Reachability Problem of Timed Petri Nets

Ahmet Yılmaz and Mürvet Kırcı

Electronics and Communication Engineering Department, İstanbul Technical University, Turkey
ahmet.yilmaz@beko.com.tr, ucerm@itu.edu.tr

Abstract- Deterministic timed transitions are studied in this paper. A new method is developed for reachability analysis of timed Petri nets. Reachability analysis is a strong analysis method to determine the states in the discrete event systems.

With time added, real time analyzing is possible. Developed model overcomes some limitations of timed Petri nets’ reachability analysis. Our method is supported by C software and real time analysis of discrete event systems is simulated.

Keywords- Timed petri nets, reachability tree, performance analysis.

I. INTRODUCTION

In real life, there are lots of discrete event dynamic systems (DEDS) like factory production processes, communication systems, traffic control systems, military control and command systems. Petri nets are one of the most popular analysis methods presenting a simple graphical presentation and powerful mathematical structure. Since time independent Petri net analysis is incapable of performance analysis, timed Petri nets modeling is developed.

Time in Petri nets is first introduced by Ramchandani. In the first developed Petri net, time is assigned to transitions [1, 2, 3, 5, 7]. As transitions correlated events, there is a duration between starting time and finishing time of an event [4]. In this paper Ramchandani’s approach will be used. This model is called “Transitions timed Petri nets” [4, 6, 7]. In the transitions timed Petri nets (TTPN), tokens are moving in two steps. In the first step, firing starts instantaneously, if a transition is enabled, and input places of the fired transitions lose tokens. This state is preserved for a firing delay duration assigned to the transition. At the second step output places earn tokens [8, 9, 10].

State vector: State vector shows the token number for each place.

\[ M = \{ m_i \in M \mid m_i = M(P_i) \} \]

Transition vector: Transition vector is sorted following the priority assigning to avoid conflict.

\[ T = \{ t_1, t_2, ..., t_n \} \]

Input and Output matrix:
Input matrix indicates the weight of the output arcs from places to transitions.

\[ I = \{ i_{ij} \in I \mid i_{ij} = w(P_i, T_j) \} \]

Output matrix indicates the weight of the input arcs from transitions to places.

\[ Q = \{ q_{ij} \in Q \mid q_{ij} = w(T_j, P_i) \} \]

Firing duration vector: This vector indicates the firing duration for each transition.

\[ D = [d_1, d_2, ..., d_n] \]
\[ D : \{ d_j \in D \mid d_j = \alpha_j \} \] (4)

\( j=1,\ldots,n \) and \( \alpha_j \) is the firing duration of \( t_j \).

**Sequential firing duration vector:** This vector indicates the duration between two sequential firings for a transition. Two firings for a transition are not allowed at the same time.

\[ DG = [d_{g1}, d_{g2}, \ldots, d_{gn}] \]

\[ DG : \{ d_j \in DG \mid d_j = \beta_j \} \] (5)

\( j=1,\ldots,n \) and \( \beta_j \) is the sequential firing duration of \( t_j \).

**Firing end time storage matrix:** This matrix is used to store the ending time of a fired transition.

\[ Z : \{ z_{ij} \in Z \mid z_{ij} = t + d_j \} \] (6)

\( i=1,\ldots,h \) and \( j=1,\ldots,n \) and \( t \) is the time

**Sequential firing start time storage matrix:** This matrix is used to store the sequential starting time of a transition.

\[ ZZ : \{ z_{zij} \in ZZ \mid z_{zij} = t + d_j \} \] (7)

\( i=1,\ldots,h \) and \( j=1,\ldots,n \) and \( t \) is the time

**Sequential firing coefficient vector:** This vector indicates the sequential firing number for a transition.

\[ ZE = [ze_1, ze_2, \ldots, ze_n] \]

\[ ZE = \{ ze_j \in ZE \mid m_i \geq i_{ij} \Rightarrow ze_j = ze_j + 1 \} \] (8)

\( i=1,\ldots,m \) and \( j=1,\ldots,n \)

**Enable Transition vector:** This vector is used to determine conflict cases. If a transition is enabled, "1" is assigned to that transition.

\[ T_E = [t_{E1}, t_{E2}, \ldots, t_{En}] \]

\[ T_E = \{ E_j \in T_E \mid m_i \geq i_{ij} \Rightarrow t_{E_j} = 1 \} \] (9)

\( j=1,\ldots,n \)

**Last firing time storage vector:** This vector indicates the last firing time of a transition and it is used to control sequential firings.

\[ Sa = [sa_1, sa_2, \ldots, sa_n] \]

\[ Sa = \{ sa_j \in Sa \mid m_i \geq i_{ij} \Rightarrow sa_j = t \} \] (10)

\( i=1,\ldots,n \) and \( j=1,\ldots,m \) and \( t \) is time

**Conflict:** If a firing of an enabled transition prevents the firing of another transition, conflict occurs.

In Fig 2, if transition T1 starts firing, the firing of T2 will be prevented. Conflict is determined by comparing two enabled transitions’ vectors. First vector \( T_{E1} = [1, 1, 1] \) is calculated without removing tokens from the input places to determine which transitions are enabled. Then second vector \( T_{E2} = [1, 0, 1] \) is calculated by following priority assigning and removing tokens for fired transitions. After comparing two vectors, it is shown that transition T2 is effected from conflict.

### III. REACHABILITY ANALYSIS ALGORITHM

The newly developed reachability analysis has 3 steps. In the first step instantaneous firings will be initiated. In the following step sequential firings will be initiated. The last step involves firing of transitions.

**The reachability analysis algorithm** given has the following advantages:

**Places can earn tokens before they lose all of their tokens:** This is modeled using an additional vector \( ZB \) with the same dimension as the state vector M. If places earn tokens before they lose all of their tokens, (i) a flag kn is set to 1, (ii) newly earned tokens are added to ZB, and (iii) newly earned tokens are shared with transitions’ firings.

**Deadlock and conflict is detected:** Deadlock is detected by checking matrices Z and ZZ. Deadlock time ZZ and state is included in the output file. Conflict is detected by comparing enabled transitions vectors. Additionally conflict information is included in the output file.

**Number of tokens, weight of input and output arcs are not limited to one:** Reachability analysis algorithm is designed to avoid these limitations.

**Firing sequences is included in the output file:** This information is useful to check the reachability graph and it indicated the marked language for the analyzed system.

A new firing may be allowed before other firings are finished: This limitation is removed during the reachability analysis.
Sequential firing of transitions is allowed: This is allowed by assigning a delay between two sequential firings. This information is indicated in DG vector.

Developed Reachability Analysis Algorithm

START
get the input values: M, I, Q, D, DG, analysis time
ANALYSE(M,t)
enabling checking for M
enabling_i_transition =false;

FOR  \( i =1 \) to \( n \)
   IF  all the member of \( M[0..m] \) array is bigger or equal than the member of \( I[1..m][i] \)
enabling_i_transition =true
   ELSE
      i=i+1
   END IF
END FOR

REPEAT
do instantaneous firing process control
   IF  enabling_i_transition =true and \( t-sa[i]>dg[i] \)
      start instantaneous firing
   write firing end time to Z matrix
   write the firing time to \( sa[i] \)
   output (M, t)
   ELSE
      write the firing starting time to ZZ matrix.
   END IF
UNTIL checking all transitions.

REPEAT
do deadlock control
   IF Z matrix and ZZ matrix has no nonzero element
      system locked
      exit program
   ELSE
      go on analyzing.
   END IF
UNTIL checking all elements of Z and ZZ matrixes.

REPEAT
do sequential firing control.
   IF knt=1
      calculate sequential firing correlation \( ZE \)
      for all transitions by using \( ZB \) vector
      \( mt=1; \)
   ELSE
      calculate sequential firing coefficient \( ZE \)
      for all transitions by using \( M \) matrix.
   END IF
UNTIL checking all transitions.

IV. PRACTICAL APPLICATION EXAMPLE

As an example we shall model an intercity bus voyage using developed reachability analysis.
Every terminal has one input and one output way. States and transitions should be determined in the model first.

Table (1): Places and transitions for intercity bus voyage

<table>
<thead>
<tr>
<th>Transitions</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Istanbul-Ankara voyage</td>
</tr>
<tr>
<td>T2</td>
<td>Istanbul-Izmir voyage</td>
</tr>
<tr>
<td>T3</td>
<td>Ankara-Istanbul voyage</td>
</tr>
<tr>
<td>T4</td>
<td>Ankara-Izmir voyage</td>
</tr>
<tr>
<td>T5</td>
<td>Izmir-Ankara voyage</td>
</tr>
<tr>
<td>T6</td>
<td>Izmir-Ankara voyage</td>
</tr>
<tr>
<td>T7</td>
<td>Passengers getting out and bus preparing to start a new voyage in Istanbul terminal</td>
</tr>
<tr>
<td>T8</td>
<td>Passengers getting out and bus preparing to start a new voyage in Ankara terminal</td>
</tr>
<tr>
<td>T9</td>
<td>Passengers getting out and bus preparing to start a new voyage in Izmir terminal</td>
</tr>
</tbody>
</table>

Places

<table>
<thead>
<tr>
<th>Places</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Bus arrived to Istanbul terminal</td>
</tr>
<tr>
<td>P2</td>
<td>Bus is ready to start voyage in Istanbul terminal</td>
</tr>
<tr>
<td>P3</td>
<td>Bus arrived to Izmir terminal</td>
</tr>
<tr>
<td>P4</td>
<td>Bus is ready to start voyage in Izmir terminal</td>
</tr>
<tr>
<td>P5</td>
<td>Bus arrived to Ankara terminal</td>
</tr>
<tr>
<td>P6</td>
<td>Bus is ready to start voyage in Ankara terminal</td>
</tr>
</tbody>
</table>

Timed Petri net model is designed.

Figure (4): Timed Petri net graph for intercity bus voyage

Input information is given below for reachability analysis:

\[
I = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
Q = \begin{bmatrix}
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
D = \begin{bmatrix}
7 & 9 & 7 & 8 & 9 & 8 & 1 & 1 & 1 & 1
\end{bmatrix}
\]

A practical example is designed with the developed algorithm. It is possible to check the states of the system in the time domain during the operations.

VI. CONCLUSION

In this paper, a new reachability analysis algorithm is developed for Timed Transition Petri Nets. This algorithm is supported with C programming in DEVC++ software improved environment. Developed algorithm is applied to some practical applications with 44 places and 24 transitions and 12 places and 10 transitions, respectively, and results are verified in the model.
REFERENCES


