Working with Petri Nets and Asynchronous Circuits

Tomohiro Yoneda

National Institute of Informatics, Tokyo 101-8430, Japan

Abstract. Petri nets and asynchronous circuits are very important and special topics in my research career. Petri nets have been a useful tool for me to formally verify and synthesize asynchronous circuits. In this paper, I especially focus on handling timing issues in such frameworks, and would like to summarize my research experiences built on both time Petri nets and timed asynchronous circuits.

1 Time Petri nets and timed circuits

A *time Petri net* [1] is one of timed extensions of a Petri net where timing constraints are given to their transition firings. Two nonnegative rational numbers are assigned to each transition of a time Petri net as shown in Fig. 1. In this example, suppose that a transition t_0 becomes enabled (*i.e.*, a marking where both its input places p_0 and p_1 have tokens is reached) at time point T. Then, t_0 cannot fire before time $T + a$, and t_0 must fire before or at time $T + b$.

These timing constraints given to transitions of time Petri nets are useful to model a *timed circuit*, where each gate in it has bounded delays, as shown in Fig. 2 (a). In this paper, it is considered that the timed gate shown in Fig. 2 (a) is equivalent to a gate with an inertial delay element as shown in Fig. 2 (b). The two parameters of the delay element show its lower and upper bounds of the delay, that is, the actual delay of the gate is non-deterministic, but is always within the range. Furthermore, since it is an inertial delay element, a pulse whose width is shorter than a always disappears (*i.e.*, it cannot go through the delay element). A wider pulse may or may not disappear, if its width is shorter than or equal to b.

The timed gate shown in Fig. 2 can be modeled by a time Petri net as shown in Fig. 3. This time Petri net is extended in several points: (1) Each transition

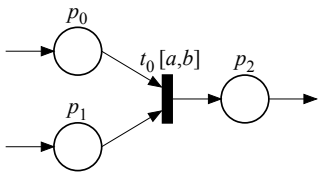

Fig. 1. A time Petri net.

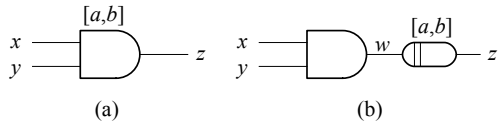

Fig. 2. A timed gate.

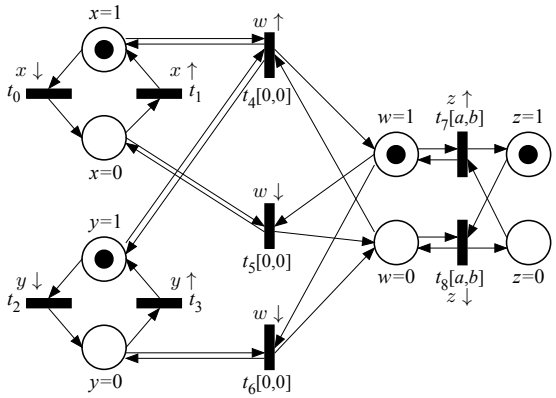

Fig. 3. A time Petri net representing the timed gate shown in Fig. 2.

is labeled with a rising or falling change of a signal. The firing of a transition represents the change of the corresponding signal. (2) Each place is labeled with a value of a signal. If a place has a token, the corresponding signal has the value at the place. (3) Some transitions have no timing constraints (*e.g.*, t_0, t_1, t_2, and t_3). These transitions are called *input transitions*. The remaining transitions are called *output transitions*. When a set of time Petri nets representing gates or modules are considered, the firings of input transitions are invoked in synchronization with the firings of the corresponding output transitions.

2 Partial order reduction

State space enumeration of a formal model is an essential technique needed for both formal verification and asynchronous circuit synthesis. However, it is not easy to complete the state space enumeration of a system unless the system is very small. Thus, we focused on the fact that our purpose can be achieved by checking whether some specific properties hold or not. One such simple example of the properties is whether a specific transition can fire in some reachable state. This property can easily be checked, if the whole reachable state space is examined, but it is also possible to check it in a reduced state space. The partial order reduction is a technique to obtain such a reduced state space for the given property. Stubborn set method [2] is one of such techniques, and we extended similar ideas for time Petri nets [3, 4].

Let's consider a time Petri net shown in Fig. 4 (a), and assume that we want to check whether transition t_5 can fire in some reachable state. After firing t_0, t_1 can fire concurrently with t_3 and t_4 from their timing constraints. Two transitions t_2 and t_5 are *in conflict*, which means that firing one of those transitions disables the other. After firing t_1, t_2 becomes enabled. Again, from the timing constraints, t_2 can fire at the state, and its firing disables t_5. Similar situation happens for

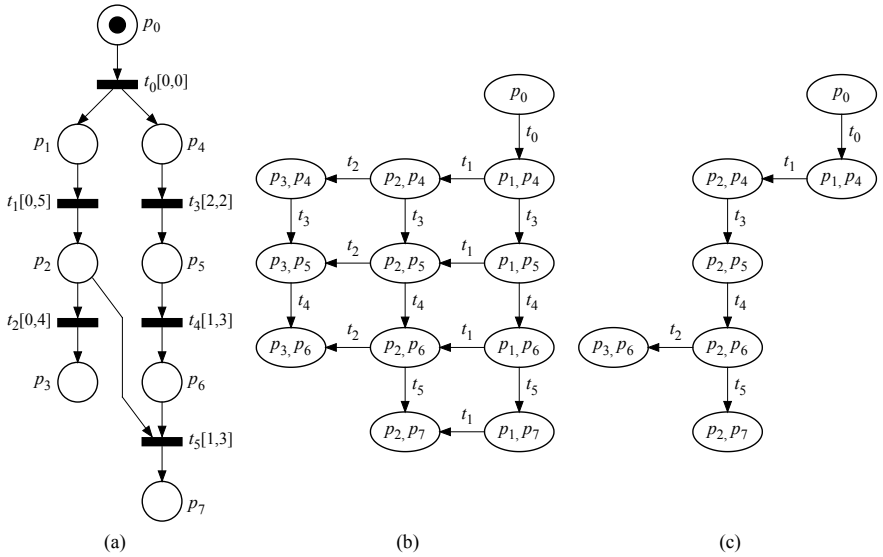

Fig. 4. (a) A time Petri net, (b) its full state space, and (c) its reduced state space.

t_5. Hence, we have a reachable state graph as shown in Fig. 4 (b). This graph is obtained by firing every transition that can fire in each state. This is the full state enumeration.

In order to check the above property without constructing the full state graph, we usually fire one enabled and *firable* transition in each state. Intuitively, an enabled transition is firable, if it can fire with respect to its timing constraints. After firing t_0, both t_1 and t_3 satisfy this condition. Since we choose one such transition arbitrarily, assume that t_1 is chosen. After firing t_1, t_2 and t_3 are such transitions. This state is special, because firing t_2 takes away the possibility to fire t_5. Thus, firing only t_2 in this state may lead to an incorrect decision. Instead, enabled and firable transitions that may eventually make t_5 enabled in time are searched. In this case, it is t_3. Our algorithm requires that t_3 should also be fired when t_2 is fired from this state. On the other hand, t_3 is an enabled and firable transition in this state, and it conflicts with no other transitions. Thus, when t_3 is fired in this state, it is not required to fire any other transitions. For this reason, our algorithm chooses to fire t_3 in this state. t_4 is fired similarly, and then, both t_2 and t_5 become enabled. In this state, there is no other option except for firing t_2 or t_5, and the reduced state space shown in Fig. 4 (c) is obtained.

A key idea of our algorithm is that in case where firing a transition t_a is considered and a disabled transition t_b is in conflict with t_a, enabled and firable transitions that may eventually make t_b enabled in time are searched. Firing such transitions (as well as t_a) prevents us from missing possible transition firings. By the way, the above "in time" is also important. For example, if the

timing constraints of t_3 is $[8, 8]$ instead of $[2, 2]$, t_3 cannot make t_5 enabled in time, because t_2 must fire before time $T(t_0) + 9$ (let $T(u)$ denote the time when transition u fires).

I mainly worked with Holger Schlingloff for this research topic. We both stayed at Carnegie Mellon University in 1990-1991 as visiting researchers of Prof. E. M. Clarke. We published 7 co-authored papers. An incomplete list of researchers with whom I communicated on this topic is B. Berthomieu, P. Godefroid, K. L. McMillan, D. Peled, and A. Valmari.

3 Formal verification of timed asynchronous circuits

Our framework of formal verification of timed asynchronous circuits is a conformance checking. In this framework, a *safety failure* for given specification and circuit is a situation where a possible behavior in either side (specification or circuit) is not possible in the other side. This situation usually happens when the circuit produces a bad output that is not specified in the specification. For this purpose, the given specification is first modeled by a time Petri net. This time Petri net specifies when the input signals of the circuit can change, with the timing constrains given to its output transitions. It also specifies when the output signals of the circuit can change, with its input transitions. For example, Fig. 5 (b) is a specification of a C-element in Fig. 5 (a). Note that transitions labeled with τ are *dummy* transitions that are output transitions, but have no corresponding input transitions.

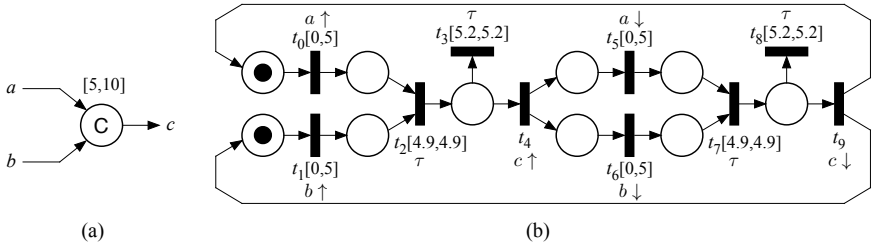

Fig. 5. (a) a circuit, and (b) its specification.

A circuit is modeled by a set of time Petri nets. As shown in Fig. 3, a simple gate is straightforwardly modeled by a time Petri net. For connecting several gates to compose a circuit, it is only needed to prepare a set of gates. Two important roles of our verification algorithm are (1) to synchronize the firings of corresponding input and output transitions to explore the state space of the system (*i.e.*, a set of time Petri nets for a specification and gates), and (2) to check whether for any firable output transition there exists a corresponding enabled input transition. By the role (1), when an output of a gate is connected to an input of another gate, its connection is represented by firing those input and

output transitions in synchronization with each other. The role (1) also works for specifying how a circuit should be used with respect to the specification. Remember that the output transitions of the specification decide how the corresponding input signals of a circuit behave, by synchronizing the corresponding output transitions in a specification and the input transitions in a circuit. That is, the specification stimulates the circuit. The role (2) is exactly for detecting the safety failures.

As mentioned in the previous section, it is not usually possible to explore the full state space. Thus, we have applied the partial order reduction for the conformance checking. For this purpose, we need to consider a property for the partial order reduction, such that a safety failed state is reachable in the reduced state space if and only if a safety failed state is reachable in the full state space. Remember that the simple partial order reduction algorithm mentioned in Section 2 searches enabled and firable transitions that make t_b enabled in time, where t_b is in conflict with t_a and t_a is fired at the current state. For handling the conformance checking, this search is extended such that an input transition t_c is considered for the search similarly to t_b, where t_d is an output transition that corresponds to t_c and t_d is made enabled by t_a (see Fig. 6).

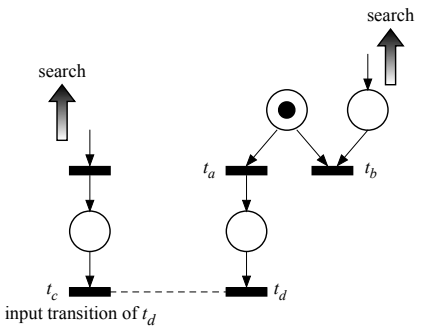

Fig. 6. Extended search of enabled and firable transitions.

I mainly worked with Chris Myers for this research topic. We had a joint travel grant for 3 years, and published 15 co-authored papers related to this topic. An incomplete list of researchers with whom I communicated on this topic is P. A. Beerel, J. Cortadella, D. Dill, K. L. McMillan, and H. Schlingloff.

4 Synthesis of asynchronous circuits

Synthesis of an asynchronous circuit is to obtain an asynchronous circuit M for a given specification G such that M behaves identically to G. In our case, G is given as a timed signal transition graph, which is a kind of a time Petri net. By using timed signal transition graphs for specifications, several behaviors that are

impossible due to the given timing constraints are ignored, and thus, a smaller circuit is expected to be finally synthesized.

If the full state space of G is obtained, and it satisfies several properties, M can be obtained by a standard logic synthesis algorithm for asynchronous circuits. Such logic synthesis algorithm is implemented in several tools. The most popular tool may be Petrify [5]. Again, it is not easy to explore the full state space of the given specifications, especially when those specifications are obtained in a high-level synthesis process.

Our approach to this problem is a decomposition based synthesis [6, 7]. In our approach, for an output x of the given G, some subset V of signals of G and $\mathsf{abs}(G, V, x)$ are first obtained, where $\mathsf{abs}(G, V, x)$ and G have equivalent behaviors with respect to signals in V, and $\mathsf{abs}(G, V, x)$ is synthesizable. Then, a standard logic synthesis algorithm is applied to $\mathsf{abs}(G, V, x)$ instead of G, which generates a sub-circuit for an output x. This process is repeated for each output of G. Usually, the state space of $\mathsf{abs}(G, V, x)$ is much smaller than that of G. Thus, the synthesis time can be dramatically reduced. The construction of V is, however, not trivial. Actually, it can easily be obtained from the full state space of G, but it does not help us. Our key contribution is an algorithm to efficiently obtain V using a technique similar to the partial order reduction for time Petri nets. Note that this approach uses a restricted information through a subset V of signals of G. Thus, the synthesized circuits by our method may not be optimal. According to the experimental results, however, this overhead is pretty small [6, 7].

I worked with Chris Myers also for this research topic. We published 7 co-authored papers related to this topic. An incomplete list of researchers with whom I communicated on this topic is J. Cortadella, V. Khomenko, M. Schaefer, and W. Vogler.

5 Conclusion

In this paper, I summarized my previous work related to time Petri nets. I have chosen such research topics, because I had many chances to talk with Alex Yakovlev through those topics in related conferences such as ASYNC and ACSD. Recently, I have been more interested in designing asynchronous circuits such as asynchronous NoCs. At ACSD conference in 2011, Alex invited me as a key-note speaker, and I was greatly honored to be able to give a talk on asynchronous NoCs there.

References

1. P. Merlin and D. J. Faber. Recoverability of communication protocols. *IEEE Trans. on Communication*, COM-24(9):1036–1043, 1976.
2. A. Valmari. Stubborn sets for reduced state space generation. *LNCS 483 Advances in Petri Nets*, pages 491–515, 1990.

3. T. Yoneda, A. Shibayama, H. Schlingloff, and E. M. Clarke. Efficient verification of parallel real-time systems. *LNCS 697 Computer Aided Verification*, pages 321–332, 1993.
4. T. Yoneda and H. Schlingloff. Efficient verification of parallel real-time systems. *Formal Method in System Design*, pages 187–215, 1997.
5. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, March 1997.
6. T. Yoneda, H. Onda, and C. Myers. Synthesis of speed independent circuits based on decomposition. *Proc. of 10th International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 135–145, 2004.
7. T. Yoneda and C. J. Myers. Synthesis of timed circuits based on decomposition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(7):1177–1195, July 2007.