Discrete Event Systems

Petri Nets

Lana Josipović
Digital Systems and Design Automation Group
dynamo.ethz.ch

ETH Zurich (D-ITET)
December 15, 2022

Most materials from Lothar Thiele and Romain Jacob
Last week in
Discrete Event Systems
Token Game of Petri Nets

A marking $M$ activates a transition $t \in T$ if each place $p$ connected through an edge $f$ towards $t$ contains at least one token.

If a transition $t$ is activated by $M$, a state transition to $M'$ fires (happens) eventually.

Only one transition is fired at any time.

When a transition fires
  - it consumes a token from each of its input places,
  - it adds a token to each of its output places.
Token Game of Petri Nets

A marking $M$ activates a transition $t \in T$ if each place $p$ connected through an edge $f$ towards $t$ contains at least one token.

If a transition $t$ is activated by $M$, a state transition to $M'$ fires (happens) eventually.

Only one transition is fired at any time.

When a transition fires
- it consumes a token from each of its input places,
- it adds a token to each of its output places.
Concurrent Activities

Finite Automata allow the representation of decisions, but no concurrency. Petri nets support concurrency with intuitive notations:

**Decision**

- decision / conflict

**Concurrency**

- fork
- join / synchronization
This week in
Discrete Event Systems
Discrete Event Models with Time

In many discrete event systems, time is an important factor.

- queuing systems
- computer systems
- digital circuits
- workflow management
- business processes
Discrete Event Models with Time

In many discrete event systems, time is an important factor.

Based on a timed discrete event model, we would like to determine properties:

- queuing systems
- computer systems
- digital circuits
- workflow management
- business processes
- delay
- throughput
- execution rate
- resource load
- buffer sizes
Discrete Event Models with Time

In many discrete event systems, time is an important factor.

There are many ways of adding the concept of time to Petri nets and finite automata.

In the following, we present one specific model.

Based on a timed discrete event model, we would like to determine properties:

- queuing systems
- computer systems
- digital circuits
- workflow management
- business processes
- delay
- throughput
- execution rate
- resource load
- buffer sizes

There are many ways of adding the concept of time to Petri nets and finite automata. In the following, we present one specific model.
Discrete Event Models with Time

What can you do with a timed model?

**Verify** timed properties

- How long does it take until a certain event happens?
- What is the minimum time between two events?
Discrete Event Models with Time

What can you do with a timed model?

**Verify** timed properties
- How long does it take until a certain event happens?
- What is the minimum time between two events?

**Simulate** the model
- Given a specific input, how does the system state evolve over time?
- Is the resulting trace of execution what we had in mind?
Time Petri Net

We define a delay function \( d: T \rightarrow R \) that determines the delay between the activation of a transition \( t \) and its firing.

- Repeated calls may lead to the same value or to different ones every time.
- The function is called for every new activation of transition \( t \) and determines the time until the transition fires.

\[ d(t) = 1 \text{ s} \]
Time Petri Net

We define a delay function \( d: T \rightarrow R \) that determines the delay between the activation of a transition \( t \) and its firing.

- Repeated calls may lead to the same value or to different ones every time.
- The function is called for every new activation of transition \( t \) and determines the time until the transition fires.

\[
d(t) = 1 \text{ s}
\]
Time Petri Net

We define a delay function $d: T \rightarrow R$ that determines the delay between the activation of a transition $t$ and its firing.

- Repeated calls may lead to the same value or to different ones every time.
- The function is called for every new activation of transition $t$ and determines the time until the transition fires.

\[
d(t) = 1 \text{ s}
\]
Time Petri Net

We define a delay function \( d: T \rightarrow R \) that determines the delay between the activation of a transition \( t \) and its firing.

- Repeated calls may lead to the same value or to different ones every time.
- The function is called for every new activation of transition \( t \) and determines the time until the transition fires.
- An activation is canceled whenever a token is removed from some input place of \( t \) (and a new activation can start immediately).

\[ d(t_1) = 1 \text{ s} \]
\[ d(t_2) = 2 \text{ s} \]

If the transition \( t \) loses its activation, then \( d(t) \) is called again at the next activation.
We define a delay function \( d: T \rightarrow R \) that determines the delay between the activation of a transition \( t \) and its firing.

- Repeated calls may lead to the same value or to different ones every time.
- The function is called for every new activation of transition \( t \) and determines the time until the transition fires.
- An activation is canceled whenever a token is removed from some input place of \( t \) (and a new activation can start immediately).

If the transition \( t \) loses its activation, then \( d(t) \) is called again at the next activation.

\[
\begin{align*}
d(t1) &= 1 \text{ s} \\
d(t2) &= 2 \text{ s}
\end{align*}
\]

\( t2 \) is reactivated: it will never fire! (same if 2 tokens in p1)
Time Petri Net

We define a delay function \( d: T \rightarrow R \) that determines the delay between the activation of a transition \( t \) and its firing.

- Repeated calls may lead to the same value or to different ones every time.
- The function is called for every new activation of transition \( t \) and determines the time until the transition fires.
- An activation is canceled whenever a token is removed from some input place of \( t \) (and a new activation can start immediately).
  - If the transition \( t \) loses its activation, then \( d(t) \) is called again at the next activation.
- Only one transition fires at a time (same as with regular Petri nets).
  - If two transitions have the same firing time, one of them is chosen non-deterministically to fire first.
Time Petri Net

An activation is canceled whenever a token is removed from some input place of $t$. 

$t=0$  
d(t1)  
d(t2)  
d(t3)

An activation is canceled whenever a token is removed from some input place of t.

firing time of t4; t1 and t2 get activated

t=0 d(t1) d(t3) d(t2) time

d(t1) 
d(t2) 
d(t3) 

Petri Net
An activation is canceled whenever a token is removed from some input place of t.
An activation is canceled whenever a token is removed from some input place of t.
Time Petri Net

firing times of t4

d(t1)

d(t2)

d(t3)

An activation is canceled whenever a token is removed from some input place of t.
An activation is canceled whenever a token is removed from some input place of $t$. 

Time Petri Net

firing times of $t_4$  firing time of $t_3$;

d($t_1$)  
d($t_2$)  
d($t_3$)
Time Petri Net

An activation is canceled whenever a token is removed from some input place of $t$.

firing times of $t_4$
firing time of $t_3$; $t_1$, $t_3$ lose activation; $t_1$, $t_3$ activated again
firing time of $t_2$;

$d(t_1)$
$d(t_2)$
$d(t_3)$
An activation is canceled whenever a token is removed from some input place of \( t \).
An activation is canceled whenever a token is removed from some input place of $t$. 

**Time Petri Net**

- **firing times of $t_4$**
- **firing time of $t_3$; $t_1, t_3$ lose activation; $t_1, t_3$ activated again**
- **firing time of $t_2$; $t_1, t_2$ lose activation; $t_1, t_2$ activated again**
- **firing time of $t_1$; $t_1, t_2, t_3$ lose activation; $t_3$ activated again**

- $d(t_1)$
- $d(t_2)$
- $d(t_3)$
The time when a transition \( t \) fires is called the **firing time**.

A time Petri net can be regarded as a generator for firing times of its transitions.

How do we get the firing times? By simulation!

- **firing time sequences for transitions** \( t_1, t_2 \) and \( t_3 \)

\[
\begin{align*}
\text{d}(t_1) &= 1 \\
\text{d}(t_2) &= 2 \\
\text{d}(t_3) &= 3 \\
\end{align*}
\]
The time when a transition \( t \) fires is called the **firing time**.

A time Petri net can be regarded as a generator for firing times of its transitions.

How do we get the firing times? By simulation!

- \( d(t_1) = 1 \)
- \( d(t_2) = 2 \)
- \( d(t_3) = 3 \)

**Event sequence (firing of transitions)**

- Event sequence: \( \{1, 6, 9, 12, ...\} \)
- Event sequence: \( \{5, 8, 11, 13, ...\} \)
- Event sequence: \( \{3, 6, 9, 12, ...\} \)

**firing time sequences for transitions**

- \( t_1 \)
- \( t_2 \)
- \( t_3 \)
Simulation Principle

The simulation is based on the following basic principles.

1. The simulator maintains a set $L$ of currently activated transitions and their firing times. We call $L$ the event list from now on.

2. A transition with the earliest firing time is selected and fired. The state of the Petri net as well as the current simulation time is updated accordingly.

3. All transitions that lost their activation during the state transition are removed from the event list $L$.

4. Afterwards, all transitions that are newly activated are added to the event list $L$ together with their firing times.

5. Then we continue with 2. unless the event list $L$ is empty.

This simulation principle holds in one form or another for any simulator of timed discrete event models.

Add tuple to $L$ when $t_i$ is activated:

$$L = \{(t_i, \tau_i)\}$$

$$\tau_i = \tau + d(t_i)$$

$\tau$: current simulation time (activation time of $t_i$)
Simulation Principle

- Event list L
- State $M$
- Simulation time $\tau$

Event list L → Current state → Initialization
Simulation Principle

- Event list $L$
- State $M$
- Simulation time $\tau$

Remove transition $t'$ with the earliest firing time $\tau'$

Event list $L$ → Current state → Fire $t'$
Simulation Principle

Initialization
- Event list L
- State $M$
- Simulation time $\tau$

Update
- state
- simulation time

$M := M + Au'$
$\tau := \tau'$

Event list L

Current state

remove transition $t'$ with the earliest firing time $\tau'$

Fire $t'$
Simulation Principle

Initialization
- Event list L
- State $M$
- Simulation time $\tau$

Update
- state
- simulation time

Event list L → Current state → Fire $t'$ → Remove transitions that lost their activation during the state transition

Remove transition $t'$ with the earliest firing time $\tau'$

$M := M + A \cdot u'$
$\tau := \tau'$
**Simulation Principle**

- **Initialization**
  - Event list $L$
  - State $M$
  - Simulation time $\tau$

- **Update**
  - state
  - simulation time

---

- **Event list $L$**
- **Current state**
  - $M := M + A u'$
  - $\tau := \tau'$
- **Fire $t'$**
- **Remove transitions that lost their activation during the state transition**
- **Add transitions that are newly activated after the state transition**
- remove transition $t'$ with the earliest firing time $\tau'$
Simulation Principle

- Event list $L$
- State $M$
- Simulation time $\tau$

Initialization

- Event list $L$
- State $M$
- Simulation time $\tau$

Update

- state
- simulation time

Iterate until $L$ is empty

Current state

- $M := M + A u'$
- $\tau := \tau'$

Remove transitions that lost their activation during the state transition

Add transitions that are newly activated after the state transition

Remove transition $t'$ with the earliest firing time $\tau'$
Simulation Example

\[ d(t1) = 1 \]
\[ d(t2) = 2 \]
\[ d(t3) = 3 \]

\[ L = \{ (t_i, \tau + d(t_i)) \} \]
Simulation Example

\[d(t_1) = 1\]
\[d(t_2) = 2\]
\[d(t_3) = 3\]

\[\tau = 0:\]
\[M = \begin{bmatrix} 2 & 0 & 1 & 0 \end{bmatrix}\]
\[L = \{(t_1, 1), (t_3, 3)\}\]
Simulation Example

\[ d(t_1) = 1 \]
\[ d(t_2) = 2 \]
\[ d(t_3) = 3 \]

\[ L = \{ t_1, t_3 \} \]
\[ \tau = 0: \]
\[ M = [2 \ 0 \ 1 \ 0] \quad L = \{ (t_1, 1), (t_3, 3) \} \]
\[ \tau = 1: \]
\[ M = [0 \ 1 \ 2 \ 0] \quad L = \{ (t_3, 3) \} \]
Simulation Example

\[ d(t_1) = 1 \]
\[ d(t_2) = 2 \]
\[ d(t_3) = 3 \]

\[ L = \{ t_1, 1 \}, \{ t_3, 3 \} \]

\[ \tau = 0: \]
\[ M = [2 0 1 0] \]
\[ L = \{ (t_1, 1), (t_3, 3) \} \]

\[ \tau = 1: \]
\[ M = [0 1 2 0] \]
\[ L = \{ (t_3, 3) \} \]

\[ \tau = 3: \]
\[ M = [1 1 1 2] \]
\[ L = \{ (t_3, 6), (t_2, 5) \} \]
Simulation Example

d(t1) = 1

d(t2) = 2

d(t3) = 3

\[ L = \{ \tau \} \]

\[ L = \{ (t_1, \tau + d(t_1)) \} \]

\[ \tau = 0: \]

\[ M = \begin{bmatrix} 2 & 0 & 1 & 0 \end{bmatrix} \quad L = \{ (t_1, 1), (t_3, 3) \} \]

\[ \tau = 1: \]

\[ M = \begin{bmatrix} 0 & 1 & 2 & 0 \end{bmatrix} \quad L = \{ (t_3, 3) \} \]

\[ \tau = 3: \]

\[ M = \begin{bmatrix} 1 & 1 & 1 & 2 \end{bmatrix} \quad L = \{ (t_3, 6), (t_2, 5) \} \]
Simulation Algorithm (1)

**Initialization:**
- Set the initial simulation time $\tau := 0$
- Set the current state to $M := M_0$
- For each activated transition $t$, add the event $(t, \tau + d(t))$ to the event list $L$

**Determine and remove current event:**
- Determine a firing event $(t', \tau')$ with the earliest firing time:
  \[ \forall 1 \leq i \leq N : \tau' \leq \tau_i \text{ where } L = \{(t_1, \tau_1), (t_2, \tau_2), \ldots, (t_N, \tau_N)\} \]
- Remove event $(t', \tau')$ from the event list $L$: $L := L \setminus \{(t', \tau')\}$

**Update current simulation time:** Set current simulation time $\tau := \tau'$

**Update token distribution $M$:**
- Suppose that the firing transition has index $j$, i.e. $t_j = t'$. Then, the firing vector is:
  \[ u' = [0 \cdots 0 1 0 \cdots 0]_j^t \]
- Update current state $M := M + A u'$
Simulation Algorithm (2)

Remove transitions from \( L \) that lost activation:

- Determine the set of places \( S' \) from which at least one token was removed during the state transition caused by \( t' \):

\[
S' = \{ p | (p, t') \in F \}
\]

- Remove from event list \( L \) all transitions in \( T' \) that lost their activation due to this token removal:

\[
T' = \{ t | (p, t) \in F \land p \in S' \}
\]

Add all transitions to event list \( L \) that are activated but not in \( L \) yet:

- If some transition \( t \) with \( M(p) \geq W(p, t) \) for all \( (p, t) \in F \) is not in \( L \), then add \( (t, \tau + d(t)) \) to the event list:

\[
L := L \cup \{(t, \tau + d(t))\}
\]
Petri Net Simulators

There are many simulators available

An overview

Examples

www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html
Discrete Event Models with Time

In many discrete event systems, time is an important factor.

There are many ways of adding the concept of time to Petri nets and finite automata. In the following, we present one specific model. What are the others?
There are mainly three ways to count time

**Delay** on the transition firing

**Duration** of the transition

**Age** of the tokens
There are mainly three ways to count time

**Delay** on the transition firing

**Duration** of the transition

**Age** of the tokens

Expressivity and analysis feasibility may vary between the models.

Time Petri nets
Covered here

Timed Petri nets
www.lsv.fr/~haddad/disc11-part1.pdf
Your turn to practice!

after the break

1. Model arithmetic operations with Petri nets
2. Use a simulator to explore the timed behavior of a simple Petri net model
3. Use a model-checker to adapt a system design
Quick recap
Discrete Event Systems (Part 3)

- How to efficiently explore the state space of DES models?
- How to formulate temporal properties of interest?
- How to formally verify such properties?
- How to efficiently model concurrency in DES?

Set of states & BDDs
CTL formulas
Reachability & model-checking
Petri nets w/ and w/o time
Thank you for following
Discrete Event Systems! 😊

Lana Josipović
Digital Systems and Design Automation Group
dynamo.ethz.ch

ETH Zurich (D-ITET)
December 15, 2022

Most materials from Lothar Thiele and Romain Jacob