Figure 3, and we show the deduction at the right-side of Figure 9 again. We show that: assuming the signal path from a through INV to c is faster than the another wire path, when the input wave is given, c is always 0.

The timing constraint is represented as follows.

```
record Constraint : Set₁  where
 field
  Delta : ExArrow Bool Bool
  posDelta : Positive Delta
  conEq :
   Delta >> fork |L >> INV >> L| AND ≡ fork |R >> BUF >> R| AND
```

`Delta` is a positive ExArrow, and satisfies the equation as shown in the lower left of Figure 9.

Input waves are represented as follows.

```
record Input : Set₁  where
 field
  inp₀  :
   {f : ExArrow Bool Bool} →
    Positive f →
     PROP t (- f) false
  inp₁  :
   {f : ExArrow Bool Bool} →
    Positive f →
     PROP t f true
```

`t` indicates the changing point of the values, as a world in the environment. Before `t`, `false` is given by $\texttt{inp}_0$, and after `t`, `true` is given by $\texttt{inp}_1$.

On the premises above, we deduced two theories below; in words of Agda, proof terms were obtained and type-checking passed.

```
Th₀  :
  (f : ExArrow Bool Bool) →
  ∃ (λ g → Positive g × (f >> - (R| AND) >> g ≡ - Δ)) →
   PROP t f false
Th₁  :
  (f : ExArrow Bool Bool) →
  ∃ (λ g → Positive g × (f >> - (L| AND) >> - INV ≡ g)) →
   PROP t f false
```

$\texttt{Th}_0$ expresses a case that `R| AND` dominates output, and $\texttt{Th}_1$ expresses the other case that `L| AND` dominates output. Reverting to Figure 9, on the right side, $\texttt{Th}_0$ indicates the left half of conclusions, and $\texttt{Th}_1$ indicates the right half. Therefore, our method is basically implemented in a theorem proving language. However, some issues are pointed out in the next subsection.

7.3. Issues on the implementation.

On the implementation, we leave three issues. First, since Arrow was adopted, we gave up a distinction of worlds. As the previous subsection shows, all worlds have type `Bool`. It is convenient to define functions of gates, but might bring confusion during reasoning. Secondly, proving needs non-negligible labor. We wrote 24 lines total by the hand to prove $\texttt{Th}_0$ and $\texttt{Th}_1$. Although such a toy example cost like that, proving labor will be a large problem toward the practical use. Finally, the totality of the proof was not guaranteed. We proved $\texttt{Th}_0$ and $\texttt{Th}_1$ individually, and we suppose these compose the demanded property, "c is always 0."; but we could not prove the final property in Agda at this point. There is no way to prove that $\texttt{Th}_0$ and $\texttt{Th}_1$ cover whole worlds for signal c. In general, the problem is lack of a judgment algorithm for worlds' totality.

8. Conclusions.

We presented a verification method for asynchronous circuits and a verification example of an asynchronous FIFO on the method. The method is based on a theorem proving manner; thus the state explosion problem is avoided on a fundamental level, and a verification result is always available regardless of the size of a design under verification and computational resources. We also demonstrated an experimental verification of an FIFO and an implementation of our logical system in theorem proving language, and these show practical utility of our method. Although our method needs hand-proving at this moment, this could be a practical barrier. Future works will attempt to implement a description language and to automate proving on it.

REFERENCES

[1] J. A. Brzozowski and C.-J. H. Seger, *Asynchronous Circuits*, 1995.

[2] P. Blackburn, M. De Rijke and Y. Venema, *Modal Logic*, Cambridge University Press, 2002.

[3] R. Clariso and J. Cortadella, Verification of timed circuits with symbolic delays, *Proc. of Asia and South Pacific Design Automation Conference*, pp.628-633, 2003.

[4] R. Clariso and J. Cortadella, Verification of concurrent systems with parametric delays using octahedra, *The 5th International Conference on Application of Concurrency to System Design*, pp.122-131, 2005.

[5] H. Kim, P. Beerel and K. Stevens, Relative timing based verification of timed circuits and systems, *Proc. of the 8th International Symposium on Asynchronous Circuits and Systems*, pp.115-124, 2002.

[6] X. Kong and R. Negulescu, Bolstering faith in gasp circuits through formal verification, *The 10th International Symposium on Asynchronous Circuits and Systems*, pp.113-124, 2004.

[7] J. Joyce and G. Birtwistle, *Proving a Computer Correct in Higher Order Logic*, 1985.
[8] T. F. Melham, *Higher Order Logic and Hardware Verification*, Cambridge University Press, 2009.
[9] T. Braibant, Coquet: A coq library for verifying hardware, *Certified Programs and Proofs*, pp.330-345, 2011.
[10] M.-M. Bidmeshki and Y. Makris, Vericoq: A verilog-to-coq converter for proof-carrying hardware automation, *IEEE International Symposium on Circuits and Systems*, pp.29-32, 2015.
[11] S. M. Lane, Categories for the working mathematician, *Graduate Texts in Mathematics*, no.5, 1971.
[12] K. Stevens, R. Ginosar and S. Rotem, Relative timing [asynchronous design], *IEEE Trans. Very Large Scale Integration Systems*, vol.11, no.1, pp.129-140, 2003.
[13] R. Negulescu, A technique for finding and verifying speed-dependences in gate circuits, *Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pp.189-198, 1997.
[14] R. Negulescu and A. Peeters, Verification of speed-dependences in single-rail handshake circuits, *The 4th International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp.159-170, 1998.
[15] C. Molnar, I. Jones, W. Coates and J. Lexau, A FIFO ring performance experiment, *The 3rd International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp.279-289, 1997.
[16] U. Norell, Dependently typed programming in agda, *Advanced Functional Programming*, pp.230-266, 2009.
[17] J. Hughes, Programming with arrows, *Advanced Functional Programming*, pp.73-129, 2005.