An improvement in Timing Analysis of Asynchronous Real-time System

Yingying Zhang

CIS 6930 Final Project Report

Department of Computer Science and Engineering
University of South Florida

September 17, 2012
# Contents

1 Introduction .................................................. 3
   1.1 Related Work ........................................... 4
   1.2 Overview ............................................... 4

2 Timing Analysis .............................................. 6
   2.1 Labeled Petri-Nets: Definition .......................... 6
   2.2 Labeled Petri-Nets: Semantics .......................... 9
   2.3 Zones .................................................. 10
   2.4 Reachability Analysis ................................... 15
       2.4.1 Untimed Reachability Analysis ..................... 16
       2.4.2 Timed Reachability Analysis ....................... 16
   2.5 Examples .............................................. 20

3 POSET Timing Analysis ....................................... 34
   3.1 Introduction ............................................ 34
   3.2 Algorithms ............................................. 35
   3.3 POSET Timing Example ................................... 42

4 Improvement of POSET Timing Analysis ....................... 56
   4.1 Introduction ............................................ 56
   4.2 Algorithms ............................................. 56
   4.3 Example ................................................. 60

5 Conclusion .................................................. 74

Acknowledgements ........................................... 74

Bibliography ................................................ 75
Chapter 1

Introduction

With the increasing complexity of modern system, making errors during design process has increasing exponentially [3]. In order to detect the whole possible errors, model checking [23][24] plays an important role to improve the quality of system design. Model checking, which is a formal automated analysis method for verifying hardware and software systems, systematically checks whether a model of a given system satisfies a desired property such as deadlock freedom and request-response properties [5]. Properties of the system are expresses as formulas which uses the state-transition graph as a model [11].

For synchronous systems, all state variables are updated simultaneously due to the global control of a clock. Instead of sharing a common clock and exchanging data on clock edges, asynchronous designs communicate through control protocols, and multiple components can perform executions concurrently. When verifying such a system, concurrently enabled executions need to be interleaved so that all possible orderings of executions are considered to avoid missing any behavior. The need to consider all possible interleavings of concurrent executions is the main cause of state explosion as the number of interleavings grows exponentially if a system has a high degree of concurrency, and this leads to an excessively large state space for even a relatively small system.

When considering timing properties in asynchronous system, it can be seen as asynchronous real-time system. It determines the correctness behavior not only depends on the result of the computations but also on the time constraint when these result generated [10], which is always used in time-critical applications. To verify this asynchronous real-time system with timing properties, model checking assumes that timing constraints are given in each transitions explicitly. The verifier determines whether the system satisfies the timing constraints during analysis.
1.1 Related Work

In the last two decades, model checking applied in real-time system is considerably short. First researchers [18][15] focus on discrete time analysis, which can be extended on untimed model checking easily. The discrete-time analysis technique was first applied to timed circuits by Burch [21][16].

If considering the model in the asynchronous real-time system, new complexities arise since we need to focus on dense time instead of discrete time. A standard dense time approach [19][13] is to analyze the transition relation with a finite set of real-valued clocks which can be proceeded at a uniform rate, then to constrain the clocks and determine which transition can be fired.

Dill devised the zone method of representing timed states in [22] to save dense time of each clocks. Myers developed an efficient method in [9] to perform timed state space exploration on circuit design. Since the zone approach had a considerable shortcoming for concurrency system, Myers and Rokicki introduced a partially ordered sets timing analysis approach [9] to separate concurrency from causality, which lead to significant improvement in circuit quality. Rokicki also introduced a structure orbital nets [12] to model circuit systems and presented a sequence of timing analysis techniques to explore state space which reduced the number of edges in the timed reachability graph.

1.2 Overview

We have divided our presentation into five chapters. Each chapter introduces information based on the previous chapters by adding more concepts and algorithms.

The introduction chapter describes how model checking works on the timing synchronous systems and what’s the problem during model checking.

Chapter 2 introduces a common modeling formalism, labeled petri net (LPN), to model the asynchronous designs. And we describes how timing affects the execution and behavioral semantics of the LPNs. Next, we introduce several different methods to model timing properties of LPNs, such as discrete time, region and zone, choose the effective zone approach to analyze the timing difference between each compared transition pair of LPNs. Then, we discuss the reachability analysis approach to analyze the dynamic behavior of a concurrent asynchronous system modeled with timed labeled petri nets, and show the whole state space exploration process with updated zone algorithms. Finally, we give a real-time circuit example, and analysis the timing
change process for each firing event during reachability timing analysis.

Chapter 3 introduces an improved method, POSET, to deal with the redundancy zone generated by zone approach. For the concurrent untimed system, firing the same transitions in different order can result in the same final end state. But for real-time system with zone method, it will result in different zones. In order to deal with this problem and reduce the state space without effect the final result, POSET method maintains an extra matrix to records fired but still necessary transitions’ clock information, and derive the new zone based on the updated POSET matrix. Using this POSET method, we can make sure to get the exact one final zone for firing the same transitions in different order. For better understanding, we gives the detail process and algorithms to explore state space with updated POSET matrix and zone, then analyze the timing change process for the same circuit example as Chapter 2 and proves the state space reduction.

Actually, the POSET method still cost much more memory space to keep POSET matrix and zone information for each firing event. In order to save state space even more, Chapter 4 introduces an improvement method based on the zone analysis approach with POSET thought. In this case, we do not need to have extra step to maintain and update the POSET matrix, only update and keep the zone information for each zone generation step. For better understanding, we gives the detail process and algorithms to explore state space with updated zone information, then analyze the timing change process for the same circuit example as Chapter 3, which can saving the state space by ignore POSET matrix and keep the same result as example in Chapter 3.

Chapter 5 summaries two improved methods to verify concurrent synchronous systems via model checking. One is to use zone concept in Labeled Petri nets structure during reachability analysis. Another one is to give an improved method with POSET thoughts to reduce the sate space.
Chapter 2

Timing Analysis

Timing is an important component of asynchronous real-time system. In order to better understand how to do timing analysis during model checking, we introduce the labeled petri-nets which are widely used to model and analyze synchronous designs, then describe the corresponding timing analysis algorithms. At last, we give an example to explain how the timing reachability analysis works step by step.

2.1 Labeled Petri-Nets : Definition

This paper uses Labeled Petri-Nets which describe a large state space with a compact structure to model asynchronous systems. Petri-Nets are a common modeling formalism for asynchronous designs [20]. A Petri-net is a directed graph with a set of transitions and a set of places. A labeled Petri-Net is a Petri-net where transitions are labeled with various information representing a system’s properties and behavior [1]. Its definition is given as follows.

Definition 2.1.1 A labeled Petri-net (LPN) is a tuple \( N = (V, P, T, F, \mu_0, init, L) \), where

1. \( V \) is a set of state variables of the integer type,
2. \( P \) is a finite set of places,
3. \( T \) is a finite set of transitions,
4. \( F \subseteq (P \times T) \cup (T \times P) \) is a finite set of the flow relations,
5. \( \mu_0 \subseteq P \) is a finite set of initially marked places,
6. \textit{init} : \(V \rightarrow \mathbb{Z}\) is a labeling function that assigns each variable an initial value.

7. \(L = \langle \text{Guard}, \text{Delay}, \text{Assign} \rangle\) is a triple of labeling functions for transitions in \(T\), which is defined below.

A simple asynchronous circuit and its corresponding LPNs are shown in Figure 2.1 and Figure 2.2. The circuit in Figure 2.1 consists of three components, and Figure 2.2 shows the LPNs for each component in the circuit.

![Figure 2.1: A simple asynchronous circuit.](image)

For each component, its LPN has 4 places and 4 transitions. The places are represented as circles, and the transitions are represented as boxes. Each place is preceded and followed by one or more transitions, and each transition is preceded and followed by one or more places. The flow relations are represented by the edges connecting the transitions and places. The bullets found in some places are called tokens. Each place can have at most one token. A place is marked if it has a token. A marking of LPN, \(\mu \subseteq P\) is a set of marked places.

The dynamic behavior of a concurrent system is captured by LPN transitions with labeling. Each transition \(t \in T\) has a \textit{preset} denoted by \(\bullet t = \{p \in P|(p, t) \in F\}\), which is the set of places connected to \(t\), and a \textit{postset} denoted by \(t\bullet = \{p \in P|(t, p) \in F\}\), which is the set of places to which \(t\) is connected. The \textit{preset} and \textit{postset} for places are defined similarly.

Before defining the transition labels formally, the grammar used by these labels is introduced first below [1]. The numerical portion of the grammar is
defined as follows:

\[
\chi ::= c_i \mid v_i \mid (\chi) \mid -\chi \mid \chi + \chi \mid \chi - \chi \mid \chi \ast \chi \mid \\
\chi/\chi \mid \chi^\chi \mid \chi \% \chi \mid \text{NOT}(\chi) \mid \text{OR}(\chi, \chi) \mid \\
\text{AND}(\chi, \chi) \mid \text{XOR}(\chi, \chi)
\]

where \(c_i\) is an integer constant from \(\mathbb{Z}\), and \(v_i\) is an integer variable. The functions NOT, OR, AND, and XOR are bit-wise logical operations, and they are only applicable to integers and assume a 2’s complement format with arbitrary precision. The set \(P_\chi\) is defined to be all formulas that can be constructed from the \(\chi\) grammar.

The Boolean portion of the grammar is as follows:

\[
\phi ::= \text{true} \mid \text{false} \mid v_i \mid \neg \phi \mid \phi \& \phi \mid \phi \lor \phi \mid \chi \equiv \chi \mid \\
\chi \geq \chi \mid \chi > \chi \mid \chi \leq \chi \mid \chi < \chi
\]

where the integer \(v_i\) is regarded as \text{true} if its value is nonzero, and \text{false} otherwise. In this sense, it is similar to the semantics of the C language. The set \(P_\phi\) is defined to be all formulas that can be constructed from the \(\phi\) grammar.
As in Definition 2.1.1, each LPN transition is labeled with an enabling condition and a set of variable assignments. LPN transition labeling is defined by 

$$L = \langle \text{Guard}, \text{Delay}, \text{Assign} \rangle$$

- **Guard**: $T \rightarrow P_\phi$ labels each LPN transition with a Boolean expression that defines its enabling condition.
- **Delay**: $T \rightarrow Q^+ \times (Q^+ \cup \infty)$ labels each LPN transition with a possibly unbounded delay which includes a pair of time constraint $[EFT, LFT]$, where $Q^+$ is the set of non-negative rational numbers, $EFT$ represents the earliest firing time and $LFT$ represents the latest firing time.
- **Assign**: $T \times V \rightarrow P_\chi$ labels each LPN transition $t \in T$ and variable $v \in V$ with a set of integer assignments made to $v$ when $t$ fires.

For transition $t_1$ in Figure 2.2, the place in $\bullet t_1$ is marked, and $\bullet t_1$ is the same as $t_1 \cdot$. Its enabling condition $Guard(t_1) = (z = 0) \land (v = 0)$, and it has one assignment $Assign(t_1) = \{v := 1\}$. The Delay constraint for $t_1$ is $[1,3]$ which means this transition can only be enabled during $[1,3]$ clock period. It is explained further in Section 2.2.

### 2.2 Labeled Petri-Nets: Semantics

In this section, we describe how timing changes the execution and behavioral semantics of Labeled Petri-Nets. The semantics of LPN is defined as a transition systems where the state includes current location, variable information and the current values of clocks.[7]. The state of the LPNs is a triple $(\mu, \sigma, C)$, where $\mu$ denotes the marking, $\sigma$ denotes the vector of variable values and $C$ denotes current non-negative value of clocks. Given a state $s$, $\mu(s)$ is a set of marked places in $s$, $\sigma(s)$ is the state vector of $s$, $C(s)$ is the clock vector of $s$. Also, for any expression $e \in P_\chi \cup P_\phi$, $value(e, s)$ denotes a function that returns the value of expression $e$ in state $s$.

In LPNs, the clock of the transition $t$, denoted as $C.t \in C(s)$, is initially set to zero if it is initial marked, or set to a value belonging to the time interval of this transition which is fired and cause this place. Then, clocks of places develop with time synchronously. [4] With this constraint of time function, the LPNs enabled transition and firing transition are defined below:

**Definition 2.2.1 (Timed LPNs Enabled Transition)** A LPN transition $t$ is enabled at state $s$ only if the following two conditions are met:

1. $\bullet t \subseteq \mu(s)$,
2. value(e, s) is true or not zero for \( e = \text{Guard}(t) \).

**Definition 2.2.2 (Timed LPNs Fired Transition)** A LPNs transition \( t \) can be fired at state \( s \) only if the following two conditions are met:

1. \( t \in \text{enb}(s) \), \( \text{enb}(s) \) represents all enabled transition at state \( s \).
2. \( C.t \in [EFT, LFT] \).

An enabled transition can be only fired when its clock time is no less than \( EFT \), and must be fired before \( LFT \) if continuously enabled. Time may not increment beyond the minimum deadlines set by the \( LFT \) of all enabled transitions.

A simple circuit example is shown in Figure 2.2, every transition has its preset included in the initial marking. In the initial state, the values of variable \( u \) and \( z \) are 0, \( \text{Guard}(t_{11}) \), which is \( u = 0 \land z = 0 \), is evaluated to be true, and the initial clock of this transition is set to zero, which is in time constraint \([0,3]\), therefore transition \( t_{11} \) is enabled in the initial state and can only be fired between 0 and 3 clock time.

Based on above two definitions, there are two types of transitions between states, one is the delay transition \( \delta(t) \) which means it can be delay for some time \( d \), another one is the action transition \( \omega(t) \) which means it can be fired immediately.

**Definition 2.2.3 (Operational Semantics)** The semantics of the LPNs is a transition system where the state is tuple \( \langle \mu, \sigma, c \rangle \), and transition \( t \) is defined by the rules:

1. \( \langle \mu, \sigma, C \rangle \xrightarrow{\delta(t)} \langle \mu, \sigma, C + d \rangle \) if \( C.t \in t_{[EFT,LFT]} \) and \( (C.t + d) \in t_{[EFT,LFT]} \) for a non-negative real \( d \in Q^+ \).

2. \( \langle \mu, \sigma, C \rangle \xrightarrow{\omega(t)} \langle \mu', \sigma', C' \rangle \) if \( \langle \mu, \sigma \rangle \xrightarrow{t} \langle \mu', \sigma' \rangle \) and \( C.t \in t_{[EFT,LFT]} \). For each enabled transition \( t' \) in the new state, set \( C'.t' = 0 \) if \( t' \in \text{enb}(\mu', \sigma') \setminus \text{enb}(\mu, \sigma) \), \( C'.t' = C.t \) if \( t' \in \text{enb}(\mu', \sigma') \cap \text{enb}(\mu, \sigma) \).

### 2.3 Zones

In real-time systems, the correctness not only depends on the result of output but also on the clock time in which this output is produced. Therefore, in order to verify real-time system with timed petri-net structure correctly, it is necessary to analysis the timed state space for each untimed state to
determine which transition can be enabled to fire. There are three techniques
to divide the timed state space for each untimed state into equivalence classes
[9].

The first techniques is called regions. It consists of each clock into an
integer and a fractional components. Given a system with two clocks \( c_1 \) and
\( c_2 \), which can take any values between 0 and 5. When the two clocks have
zero fractional components, the region is a point. When clock \( c_1 \) has a zero
fractional component and \( c_2 \) has a nonzero fractional component, the region
is a vertical line segment. When clock \( c_1 \) has a nonzero fractional component
and \( c_2 \) has a zero fractional component, the region is a horizontal line seg-
ment. When both two clocks have nonzero but equal fractional components,
the region is a diagonal line segment. Finally, when two clocks have nonzero
fractional components and one clock is larger, the region is a triangle. The
set of all possible equivalence classes is shown in Figure 2.3. Each triangle
is a region, as is each point and each line segment. Thus, we have two open
triangles, one diagonal, one vertical and one horizontal line segment foe each
integer value less than five, plus a few border regions, and the total is 171
regions.

For timed petri-nets, we never need to check if a clock is strictly less
or greater than a bound, so the fractional components are not necessary
and discrete timing technique can be used to track the states. This method
calculates every possible combination of integer time values for each untimed
states. Given two clock \( c_1 \) and \( c_2 \), which can take any values between 0 and
5, the possible timed states are shown in Figure 2.4, which contains only
36. Unfortunately, this discrete timing technique is still exponential in the
number of concurrent clocks and the timing bounds.
In order to represent equivalence classes of timed states effectively, we use convex polygons, called zones, which can consider the entire square region in Figure 2.3 as a single region shown in Figure 2.5. In other word, one zone can represent 171 regions or 36 discrete states.

Let us consider how to describe the square region to a single one. Obviously, each clock needs upper bounds $c_u$ and lower bounds $c_l$. If one transition starts a clock $t_i$ and enabled another transition with separately, and the latter transition also start a clock $t_j$. So $t_j$ should be less than or equal to the value of $t_i$ because $t_j$ was started after $t_i$, then we can represent a zone by a set of following constraint: $t_i <= c_u, c_l <= t_i, t_i - t_j <= c_d$, where $c_l, c_u, c_d$ are integer constants. To make them uniform, a dummy clock $t_0 = 0$ is introduced, then we can change $t_i <= c_u$ to $t_i - t_0 <= c_u$, change $c_l <= t_i$ to
Then any convex polygon can be represented by a data structure called different bound matrix (DBM), each element in DBM is calculated by a linear inequality of each pair of clocks ($t_i - t_j \leq c_d$). For the example in Figure 2.3, a single region with the set of linear inequalities and the corresponding DBM is shown in Figure 2.6. In this DBM, the maximum value of the clock are in the top row and the minimum values of the clock are in the left column (which are negated).

During state space explosion, it is necessary to analysis when you have encountered a timed state that is visited before, and whether the zone is same as before. Actually, the same zone can be represented by many different DBMs. In order to make sure there is a canonical DBM for each zone, we need to tight all inequalities maximally, called canonicalization. It means to reduce of all the inequalities to their most constrained value.

Consider a DBM as an adjacency matrix to stand for a weighted, directed graph, finding the canonical DBM is equivalent to finding all pairs of shortest paths, which can be handled by Floyd’s algorithm shown in Algorithm 1. The DBM of zone is noted as $M$ in the algorithms and examples, $M[i, j] = c$ represents the clock time $t_j - t_i \leq value$ which is shown at row $i$ and column $j$.

**Algorithm 1: canonicalize($M$)**

**Input:** A zone $M$

**Output:** A tightest matrix for the same zone

1. **foreach** $i = 0$ to $n - 1$ **do**
2.   **foreach** $j = 0$ to $n - 1$ **do**
3.     **foreach** $k = 0$ to $n - 1$ **do**
4.       **if** $M[j, k] > M[j, i] + M[i, k]$ **then**
Figure 2.7 shows a demo of DBM with its corresponding graph. Using Floyd’s all-pairs shortest-path algorithm to adjust the weight for each edge, then get the new DBM. From this graph, the weight from $t_2$ to $t_1$ can be changed to shortest distance 5, and so as the weight from $t_1$ to $t_2$. The new DBM is shown in Figure 2.8.

State space exploration could not finish if the clocks are unbounded, which means the clock can take on any nonnegative value above its minimum value. In order to keep exploration finitely, normalization method is used to ensure that no DBM is unbounded. This method assumes that the value of the clock with an infinite upper bound is irrelevant once it exceeds its lower bound. This reduction operation will always keep a constraint DBM whose constraints satisfy finite upper bounds.

We define $premax$ to record the max constraint value of transition. If one transition has exactly upper bound time $t_u$, then $premax = t_u$, otherwise, $premax$ is equal to the lower bound time $t_l$. Then normalization adjust clock in following three steps: First, if the lower bound of the clock $t_i$ has already exceeded its $premax$, then reduce back to $premax$. Then to find the minimum maximum value, which means choose the minimum value for each clock and reduce the clocks whose maximum already exceeds $premax$. Algorithm 2 shows main normalization. Finally, if there exist clock with changed upper bound, then canonicalize the DBM.
Algorithm 2: Normalize\((M, enb)\)

**Input:** A zone \(M\) and the enabled transition set \(enb\)

**Output:** Updated zone

1. \(\text{foreach } t_i \in enb \text{ do} \)
2. \(\text{if } M[i, 0] + \text{premax}_i < 0 \text{ then} \)
3. \(\quad \text{temp} = M[i, 0] + \text{premax}_i; \)
4. \(\quad \text{foreach } j = 1 \text{ to } n \text{ do} \)
5. \(\quad \quad M[i, j] = M[i, j] - \text{temp}; \)
6. \(\quad M[j, i] = M[j, i] + \text{temp}; \)
7. \(\text{foreach } t_i \in enb \text{ do} \)
8. \(\text{if } M[0, i] > \text{premax}_i \text{ then} \)
9. \(\quad \text{temp} = \text{premax}_i; \)
10. \(\quad \text{foreach } j = 1 \text{ to } n \text{ do} \)
11. \(\quad \quad \text{if } \min(M[0, j], \text{premax}_j) - M[i, j] > \text{premax}_i \text{ then} \)
12. \(\quad \quad \quad \text{temp} = \min(M[0, j], \text{premax}_j) - M[i, j]; \)
13. \(\quad \text{if } M[0, i] > \text{temp} \text{ then} \)
14. \(\quad \quad M[0, i] = \text{temp}; \)
15. \(\quad \quad \text{changed.add}(i); \)

The detail of zone change is shown in Section 2.4.

### 2.4 Reachability Analysis

A basic approach for analyzing the dynamic behavior of a concurrent system modeled with LPNs is reachability analysis (RA), which finds all possible state transitions and thus reachable states for such a system. The reachable state space is typically represented by a state graph. State graph is a directed graph where vertices represent states and edges represent state transitions.

Given a LPN model, the set of transitions enabled in a state \(s\) is denoted by \(enb(s)\). The reachable state space of a LPN model can be found by exhaustively firing every enabled transitions starting at the initial state. Firing a transition leads to a new state by generating a new marking and a new state vector according to the assignments labeled for such a transition. \(s' = t(s)\) denotes that a new state \(s'\) is produced by firing transition \(t\) in state \(s\).
2.4.1 Untimed Reachability Analysis

For untimed system, the transition can be enabled when its preset is marked and $Guard(t)$ is satisfied. The procedure to find the untimed reachable state space of a given LPN model is given in Algorithm 3. Figure 2.9 shows a fully explored untimed state graph of the LPN example in Figure 2.2, which uses reachability analysis to traverse all possible states and transitions. There are 20 states found in this graph.

**Algorithm 3: search($\langle T, P, F, \mu_0 \rangle$)**

- **Input:** $T$: transitions, $P$: places, $F$: flow relation, $\mu_0$: initial marking,
- **Output:** Reachable number of state in stateTable

1. stack.push($s_0$);
2. stateTable.add($s_0$);
3. while stack is not empty do
   4. $s = \text{stack.pop}()$;
   5. foreach $t \in \text{enb}(s)$ do
      6. $s' = t(s)$;
      7. if $s' \notin \text{stateTable}$ then
         8. stack.push($s'$);
         9. stateTable.add($s'$);

2.4.2 Timed Reachability Analysis

Timing analysis proceeds as does untimed analysis. The set of reachable timed states forms a reachability graph. With zone technique, each state in the reachability graph consists of an untimed state and a timed zone representing reachable timed states. Given a timed zone $z$ and a transition $tr$ to fire, we can calculate the new timed zone $z'$ that represents all reachable timed states by firing $tr$ or advancing time from any timed states in $z$.

To use zone for verification on a timing system, we must represent the initial condition, calculate the enabled transitions and corresponding timing information, then fire transitions and update the zone with new timing information. The whole process need to under the constraint listed in the above two sections. The detail operations are shown in Figure 2.10.

During timing state space exploration with zones, a transition is timing-enabled if its clock can reach a value that if satisfied(i.e, transition $t_i$ can be fired if $M[0,t_i]$ large than or equal to the lower bound of $t_i$). Whenever there exist the untimed-enabled transitions, we need to determine which transitions
are timing-enabled in the current zone. Then select one transition to fire and generate a new zone and check whether this worked new zone has been seen before. If it has, backtrack; otherwise, add it to the hash table. Final, calculate the new untimed-enabled transitions and repeat the whole process till there is no new untimed-enabled transitions.

The core of this process is to deal with the new zone as follows:

The first step is to restrict the new zone based on clock constraint, which means this transition can be fired while its clock must have reached its low bound. Because this constraint may lead to un-maximal tight, then it is necessary to canonicalize the resulting DBM.

The second step is to project the row and column of the fired transition’s clock, because it is no longer need to keep this transition’s clock after firing.

After deleting the fired transition’s clock, we need to extend new clocks if the fired transition cause new enabled transitions, which mean adding a new row and column in the DBM. Algorithm 4 shows the extend process. We initialize the lower and upper bounds of enabled transition’s clock to 0, set the clock separation between new clocks to 0, keep the remain row values equal to the upper bounds of their clocks and keep the remain column values equal to the lower bounds of their clocks. Figure 2.11 shows the comparison of original DBM and new DBM with one transition’s extension.[12]
Figure 2.10: Verification on a timed System
Algorithm 4: $\text{extend}(M, T_{\text{new}})$

**Input:** $M$: zone, $T_{\text{new}}$: new enabled transitions  
**Output:** Extended zone

1. foreach $T_i \in T_{\text{new}}$ do  
2.     $M[i, 0] = M[0, i] = 0$;  
3.     foreach $T_j \in T_{\text{new}}$ do  
4.         $M[i, j] = M[j, i] = 0$;  
5.     foreach $T_j \in \text{enb/T}_{\text{new}}$ do  
7.         $M[j, i] = M[j, 0]$;

---

Figure 2.11: Comparison of original DBM and new DBM with one transition’s extend.
The initial state has only one clock for $t_{11}(z^+)$, so the initial DBM is only two by two matrix with all zeros initially. We then advance time by setting the top row for $z^+$ to the maximum value 3. The canonicalization and normalization have no effect. The transition $z^+$ can be fired in the initial zone because $M[t_0, z^+] \geq 0$.

The third step is to advance clock to make sure all clocks are set to their upper bound. Because this change can also lead to un-maximal tight, then it is necessary to canonicalize the resulting DBM. In this step, we also need to examine whether the constraint timing values are all below the maximum timing bound requirement. If any value is larger than the maximum timing bound requirement, we need to report a failure.

The final step is to normalize the zone, which insure DBM remain bounded in order to keep the state space finite.

2.5 Examples

This section illustrate timed state space explosion with zone using circuit example shown in Figure 2.1. The time constraint for each signal as follow:

- $t_1(v^+) : [1, 3]$
- $t_2(v^-) : [1, 3]$
- $t_3(y^+) : [1, 3]$
- $t_4(y^-) : [1, 3]$
- $t_5(w^+) : [0, 2]$
- $t_6(w^-) : [0, 2]$
- $t_7(x^+) : [0, 2]$
- $t_8(x^-) : [0, 2]$
- $t_9(u^-) : [2, 5]$
- $t_{10}(u^+) : [2, 5]$
- $t_{11}(z^+) : [0, 3]$
- $t_{12}(z^-) : [0, 3]$

Figure 2.12 - Figure 2.25 explain how to generate zones based on zone timing analysis step by step. And Figure 2.26 shows the whole process graph generated via zone timing analysis method. For this partial search, there are 13 zones generated for 10 untimed states in left side of Figure 2.9.
First, we do not need to restrict $z+$’s clock, since $M[z+, t_0] = 0$ and the lower bound of $z+$ is 0. And the canonicalization has no effect. Next, we project the clock of $z+$ and extend DBM to include the clocks of new enabled transitions $t_2(v-)$ and $t_6(w-)$. Then, we advance all clocks by setting the 0th rows of DBM to their upper bounds, and canonicalize it. The normalization has no effect. In this new zone, transitions $w-$ and $v-$ can either be fired.
In this case, we choose to fire $w-$ ahead and mark that we still need to back-track $t_2(v-)$ in other order. First, we do not need to restrict $w-$’s clock, since $M[w-,t_0] = 0$ and the lower bound of $w-$ is 0. And the canonicalization has no effect. Next, we project the clock of $w-$ and extend DBM to include the clock of new enabled transition $t_7(x+)$. The time separation of $(v-,x+)$ is same as the time separation of $(v-,t_0)$, because $x+$ is currently enabled and it clock is same as the dummy clock $t_0$. Then, we advance all clocks by setting the 0th rows of DBM to their upper bounds. The canonicalization and normalization have no effect. In this new zone, transitions $v-$ and $x+$ can either be fired.
In this case, we choose to fire $x^+$ ahead and mark that we still need to backtrack $t_2(v^-)$ in other order. First, we do not need to restrict $x^+$’s clock, since $M[x^+, t_0] = 0$ and the lower bound of $x^+$ is 0. And the canonicalization has no effect. Next, we only project the clock of $x^+$, because firing $x^+$ does not result in any new enabled transition. In this new zone, transitions $v^-$ can be fired.

Since the lower bound of $v^-$ is 1, we need to restrict its clock in the DBM such that it cannot be less than 1. The canonicalization has no effect. Next, we project the clock of $v^-$ and extend DBM to include the clock of new enabled transition $t_3(y^+)$. Then, we advance $y^+$’s clock by setting $M[t_0, y^+]$ to $y^+$’s upper bound 3. The canonicalization and normalization have no effect. In this new zone, transitions $y^+$ can be fired.
Figure 2.17: A new zone after firing $t_3(y^+)$ based on Figure 2.16.

Since the lower bound of $y^+$ is 1, we need to restrict its clock in the DBM such that it cannot be less than 1. The canonicalization has no effect. Next, we project the clock of $y^+$ and extend DBM to include the clock of new enabled transition $t_9(u^-)$. Then, we advance $u^-$'s clock by setting $M[t_0, u^-]$ to $u^-$'s upper bound 5. The canonicalization and normalization have no effect. In this new zone, transitions $u^-$ can be fired.

We can go on tracking to fire enabled transition $u^-$. But in order to understand how to consider other orders, we backtrack to the zone after firing $t_6(w^-)$ which is shown in Figure 2.14.
Figure 2.18: Backtrack to zone in Figure 2.14 and generate a new zone after firing $t_2(v-)$. In this case, we choose to fire sequence $v-$ in other order. Since the lower bound of $v-$ is 1, we need to restrict its clock in the DBM such that it cannot be less than 1. The canonicalization has no effect. Next, we project the clock of $v-$ and extend DBM to include the clock of new enabled transition $t_3(y+)$. Then, we advance all clocks by setting the 0th rows of DBM to their upper bounds, canonicalize the DBM. The normalization has no effect. In this new zone, transitions $x+$ and $y+$ can either be fired.
In this case, we choose to fire sequence \{x+, y+\} ahead and mark that we still need to backtrack to fire sequence \{y+, x+\} in other order. First to fire \(x+\), which is shown on the left side. Since \(M[x+, t_0] = 0\) and the lower bound of \(x+\) is 0, we do not need to restrict its clock. And we only project the clock of \(x+\) because firing \(x+\) does not result in any new transition.

Next to fire \(y+\) based on the new generated zone, which is shown on the right side. Since the lower bound of \(y+\) is 1, we need to restrict its clock in the DBM such that it cannot be less than 1. Then we project the clock of \(y+\) and extend DBM to include the clock of new enabled transition \(t_9(u-).\) And, we advance \(u-\)’s clock by setting \(M[t_0, u-]\) to its upper bound 5. The canonicalization and normalization have no effect. Since this zone has been generated before, we can ignore this zone and backtrack to fire \(t_3(y+)\) based on zone in Figure 2.18.
Figure 2.20: An existed zone after firing $t_3(y^+)$ and $t_7(x^+)$ based on Figure 2.18.
In this case, We choose to fire sequence $\{y^+, x^+\}$ in other order.
First to fire $y^+$, which is shown on the left side. Since the lower bound of $y^+$ is 1, we need to restrict its clock in the DBM such that it cannot be less than 1. Then to canonicalize the DBM. After that, we only project the clock of $y^+$ because firing $y^+$ does not result in any new transition.
Next to fire $x^+$ based on the new generated zone, which is shown on the right side. Since $M[x^+, t_0] = -1$ and the lower bound of $x^+$ is 0, we do not need to restrict its clock. Then we project the clock of $x^+$ and extend DBM to include the clock of new enabled transition $t_9(u-)$. And, we advance $u-$’s clock by setting $M[t_0, u-]$ to its upper bound 5. The canonicalization and normalization have no effect. Since this zone has been generated before, we can ignore this zone and backtrack to fire $t_2(v-)$ based on zone in Figure 2.13.
Figure 2.21: Backtrack to zone in Figure 2.13 and generate a new zone after firing $t_2(v-)$. In this case, we choose to fire sequence $v-$ in other order. Since the lower bound of $v-$ is 1, we need to restrict its clock in the DBM such that it cannot be less than 1. Then to canonicalize the DBM. After that, we project the clock of $v-$ and extend DBM to include the clock of new enabled transition $t_3(y+)$. The time separation of $(w-, y+)$ is same as the time separation of $(w-, t_0)$, because $y+$ is currently enabled and it clock is same as the dummy clock $t_0$. Then, we advance all clocks by setting the 0th rows of DBM to their upper bounds, canonicalize the DBM. The normalization has no effect. In this new zone, transitions $w-$ and $y+$ can either be fired.
In this case, we choose to fire sequence $w-$ ahead and mark that we still need to backtrack to fire $t_3(y+)$ in other order. Since $M[w-, t_0] = -1$ and the lower bound of $w-$ is 0, we do not need to restrict its clock. Next, we project the clock of $w-$ and extend DBM to include the clock of new enabled transition $t_7(x+)$. The time separation of $(y+, x+)$ is same as the time separation of $(y+, t_0)$, because $x+$ is currently enabled and it clock is same as the dummy clock $t_0$. Then, we advance all clocks by setting the 0th rows of DBM to their upper bounds. The canonicalization and normalization have no effect. In this new zone, transitions $x+$ and $y+$ can either be fired.

![Diagram](image-url)

**Figure 2.22:** A new zone after firing $t_6(w-)$ based on Figure 2.21.
In this case, we choose to fire sequence \(\{x^+, y^+\}\) ahead and mark that we still need to backtrack to fire sequence \(\{y^+, x^+\}\) in other order.

First to fire \(x^+\). Since \(M[x^+, t_0] = 0\) and the lower bound of \(x^+\) is 0, we do not need to restrict its clock. And we only project the clock of \(x^+\) because firing \(x^+\) does not result in any new transition. Since this zone has been generated before, we can ignore firing \(y^+\) and backtrack to fire sequence \(\{y^+, x^+\}\) based on zone in Figure 2.22.
Figure 2.24: An existed zone after firing $t_3(y^+)$ and $t_7(x^+)$ based on Figure 2.22.

In this case, we choose to fire sequence \{y+, x+\} in other order. First to fire $y^+$, which is shown on the left side. Since the lower bound of $y^+$ is 1, we need to restrict its clock in the DBM such that it cannot be less than 1. The canonicalization has no effect. After that, we only project the clock of $y^+$ because firing $y^+$ does not result in any new transition.

Next to fire $x^+$ based on the new generated zone, which is shown on the right side. Since $M[x^+, t_0] = 0$ and the lower bound of $x^+$ is 0, we do not need to restrict its clock. Then we project the clock of $x^+$ and extend DBM to include the clock of new enabled transition $t_9(u-)$. And, we advance $u-$’s clock by setting $M[t_0, u-]$ to its upper bound 5. The canonicalization and normalization have no effect. Since this zone has been generated before, we can ignore this zone and backtrack to fire $t_3(y^+)$ based on zone in Figure 2.21.
Figure 2.25: Backtrack to zone in Figure 2.21 and generate two new zones after $t_3(y^+)$ and $t_6(w^-)$.

In this case, we choose to fire sequence $y^+$ in another order, which is shown on the left side. Since the lower bound of $y^+$ is 1, we need to restrict its clock in the DBM such that it cannot be less than 1. Then, to canonicalize the DBM. After that, we only project the clock of $y^+$ because firing $y^+$ does not result in any new transition.

Base on this new generated zone, transition $t_6(w^-)$ can be fired. Since $M[w^-, t_0] = -2$ and the lower bound of $w^-$ is 0, we do not need to restrict its clock. Then we project the clock of $w^-$ and extend DBM to include the clock of new enabled transition $t_7(x^+)$. And, we advance $x^+$’s clock by setting $M[t_0, x^+]$ to its upper bound 2. The canonicalization and normalization have no effect. Since this zone has been generated before, we finished all backtracking on this partial search.
Figure 2.26: A partial Zone Graph of LPN example in Figure 2.2 generated via Timing Analysis method.
Chapter 3

POSET Timing Analysis

3.1 Introduction

The zone approach in chapter 2 works well for many examples, but when there is a high degree of concurrency, it still generated more redundant zones for the same untimed state. Consider the simple example shown in Figure 3.1, (a) shows two LPN transitions $t_1$ and $t_2$ with time constraint $[1, 10]$. Firing these two transitions in different order results in different zones, which is shown in (b). The upper zone is found for the sequence $\{t_2, t_1\}$ while the lower zone is found for the sequence $\{t_1, t_2\}$. (c) shows the corresponding DBMs for these two zones. (Figure 3.2 and Figure 3.3 show the processes to generate zones from firing $t_1$ and $t_2$ in different order).

Even though these two sequences lead to the same untimed state, they result in different zones [9]. The circuit example used in chapter 2 also has this kind of concurrent transitions. In Figure 2.2, transition $w-$ and $v-$ are concurrent enabled after firing transition $z+$. Firing the sequence $\{w-, v-\}$ leads to a different zone from firing the sequence $\{v-, w-\}$, which is shown in Figure 2.26.

In fact, as the length of the sequence $n$ increases, the number of zones increases like $n!$, which would result in state space explosion. In order to deal with this problem, and separate concurrency from causality, we consider the partially ordered sets algorithm (POSET) rather than the linear sequences, which aims to analyze POSET matrix for fired transitions and generate new zone from this POSET matrix. Figure 3.4 shows the process of generating new zone based on POSET matrix, which resolves the problem in Figure 3.1. A graph representation of a POSET is shown in Figure 3.4(a). This POSET represents both the sequence $\{t_1, t_2\}$ and the sequence $\{t_2, t_1\}$. For each POSET we generate a POSET matrix which includes the time separation of
Figure 3.1: (a) A simple LPN example. (b) Zones for two sequences \(\{t_1, t_2\}, \{t_2, t_1\}\). (c) DBMs for each sequence.

\((r\) represents the reset clock which always takes the value 0).

each pair of transitions in the POSET. The final POSET matrix for sequence \(\{t_1, t_2\}\) and \(\{t_2, t_1\}\) is shown in Figure 3.4(b), and (e) shows the process to generate such POSET matrix. Transition \(t_1\) and \(t_2\) can be fired between 1 and 10 clock time after reset event \(r\). Either \(t_1\) or \(t_2\) could be fired as much as 9 clock times after the other. The generated zone from this POSET matrix is shown in Figure 3.4(c) and (d). Note that this zone includes both zones found in Figure 3.1(c). In this way, we can find one zone exactly for the one untimed state [12][9].

Section 3.2 presents the algorithms to derive the POSET matrix and to generate zone from the existing POSET matrix. Section 3.3 illustrates the POSET timing analysis algorithm on the example of Figure 2.26 in Chapter 2.

### 3.2 Algorithms

Same as the timing analysis, the set of reachable timed states forms a reachability graph. The main difference is that POSET approach maintains a POSET matrix with fired transitions and a corresponding zone with enabled transitions, which means we need to keep updated POSET matrix and new generated zone for each state during timed state space exploration. Figure 3.5 shows the whole process to verify the timed concurrency system.
Figure 3.2: The process to generate the zone for sequences \(\{t_1, t_2\}\).
Initially, transition \(t_1\) and \(t_2\) can be fired. In this case, we choose to fire \(t_1\) ahead. First to generate a new zone by restricting \(t_1\)'s low bound, then to canonicalize this zone. Next to fire \(t_1\) by projecting it, and extend new enabled transition \(t_1\). The time separation of \((r, t_1)\) is set to 0 and the time separation of \((t_1, t_2)\) is set to \([1, 10]\) just as the time separation of \((r, t_2)\).
Last to get the updated zone by advancing all enabled transitions’ time and canonicalizing the zone. The normalization has no effect.
After firing \(t_1\), \(t_2\) can be fired immediately. Then to fire \(t_2\) by projecting it, and extend new enabled transition \(t_2\). The time separation of \((r, t_2)\) is set to 0 and the time separation of \((t_1, t_2)\) is set to \([0, 9]\) just as the time separation of \((r, t_1)\). Then to get the updated zone by advancing all enabled transitions’ time. The canonicalization and normalization have no effect.
Initially, transition $t_1$ and $t_2$ can be fired. In this case, we choose to fire $t_2$ ahead. First to generate a new zone by restricting $t_2$'s low bound, then to canonicalize this zone. Next to fire $t_2$ by projecting it, and extend new enabled transition $t_2$. The time separation of $(r, t_2)$ is set to 0 and the time separation of $(t_1, t_2)$ is set to $[1, 10]$ just as the time separation of $(r, t_1)$. Last to get the updated zone by advancing all enabled transitions’ time and canonicalizing the zone. The normalization has no effect.

After firing $t_2$, $t_1$ can be fired immediately. Then to fire $t_1$ by projecting it, and extend new enabled transition $t_1$. The time separation of $(r, t_1)$ is set to 0 and the time separation of $(t_1, t_2)$ is set to $[0, 9]$ just as the time separation of $(r, t_2)$. Then to get the updated zone by advancing all enabled transitions’ time. The canonicalization and normalization have no effect.
Figure 3.4: (a) The POSET graph for sequences \{t_1, t_2\}, \{t_2, t_1\} of example in Figure 3.1. (b) POSET matrix. (c) Zone graph found for either \{t_1, t_2\} or \{t_2, t_1\} using POSET matrix. (d) Zone matrix. (e) The process to generate POSET matrix for these two sequences.
Figure 3.5: Verification on a timed concurrency System with \textit{poset} method
Let us presume that we have just encountered a new zone for an untimed state \( s \) and we must calculate the next zone. First, we determine which transitions are timing enabled in current zone, represented as \( enb(s) \), and choose one transition \( t \) to fire. Next, we need to update the POSET matrix. If there exist new enabled transition after firing \( t \), we extend POSET matrix with \( t \) and canonicalize it, then to project the unnecessary transitions. Otherwise, we only project the unnecessary transitions. Then, we need to generate a new zone based on the updated POSET matrix, canonicalize it to keep a canonical DBM, and normalize it to keep exploration finitely. Algorithm 5 is used to update the POSET matrix and find the corresponding zone.

**Algorithm 5: update-poset**

**Input:** \( P \): POSET matrix, \( enb \): enabled transitions, \( t_j \): newly fired transition

**Output:** A new zone \( M \) with updated POSET matrix

```plaintext
/*update POSET matrix*/
foreach transition \( t_i \) ∈ \( P \) do
  if \( t_j \) is causal to \( t_i \) then
    \( P[j, i] = -EFT(t_j) \);
    \( P[i, j] = LFT(t_j) \);
  else
    \( P[j, i] = \infty \);
    \( P[i, j] = \infty \);
  canonicalize(\( P \));

foreach transition \( t_i \) ∈ \( P \) do
  if any transition \( t_k \) ∈ \( enb \) is not causal to \( t_i \) then
    project(\( t_i \));
/*generate new zone*/
foreach transition \( t_k \) ∈ \( enb \) do
  \( M[k, 0] = 0 \);
  \( M[0, k] = LFT(t_k) \);
  foreach \( j = 1 \) to \( n \) do
    \( M[k, j] = P[k, j] \);
  canonicalize(\( M \));
  normalize(\( M \));
```

The algorithm takes the old POSET matrix \( P \), the newly fired transition \( t \) and the set of enabled transition \( enb \). If firing \( t \) result in a new enabled transition, we must update the POSET matrix and derive a new zone.
First, the newly fired transition $t_j$ is added to the POSET matrix (suppose insert to row $j$ and column $j$), and the timing separation of $t$ and other fired transitions in $P$ are also added. This time separation is calculated as follow:

- If the newly fired transition $t_j$ is causal to any fired transition $t_i$, the separation value of $(t_i, t_j)$ is set to $t_j$’s earliest firing time and latest firing time (i.e., $P[j, i] = -EFT(t_j)$, $P[i, j] = LFT(t_j)$). A transition $t_j$ is causal to $t_i$ only if firing $t_i$ can enabled transition $t_j$.

- If $t_j$ is not causal to $t_i$, which means these two transition are causal to the same previous fired transition, there is no relationship between them and their time separation is set to $\infty$.

The algorithm then canonicalizes the updated POSET matrix and projects the transition that are no longer needed. The transition can be projected once there no longer exists any enabled transitions enabled by this transition.

Second, this algorithm uses the updated POSET matrix to generate a new zone. The algorithm begins by setting the minimums to 0 (i.e. $M[i, 0] = 0$) and setting the maximum to their latest firing time (i.e. $M[0, i] = LFT(t_i)$).

It then copy the relevant time separations from the POSET matrix to the zone. Consider two transition $t_i$ and $t_j$, shown in Figure 3.6, are enabled by transition $t_k$ and $t_m$ separately. Suppose the firing time of $t_k$ is less than the firing time $t_m$ for a time bound $(1, 3)$, which means $t_k$ is fired before $t_m$ at lest 1 clock time and at most 3 clock time. Since $t_i$ is enabled once $t_k$ is fired and $t_j$ is enabled once $t_m$ is fired, the enabled time of $t_i$ is decided by the fired time of $t_k$ and the enabled time of $t_j$ is decided by the fired time of $t_m$. It means $t_i$ is enabled earlier than $t_j$ and the enabling time of $t_i$ is greater than the enabling time of $t_j$ for a time bound $(1, 3)$. So the $M[i, j]$ is found by copying $P[j, i]$ from the POSET matrix. Then, this zone need to be canonicalized and normalized.

If firing transition $t$ do not result in any new transition firing, The algorithm can simple project transition $t$ from POSET matrix and generated a
Figure 3.7: Initial zone with initial POSET matrix.
The POSET matrix is set to 0, represented as $r$. Since $r$ has only one enabled transition $t_{11}(z^+)$, the zone is set to two by two matrix with upper bounded time of transition $z^+$. The canonicalization and normalization have no effect. The transition $z^+$ can be fired in this initial zone because $M[0, z^+] \in [0, 3]$.

<table>
<thead>
<tr>
<th>POSSET</th>
<th>$r$</th>
<th>reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>$r$</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ZONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
</tr>
<tr>
<td>Canonicalize</td>
</tr>
<tr>
<td>Normalize</td>
</tr>
</tbody>
</table>

If this zone is searched before, we can backtrack to explore other traces. Otherwise, we can get the new generated transitions and find the new timed enabled transitions to fire.

### 3.3 POSET Timing Example

This section illustrate timed state space explosion with POSET method using circuit example shown in Figure 2.1. The time constraint for each signal is same as example in Chapter 2. Figure 3.7-Figure 3.20 explain how to generate zones based on POSET timing analysis step by step. The new zone with updated POSET is shown on the left side, while the right side shows the process tree. And Figure 3.21 shows the whole process graph analyzed above. For this partial search, there are exactly 10 zones generated for 10 untimed states in left side of Figure 2.9.
First to extend POSET matrix with the fired transition $z^+$, which enables new transition $t_6(w-)$ and $t_2(v-)$. The canonicalization has no effect and the unnecessary reset $r$ can be projected. Then to generate a new zone with all enabled transition $w-$ and $v-$. The time separation of $(w-,t_0)$ and $(v-,t_0)$ is set to $w-$‘s upper bound 2 and $v-$‘s upper bound 3. The time separation of $(v-,w)$ is set to 0 because these two transitions are enabled concurrently. Last to canonicalize the zone, and the normalization has no effect. In this new zone, transitions $v-$ and $w-$ can either be fired.
Figure 3.9: A new zone with updated POSET matrix after firing transition $t_6(w-)$ based on Figure 3.8.

In this case, we choose to fire $w-$ ahead and mark that we still need to backtrack $t_2(v-)$ in other order. First to extend POSET matrix with the fired transition $w-$, which enables new transition $t_7(x+)$. The canonicalization has no effect and no transition can be projected. Then to generate a new zone with all enabled transitions $x+$ and $v-$. The time separation of $(x+, t_0)$ and $(v-, t_0)$ is set to $x+$’s upper bound 2 and $v-$’s upper bound 3. The time separation of $(x+, v-)$ is set to $(0,2)$ according to the time separation of $(w-, z+)$ in the POSET. (The fired time of $w-$ can be greater than the fired time of $z+$ at least 0 and at most 2, which means the enabled time of $v-$ can be greater than the enabled time of $x+$ at least 0 and at most 2, because $v-$ is enabled once $z+$ is fired and $x+$ is enabled once $w-$ is fired.)

The canonicalization and normalization have no effect. In this new zone, transitions $x+$ and $v-$ can either be fired.
In this case, we choose to fire $x^+$ ahead and mark that we still need to backtrack $t_2(v^-)$ in other order. Since no new transition can be enabled when firing $x^+$, we do not add $x^+$ into POSET. And the unnecessary transition $w^-$ can be projected. Then to generate a new zone with enabled transition $v^-$. The canonicalization and normalization have no effect. In this new zone, transitions $v^-$ can be fired.

First to extend POSET matrix with the fired transition $v^-$, which enables new transition $t_3(y^+)$. The canonicalization has no effect. Next to project unnecessary transition $z^+$. Then to generate a new zone with new enabled transition $y^+$. The canonicalization and normalization have no effect. In this new zone, transitions $y^+$ can be fired.
Figure 3.12: A new zone with updated POSET matrix after firing transition $t_3(y+)$ based on Figure 3.11.
First to extend POSET matrix with the fired transition $y+$, which enables new transition $t_9(u-)$. The canonicalization has no effect. Next to project unnecessary transition $v-$. Then to generate a new zone with new enabled transition $u-$. The canonicalization and normalization have no effect. In this new zone, transitions $u-$ can be fired.
We can go on tracking to fire enabled transition $u-$. But in order to understand how to consider other orders, we backtrack to the zone after firing $t_6(w-)$ which is shown in Figure 3.9.
Figure 3.13: Backtrack to zone in Figure 3.9 and generate a new zone with updated POSET matrix after firing $t_2(v^-)$.

In this case, we consider to fire $v^-$ in other order. First to extend POSET matrix with the fired transition $v^-$, which enables new transition $t_0(y^+)$. Since transition $v^-$ and $w^-$ are causal to $z^+$, the time separation of $(v^-, z^+)$ is constraint to $v^-$’s low bound and upper bound, the time separation of $(v^-, w^-)$ is set to $\infty$. Next to canonicalize this updated POSET matrix and project the unnecessary transition $z^+$. Then to generate a new zone with all enabled transitions $x^+$ and $y^+$. The time separation of $(x^+, t_0)$ and $(y^+, t_0)$ is set to $x^+$’s upper bound 2 and $y^+$’s upper bound 3. The time separation of $(x^+, y^+)$ is set (3,1) according to the time separation of $(w^-, v^-)$ in the POSET. (The fired time of $v^-$ can be greater than the fired time of $w^-$ at least -1 and at most 3, which means the enabled time of $x^+$ can be greater than the enabled time of $y^+$ at least -1 and at most 3, because $y^+$ is enabled once $v^-$ is fired and $x^+$ is enabled once $w^-$ is fired.) Then to canonicalize this zone and the normalization has no effect. In this new zone, transitions $x^+$ and $y^+$ can either be fired.
Figure 3.14: An existed zone with updated POSET matrix after firing $t_7(x^+)$ based on Figure 3.13.

In this case, we choose to fire $x^+$ ahead and mark that we still need to backtrack $t_3(y^+)$ in other order. Since no new transition can be enabled after firing $x^+$, we do not add $x^+$ into POSET. Next to project unnecessary transition $w^-$. Then to generate a new zone with enabled transition $y^+$. The canonicalization and normalization have no effect. Since this new zone has been generated before, we can ignore this zone and backtrack to fire $t_3(y^+)$ based on zone in Figure 3.13.
Figure 3.15: Backtrack to zone in Figure 3.13 and generate a new zone with updated POSET matrix after firing $t_3(y+)$. In this case, we choose to fire $y+$ in other order. Since no new transition can be enabled when firing $y+$, we do not add $y+$ into POSET. Next to project unnecessary transition $v-$. Then to generate a new zone with enabled transition $x+$. The canonicalization and normalization have no effect. In this new zone, transitions $x+$ can be fired.
In this case, transition $x^+$ can be fired. First to extend POSET matrix with the fired transition $x^+$, which enables new transition $t_9(u^-)$. The canonicalization has no effect. Next to project unnecessary transition $w^-$. Then to generate a new zone with new enabled transition $u^-$. The canonicalization and normalization have no effect. Since this zone has been generated before, we can ignore this zone and backtrack to fire $t_2(v^-)$ based on zone in Figure 3.8.
Figure 3.17: Backtrack to zone in Figure 3.8 and generate a new zone with updated POSET matrix after firing $t_2(v^-)$.
In this case, we choose to fire $v^-$ in another order. First to extend POSET matrix with the fired transition $v^-$, which enables new transition $t_3(y^+)$. The canonicalization has no effect and no transition can be projected. Then to generate a new zone with all enabled transition $w^-$ and $y^+$. The time separation of $(w^-,t_0)$ and $(y^+,t_0)$ is set to $w^-$’s upper bound 2 and $y^+$’s upper bound 3. The time separation of $(w^-,y^+)$ is set (3,1) according to the time separation of $(z^+,v^-)$ in the POSET matrix. (The fired time of $v^-$ can be greater than the fired time of $z^+$ at least -1 and at most 3, which means the enabled time of $w^-$ can be greater than the enabled time of $y^+$ at least -1 and at most 3, because $y^+$ is enabled once $v^-$ is fired and $w^-$ is enabled once $z^+$ is fired.) The canonicalization and normalization have no effect. In this new zone, transitions $w^-$ and $y^+$ can either be fired.
In this case, we choose to fire transition $w^-$ ahead and mark that we still need to backtrack $t_3(y^+)$ in other order. First to extend POSET matrix with the fired transition $w^-$, which enables new transition $t_7(x^+)$. Since transition $v^-$ and $w^-$ are causal to $z^+$, the time separation of $(w^-, z^+)$ is constraint to $w^-$’s low bound and upper bound, the time separation of $(v^-, w^-)$ is set to $\infty$. Next to canonicalize this updated POSET matrix and project the unnecessary transition $z^+$. Then to generate a new zone with all enabled transitions $x^+$ and $y^+$. The time separation of $(x^+, t_0)$ and $(y^+, t_0)$ is set to $x^+$’s upper bound 2 and $y^+$’s upper bound 3. The time separation of $(x^+, y^+)$ is set (3,1) according to the time separation of $(w^-, v^-)$ in the POSET. (The fired time of $v^-$ can be greater than the fired time of $w^-$ at least -1 and at most 3, which means the enabled time of $x^+$ can be greater than the enabled time of $y^+$ at least -1 and at most 3, because $y^+$ is enabled once $v^-$ is fired and $x^+$ is enabled once $w^-$ is fired.) Then to canonicalize this zone and the normalization has no effect. Since this zone has been generated before, we can ignore this zone and backtrack to fire $t_3(y^+)$ based on zone in Figure 3.17.
Figure 3.19: Backtrack to zone in Figure 3.17 and generate a new zone with updated POSET matrix after firing $t_3(y^+)$. In this case, we choose to fire $y^+$ in other order. Since no new transition can be enabled when firing $y^+$, we do not add $y^+$ into POSET matrix. Next to project unnecessary transition $v^-$. Then to generate a new zone with enabled transition $w^-$. The canonicalization and normalization have no effect. In this new zone, transitions $w^-$ can be fired.
First to extend POSET matrix with the fired transition $w-$, which enables new transition $t_7(x+)$. The canonicalization has no effect. Next to project unnecessary transition $z+$. Then to generate a new zone with new enabled transition $x+$. The canonicalization and normalization have no effect. Since this zone has been generated before, we finished all backtracking on this partial search.
Figure 3.21: A partial poset-zone Graph of LPN example in Figure 2.2 generated via POSET Timing Analysis method.
Chapter 4

Improvement of POSET Timing Analysis

4.1 Introduction

The POSET analysis method in chapter 3 shows an efficient timing analysis approach, which could find exactly one zone for the one untimed state to improve the state space explosion. For each zone generation step, we need to keep the POSET matrix to record the time separations of the fired but still need to be used transitions, and keep the new zone to record the enabled transition’s time separation. Actually, this method still cost much more memory space to keep such information.

In order to deal with this problem, we introduce an improvement method based on the zone analysis approach with POSET thought. In this case, we do not need to have extra step to maintain and update the POSET matrix, only update and keep the zone information for each zone generation step. Section 4.2 shows the algorithms to realize this method.

4.2 Algorithms

Same as the timing analysis, we use a reachability graph to represent the set of reachable timed states. Figure 3.5 shows the whole process to verify the timed concurrency system under the improved method.

Let us presume that we have just encountered a new zone for an untimed state \( s \) and we must calculate the next zone. First, we determine which transitions are timing enabled in current zone, represent as \( enb(s) \), and choose one transition \( t \) to fire. Next, we need to generate a new zone. If firing \( t \) results in new enabled transition set \( newEnb \), we extend existed zone with
newEnb and analyze the time separation of all enabled transitions, then canonicalize it and project the unnecessary transitions. Otherwise, we only project the unnecessary transitions. At last, we need to advance the zone, canonicalize it to keep a canonical DBM, and normalize it to keep exploration finitely. Algorithm 6 is used to generate a new zone for each state during timed state space exploration.

The algorithm takes the old zone, the current enabled transition set \(enb\), and the newly enabled transition set \(newEnb\). If firing \(t \in enb\) result in a new enabled transition, we must extend a new zone.

First, the newly enabled transition set \(newEnb\) is added to the zone, and the timing separation of \(newEnb\) and other enabled transitions in \(M\) are also added. The time separation is calculated as follow:

- For each pair of newly enabled transition, the time separation is set to 0 because of concurrent enabled condition.
- For each newly enabled transition \(t_i\), the time separation of \(t_i\) and a dummy clock \(t_0\) is set to 0.
- If the newly fired transition \(t_j\) is causal to any fired transition \(t_i\), the separation value of \((t_i, t_j)\) is set to \(t_i\)’s earliest firing time and latest firing time (i.e., \(M[i,j] = -EFT(t_i), M[j,i] = LFT(t_i)\)). Because \(t_i\) is enabled earlier than \(t_j\) which means the enable time of \(t_i\) is larger than the enable time of \(t_j\) for a time bound \((EFT, LFT)\) of \(t_i\). A transition \(t_j\) is causal to \(t_i\) only if firing \(t_i\) can enabled transition \(t_j\).
- If \(t_j\) is not causal to \(t_i\), which means there is no relationship between these two transition and their time separation is set to \(\infty\).

Next, The algorithm partially canonicalizes the updated zone, the detail algorithm is shown in Algorithm 7. In this case, the algorithm only need to canonicalizes the enabled transitions except the dummy clock \(t_0\). Then the algorithm projects the transition that are no longer needed. The transition can be projected once there no longer exists any new enabled transitions enabled by this transition. Then, this zone is advanced, canonicalized and normalized.

If firing transition \(t\) do not result in any new transition firing, the algorithm can simple project transition \(t\) from zone, then to advance, canonicalize and normalize this zone.

If this zone is searched before, we can backtrack to explore other traces. Otherwise, we can get the new generated transitions and find the new timed enabled transitions to fire.
Figure 4.1: Verification on a timed concurrency System with improved POSET method
Algorithm 6: update-zone($M, enb, newEnb$)

**Input:** $M$: zone, $enb$: enabled transitions in $M$, $newEnb$: new enabled transitions set

**Output:** A new zone $M'$

1. $M' = M$; if $newEnb \neq$ then
2. /*add new enabled transitions*/
3. foreach $t_i \in newEnb$ do
4.   $M'[0,i] = 0$;
5.   $M'[i,0] = 0$;
6. /*time separation of new enabled transitions*/
7. foreach $t_i \in newEnb$ do
8.   foreach $t_j \in newEnb$ do
9.     $M'[i,j] = 0$;
10.    $M'[j,i] = 0$;
11. /*partial canonicalization*/
12.   canonicalize($M', 1$);
13. project($M'$);
14. /*Advance time*/
15. foreach $t_i \in (newEnb \cup enb)$ do
16.   $M'[i,0] = 0$;
17.   $M'[0,i] = LFT(t_i)$;
18. canonicalize($M', 0$);
19. normalize($M'$);
Algorithm 7: canonicalize($M, index$)

**Input:** $M$: zone, $index$: index of each transition in $M$

**Output:** A tightest matrix for the same zone

1. **foreach** $i = index$ to $n - 1$ do
2.   **foreach** $j = index$ to $n - 1$ do
3.     **foreach** $k = index$ to $n - 1$ do
4.       if $M[j, k] > M[j, i] + M[i, k]$ then

Figure 4.2: Initial Zone.
The initial state has only one clock for enabled transition $t_{11}(z+)$, so the initial DBM is only two by two matrix with all zeros initially. Then to advance time by setting the top row for $t_{11}(z+)$ to the maximum value 3. The canonicalization and normalization have no effect. Transition $z+$ can be fired in this initial zone because $M[0, z+] \in [0, 3]$.

### 4.3 Example

This section illustrate timed state space explosion with improved POSET method using circuit example shown in Figure.2.1. The time constraint for each signal is same as example in Chapter 2 and Chapter 3. Figure 4.2- Figure 4.15 explain how to generate zones based on improved algorithm step by step. And Figure 4.16 shows the whole process graph analyzed above. The result is same as Figure 3.21, but only store zone information.

Each generated zone is same as zone in Figure 3.21 in Chapter 3.
Figure 4.3: A new zone after firing $t_1 1(z^+)$ based on Figure 4.2.
Firing $z^+$ result in two new enabled transitions $t_2(v^-)$ and $t_6(w^-)$, so we need to extend DBM and analyze the time difference between each clock ahead. First, the time differences of $(v^-, t_0)$ and $(w^-, t_0)$ are set to 0 because of the currently enabled condition. Second, the time difference of $(v^-, w^-)$ is set to 0, because $t_2(v^-)$ and $t_6(w^-)$ are enabled concurrently. Then, the time difference of $(z^+, w^-)$ and $(z^+, v^-)$ are set to $z^+$’s time constraint, because these transitions can only be enabled after firing $z^+$. It means the enabled clock time of $z^+$ is greater than these two transitions at least 0 and at most 3. After analyzing the time difference, we partially canonicalize the DBM of $z^+, w^-, v^-$ (the canonicalization have no effect), and project unnecessary transition $z^+$. Then we advance the enabled transition’s clock time, canonicalize and normalize the zone. In this new zone, two enabled transitions $t_2(v^-)$ and $t_6(w^-)$ can either be fired.

61
Figure 4.4: A new zone after firing $t_6(w-)$ based on Figure 4.3.

In this case, we choose to fire $w-$ ahead and mark that we still need to backtrack $t_2(v-)$ in other order. Firing transition $w-$ result in new enabled transition $t_7(x+)$, so we need to extend DBM and analyze the time difference between $x+$ and each other clock. First, the time differences of $(x+, t_0)$ is set to 0 because of the currently enabled condition. Second, the time difference of $(x+, w-)\) is set to $w-$’s time constraint, because $x+$ is enabled after firing $w-$ which means the enabled clock time of $w-$ is greater than $x+$ at least 0 and at most 2. Then, the time difference of $(x+, v-)$ is set to $\infty$, since there is no relationship between them. After analyzing the time difference, we partially canonicalize the DBM of $w-, v-, x+$, and project unnecessary transition $w-$. Then we advance the enabled transition’s clock time. The canonicalization and normalization have no effect. In this new zone, transition $x+$ and $v-$ can either be fired.

<table>
<thead>
<tr>
<th>transition</th>
<th>$t_0$</th>
<th>$w-$</th>
<th>$v-$</th>
<th>$x+$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$w-$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$v-$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$x+$</td>
<td>0</td>
<td>2</td>
<td>$\infty$</td>
<td>0</td>
</tr>
</tbody>
</table>
Figure 4.5: A new zone after firing $t_7(x+)$ based on Figure 4.4.
In this case, we choose to fire $x+$ ahead and mark that we still need to backtrack $t_2(v-)$ in other order. Since no new transition enabled after firing $x+$, we can project $x+$ directly. Then we advance $v-$'s clock time. The canonicalization and normalization have no effect. In this new zone, transition $v-$ can be fired.
Figure 4.6: A new zone after firing $t_2(v-)$ based on Figure 4.5.
Firing transition $v-$ result in new enabled transition $t_3(y+)$, so we need to extend DBM and analyze the time difference between $y+$ and each other clock. First, the time differences of $(y+, t_0)$ is set to 0 because of the currently enabled condition. Second, the time difference of $(v-, y+)$ is set to $v-$’s time constraint, because $y+$ is enabled after firing $v-$ which means the enabled clock time of $v-$ is greater than $y+$ at least 1 and at most 3. After analyzing the time difference, we partially canonicalize the DBM of $v-, y+$ (the canonicalization have no effect), and project unnecessary transition $v-$. Then we advance the enabled transition $y+$’s clock time. The canonicalization and normalization have no effect. In this new zone, transition $y+$ can be fired.
Figure 4.7: A new zone after firing \( t_3(y+) \) based on Figure 4.6.

Firing transition \( y+ \) result in new enabled transition \( t_9(u-) \), so we need to extend DBM and analyze the time difference between \( u- \) and each other clock. First, the time differences of \((u-, t_0)\) is set to 0 because of the currently enabled condition. Second, the time difference of \((y+, u-)\) is set to \( y+'s \) time constraint, because \( u- \) is enabled after firing \( y+ \) which means the enabled clock time of \( y+ \) is greater than \( u- \) at least 1 and at most 3. After analyzing the time difference, we partially canonicalize the DBM of \( y+, u- \) (the canonicalization have no effect), and project unnecessary transition \( y+ \). Then we advance the enabled transition \( u- \)'s clock time. The canonicalization and normalization have no effect. In this new zone, transition \( u- \) can be fired.

We can go on tracking to fire enabled transition \( u- \). But in order to understand how to consider other orders, we backtrack to the zone after firing \( t_6(w-) \) which is shown in Figure 4.4.
Figure 4.8: Backtrack to Zone in Figure 4.4 and generate new zone after firing $t_2(v^-)$.

In this case, we consider to fire $v^-$ in other order. Firing transition $v^-$ result in new enabled transition $t_3(y^+)$, so we need to extend DBM and analyze the time difference between $y^+$ and each other clock. First, the time difference of $(y^+, t_0)$ is set to 0. Second, the time difference of $(v^-, y^+)$ is set to $v^-$’s time constraint, which means the enabled clock time of $v^-$ is greater than $y^+$ at least 1 and at most 3. Then, the time difference of $(x^+, y^+)$ is set to $\infty$, since there is no relationship between them. After analyzing the time difference, we partially canonicalize the DBM of $v^-, x^+, y^+$, and project transition unnecessary $v^-$. Then we advance the enabled transitions’ clock time, and canonicalize the zone. The normalization has no effect. In this new zone, transition $x^+$ and $y^+$ can either be fired.
In this case, We choose to fire \( x^+ \) ahead and mark that we still need to backtrack \( t_3(y^+) \) in other order. Since no new transition can be enabled after firing \( x^+ \), we can project \( x^+ \) directly. Then we advance \( y^+ \)’s clock time. The canonicalization and normalization have no effect. In this new zone, transition \( y^+ \) can be fired. Since this new zone has been generated before, we can ignore this zone and backtrack to fire \( y^+ \) based on zone in Figure 4.8.

\[
\begin{array}{|c|c|c|}
\hline
\text{Project} & t_0 & y^+ \\
\text{Advance} & t_0 & 0 & 3 \\
\text{Canonicalize} & y^+ & 0 & 0 \\
\text{Normalize} & & & \\
\hline
\end{array}
\]

Figure 4.9: An existed zone after firing \( t_7(x^+) \) based on Figure 4.8.

In this case, We choose to fire \( x^+ \) in other order. Since no new transition can be enabled after firing \( y^+ \), we can project \( y^+ \) directly. Then we advance \( y^+ \)’s clock time. In this new zone, transition \( x^+ \) can be fired. Since this new zone has been generated before, we can ignore this zone and backtrack to fire \( y^+ \) based on zone in Figure 4.8.

\[
\begin{array}{|c|c|c|}
\hline
\text{Project} & t_0 & x^+ \\
\text{Advance} & t_0 & 0 & 2 \\
\text{Canonicalize} & x^+ & 0 & 0 \\
\text{Normalize} & & & \\
\hline
\end{array}
\]

Figure 4.10: Backtrack to zone in Figure 3.13 and generate a new zone after firing \( t_3(y^+) \) based on Figure 4.8.

In this case, We choose to fire \( y^+ \) in other order. Since no new transition can be enabled after firing \( y^+ \), we can project \( y^+ \) directly. Then we advance \( x^+ \)’s clock time. In this new zone, transition \( x^+ \) can be fired.
Figure 4.11: An existed zone after firing $t_7(x+)$ based on Figure 4.10. Firing transition $x+$ result in new enabled transition $t_9(u-)$, so we need to extend DBM and analyze the time difference between $u-$ and each other clock. First, the time differences of $(u-, t_0)$ is set to 0. Second, the time difference of $(x+, u-)\) is set to $x+'$s time constraint, because $u-$ is enabled after firing $x+$ which means the enabled clock time of $x+$ is greater than $u-$ at least 0 and at most 2. After analyzing the time difference, we partially canonicalize the DBM of $x+, u-$ (the canonicalization have no effect), and project unnecessary transition $x+$. Then we advance the enabled transition $u-$’s clock time. The canonicalization and normalization have no effect. In this new zone, transition $u-$ can be fired. Since this zone has been generated before, we can ignore this zone and backtrack to fire $t_2(v-)$ based on zone in Figure 4.3.
Figure 4.12: Backtrack to zone in Figure 4.3 and generate a new zone after firing $t_2(v-)$.

In this case, we choose to fire $v-$ in other order. Firing transition $v-$ result in new enabled transition $t_3(y+)$, so we need to extend DBM and analyze the time difference between $y+$ and each other clock. First, the time differences of $(y+, t_0)$ is set to 0. Second, the time difference of $(v-, y+)$ is set to $v-$'s time constraint, because $y+$ is enabled after firing $v-$ which means the enabled clock time of $v-$ is greater than $y+$ at least 1 and at most 3. Then, the time difference of $(w-, y+)$ is set to $\infty$, since there is no relationship between them. After analyzing the time difference, we partially canonicalize the DBM of $w-, v-, y+$, and project unnecessary transition $v-$. Then we advance the enabled transition’s clock time and canonicalize the zone. The normalization has no effect. In this new zone, transition $w-$ and $y+$ can either be fired.
Figure 4.13: An existed zone after firing $t_6(w-)$ based on Figure 4.12.

In this case, we choose to fire transition $w-$ ahead and mark that we still need to backtrack $t_3(y+)$ in other order. Firing transition $w-$ result in new enabled transition $t_7(x+)$, so we need to extend DBM and analyze the time difference between $x+$ and each other clock. First, the time difference of $(x+, t_0)$ is set to 0. Second, the time difference of $(x+, w-)$ is set to $w-$’s time constraint, which means the enabled clock time of $w-$ is greater than $x+$ at least 0 and at most 2. Then, the time difference of $(x+, y+)$ is set to $\infty$, since there is no relationship between them. After analyzing the time difference, we partially canonicalize the DBM of $w-, y+, x+$, and project unnecessary transition $w-$. Then we advance the enabled transition’s clock time. The canonicalization and normalization have no effect. Since this zone has been generated before, we can ignore this zone and backtrack to fire $y+$ based on zone in Figure 4.12.
Figure 4.14: Backtrack to zone in Figure 4.12 and generate a new zone after firing $t_3(y^+)$. In this case, we choose to fire $y^+$ in other order. Since no new transition can be enabled after firing $y^+$, we can project $y^+$ directly. Then we advance $w^-$’s clock time. In this new zone, transition $w^-$ can be fired.
Firing $w$ result in new enabled transition $t_7(x+)$. So we need to extend DBM and analyze the time difference between $x+$ and each other clock. First, the time difference of $(x+, t_0)$ is set to 0. Then, the time difference of $(x+, w-) is set to $w-$’s time constraint, which means the enabled clock time of $w-$ is greater than $x+$ at least 0 and at most 2. After analyzing the time difference, we partially canonicalize DBM of $w-, x+$ (the canonicalization has no effect), and project unnecessary $w-$. Then we advance the enabled transition’s clock time. The canonicalization and normalization have no effect. Since this zone has been generated before, we finished all backtracking on this partial search.
Figure 4.16: A partial Zone Graph of LPN example in Figure ?? generated via improved POSET Timing Analysis method.
Chapter 5

Conclusion

Labeled Petri-Nets are a common modeling formalism for asynchronous designs [20]. In this report, we use Labeled Petri-Nets to model the designs and use reachability analysis to traverse all possible states and transitions during model checking.

In order to verify real-time system with timed petri-net structure correctly, we presents an idea to use zone approach in timed Labeled Petri-Nets. For each untimed state, we analyze the timed separation of each enabled transition pairs, firing the enabled transition whose clock time is under its time constraint and derive new zone which including all clock information of enabled transitions. To describe this idea, we give a circuit example and show the zone generation process step by step during reachability analysis.

The zone approach works well for many examples, but when there is a high degree of concurrency, It still generated more redundant zones for the same untimed state. In order to deal with this problem and get the exact one final zone for firing the same transitions in different order, we introduces an idea to use zone approach with POSET thought, which combine the zone approach and the POSET approach and only maintain the zone space. Proved by the experiment of the circuit example, this idea can not only reduce the number of zone by avoid generate different zones when firing transitions in different order, but also keep less information by combine the POSET matrix with zone.

Based on these two improved thought, we can verify real-time asynchronous system with little state space cost.

Acknowledgements
Bibliography


