Simulation of Discrete Event Systems

Unit 5
Petri Nets (II): Analysis of Net Models
Fall Winter 2015/2016

Univ.-Prof. Dr.-Ing. Dipl.-Wirt.-Ing. Christopher M. Schlick
Chair and Institute of Industrial Engineering and Ergonomics
RWTH Aachen University
Bergdriesch 27
52062 Aachen
Instructor:
Dipl.-Inform. Sinem Kuz
E-Mail: s.kuz@iaw.rwth-aachen.de
1. Problem classification in net analysis

2. Coverability tree
   - Definition
   - Construction algorithm
   - Application of tree models
   - Limited expressiveness

3. Linear-algebraic techniques
Focus of lecture and exercise

- **model**
  - static
  - dynamic
    - time-varying
    - time-invariant
      - linear
      - nonlinear
        - continuous states
        - discrete states

- Focus of lecture and exercise
  - deterministic
    - time-driven
    - event-driven
  - stochastic
    - discrete-time
    - continuous-time
1. Problem Classification in Net Analysis
Starting point: We want to model and simulate a modern discrete event system such as a computer-integrated manufacturing plant. This manufacturing plant contains multiple machine tools and shared resources, e.g. clamping devices, cutting tools or workpieces, and the access to these shared resources by concurrent production processes in plant subsystems has to be planned and coordinated.

There are six fundamental questions a system engineer needs to be answered with the help of discrete event system modeling and simulation:

1. Is the “consumption” of resources in the plant bounded or grows over given limits?
   → system bounded

2. Can the system with its queues and buffers be operated safely or do blocking states occur?
   → system safety and blocking

3. Is the functional design of plant behavior efficient or “over-engineered”?
   → state coverability

4. Can resources “dry out” or be generated unintentionally?
   → resource conservation

5. Is the system always reactive to changes in the demand?
   → system liveness

6. Can the system be operated persistently or do severe interrupts occur?
   → system persistence
1. **Def.:** A place $p_i \in P$ of a marked Petri net MPN with the initial state $x_0$ is $k$-bounded or $k$-safe, if the following condition holds:

- $x(p_i) \leq k$ for all $x \in R(MPN)$

The simplest case of a 1-bounded place is called a safe place. For larger values of $k$ one can also speak of a “limited place”.

If all places of a Petri net are $k$-bounded, the net is called a bounded Petri net. According to the fourth lecture we already know, that bounded Petri nets can be transformed into finite-state automata and then analyzed with the help of purely algebraic techniques.

Unbounded Petri nets represent a larger model class and can not be transformed into finite-state automata.
2. Blocking

Two types of blocking

**Deadlock:**
A deadlock in a Petri Net occurs, if - starting with the initial state $x_0$ - a net state is reached, which leaves all local state transition functions in the net undefined.

**Livelock:**
A livelock in a Petri net occurs, if tokens are generated and consumed only in a small subset of the set of places and no transition firing sequence exists to leave this partial coverage of the state space.

If a Petri net is bounded, it can be transformed into a finite-state automaton and we can use the transformed model to apply the blocking criteria that were introduced in the second lecture (see slide 2-13).

If the net is unbounded, the coverability tree is a useful analysis technique for investigating blocking states. This technique will be presented later in this lecture.
2. **Def.:** A state $y$ of a marked Petri net with the initial state $x_0$ is covered by state $x$, if the following condition holds:

- There exists an $x \in R(MPN)$, so that $x(p_i) \geq y(p_i)$ for all $i = 1,...,n$

Clearly, the notion coverability is a generalization of the already introduced notion of state reachability, because the coverability requires a threshold value for the number of tokens at a place. It is also related to the concept of eventually being able to fire a particular transition. In order to enable a transition, it is often required that a certain number of tokens is present in some places of special interest.

Consider some state $y = [y(p_1), \ldots, y(p_n)]$, which includes in each place the minimum number of tokens required to enable transition $t_j$. Suppose we are in the initial state $x_0$ and we would like to investigate if $t_j$ could be enabled. It is therefore essential to know whether we can reach state $x$ from $x_0$ such that $x(p_i) \geq y(p_i)$ for all $i = 1,\ldots,n$. If this is the case, we say that state $x$ covers state $y$. 

3. State coverability
### 4. Conservation

**3. Def.:** A marked Petri net with the initial state $x_0$ is considered as conservative with regard to the place-oriented weighting vector $c = [c_1, c_2, \ldots, c_n]$, if for all reachable states $x \in R(MPN)$ the following condition holds:

$$\sum_{i=1}^{n} c_i x(p_i) = \text{constant} \quad \forall i : c_i \in R_0$$

Obviously, this definition is the logical equivalent to a mass conservation law with regard to the tokens in the net places. For instance, if a Petri net model of a manufacturing system is conservative, we can be confident that no resources such as workpieces are generated or “dry out” unintendedly.

The weighting vector ensures a certain degree of flexibility in the “mass conservation law” when modeling and simulating Petri nets. For instance, if places for flow control are included in the net it is possible to exclude them from conservation considerations and set the corresponding coefficient in the weighting vector to zero.
5. Liveness

In Petri net analysis the notion “liveness” is associated with the transitions of the net. Therefore, liveness analysis is a local consideration between the two extremes of a deadlock and a livelock for every possible state reached from $x_0$.

We differentiate the following stages of liveness of transitions, starting with the initial net state $x_0$:

- **L0-live** or “dead”: The transition can never be enabled from the initial state

- **L1-live**: If there is some firing sequence from $x_0$ such that the transition can fire at least once

- **L2-live**: The transition can fire at least $k$ times for some given positive integer $k$

- **L3-live**: There exists some infinite firing sequence in which the transition appears infinitely often

- **L4-live**: The transition is L1-live for every possible state reached from $x_0$

The concept of coverability is closely related to that of L0 and L1 liveliness. If $y$ is the state that includes in each place the minimum number of tokens required to enable some transition $t_j$, then if $y$ is not coverable from the current state, transition $t_j$ is dead (L0-live). Thus, it is possible to identify dead transitions by checking for coverability.
Example of liveness

Transition 3 is L3-live, but not L4-live, because it is blocked, if transition 1 is activated.

Transition 2 is L0-live

Transition 1 is L1-live

$x_0 = [1, 0]$
6. Persistence

4. Def.: A marked Petri net with the initial state $x_0$ is said to be persistent, if for any pair of enabled transitions the enabling of one transition cannot disable the other.

Therefore, the previously introduced Petri net is not persistent, because the enabling of transition 1 has the effect that transition 3 cannot be enabled until a token is back in place 1, see:

![Diagram of Petri net with transitions and initial state](image)

However, the basic Petri net models of a queueing system which were introduced in lecture 4 (slides 4-17 & 4-18) are persistent.

The persistence criterion is equivalent to a property referred to as non-interuptedness. To better understand this term think of each enabled transition as having to go through some physical process, before it can fire. If the firing of a transition disables another enabled transition, it effectively interrupts this process.
Definition of a coverability tree

5. Def.: A coverability tree \( CT \) of a marked Petri net represents the states of the net that can be covered with valid transition firing sequences. A coverability tree is a hierarchical graph with directed arcs. The nodes of the tree represent (generalized) net states and the arcs, the transitions between states. The root node encodes the initial net state \( x_0 \). The child nodes express dominance relationships with respect to their parents. Duplicate nodes in the tree reflect recurrent net states. Terminal nodes represent the end of valid paths through the tree and can represent states, in which blocking occurs.

Example of a finite coverability tree:

\( CT(MPN1) \)

\[
[x(p_1), x(p_2), x(p_3)]
\]

- \([1, 1, 0]\) Root node
- \([0, 0, 1]\) Leaf node
- \([1, 1, 0]\) Duplicate node
Example of an infinite coverability tree

\[ CT(MPN2) \]

\[
\begin{bmatrix}
1, 0, 0, 0 \\
0, 1, 1, 0 \\
0, 0, 1, 1 \\
\end{bmatrix}
\]

Node dominates

Terminal node

Terminal node
1. Root node: Represents initial state $x_0$ and is always on top of the node hierarchy.

2. Terminal nodes: All nodes, which do not have leaf nodes.
   (if all vector components are also non-negative integers, all transitions are left $L_0$-live)

3. Duplicate nodes: All nodes that are identical with other nodes in the hierarchy.

4. Dominance: $x = \{x(p_1), ..., x(p_n)\}$ and $y = \{y(p_1), ..., y(p_n)\}$ are two tree nodes;
   node $x$ dominates node $y$ (notation: $x >_d y$), if
   - $x(p_i) \geq y(p_i)$ for all $i = 1, ..., n$ (simple dominance condition) and
   - $x(p_i) > y(p_i)$ for at least a $i = 1, ..., n$ (strict dominance condition)

5. $\omega$-symbol: This important tree symbol represents a possibly infinite number of tokens in a place;
   the $\omega$-symbol is used, if there is a node dominance in the tree, which can be
   generalized to an infinitely long sequence of the firing of transitions.
Deriving a coverability tree from a Petri net: Algorithm

1. **Step:** Initialize the root node of the tree with \( x = x_0 \).

2. **Step:** For each new node \( x \), evaluate the transition function \( f(x, t_j) \) for all \( t_j \in T \) as follows:
   
   2.1. **Step:** If \( f(x, t_j) \) is undefined for all \( t_j \in T \), then mark node \( x \) as a terminal node.

   2.2. **Step:** If \( f(x, t_j) \) is defined for at least one \( t_j \in T \), then create a new node \( x' = f(x, t_j) \).

      If necessary, adjust the marking of node \( x' \) as follows:

      2.2.1. **Step:** If \( f(x, t_j) = \omega \) for a \( p_i \), define \( x'(p_i) = \omega \).

      2.2.2. **Step:** If a node \( y \) exists in the path from the root node \( x_0 \) to \( x \) such that \( x' \succ_d y \), set \( x'(p_i) = \omega \) for all \( p_i \) such that \( x'(p_i) > y(p_i) \).

      2.2.3. **Step:** Otherwise, set \( x'(p_i) = f(x, t_j) \).

      If node \( x' \) is identical to a node in the path from \( x_0 \) to \( x \), then mark \( x' \) as a duplicate node.

3. **Step:** If all new nodes are either terminal nodes or duplicate nodes, stop tree construction.
Example of a stepwise constructed finite coverability tree

CT(MPN1)

1. Step $[1, 0, 0, 0]$

2.2. Step $[0, 1, 1, 0]$

2.2.3. Step $[0, 0, 1, 1]$

Terminal node

2.2. Step $[1, 0, \omega, 0]$

2.2.3. Step $[0, 0, 1, 1]$

Duplicate node, Terminal node

2.2. Step $[1, 0, \omega, 0]$

2.2.1. Step $[0, 0, \omega, 1]$

2.2.3 Step

Terminal node

2.2. Step $[1, 0, \omega, 0]$

2.2.1. Step $[0, 0, \omega, 1]$

2.2.3 Step
1. **Boundedness:**

Necessary and sufficient condition for the boundness of a Petri net is that the $\omega$-symbol does not occur in its coverability tree. In other words, the $\omega$-symbol represents a possible number of infinite tokens in a place and therefore the net cannot be bounded.

Furthermore, if the $\omega$-symbol does not occur in the coverability tree, the associated state space of the modeled discrete-event system is finite and the system behavior can be simulated with the help of finite state automata (lecture 2). In this case the coverability tree is also said to be a reachability tree, because it contains only reachable states.

2. **Safety and Blocking**

A Petri net is blocked, if there are terminal nodes in the coverability tree, which do not contain the $\omega$-symbol and are not duplicate nodes.

Conversely, a Petri net has a livelock, if the $\omega$-symbol is included in a terminal node. If a terminal node contains the $\omega$-symbol and is also a duplicate node, then the involved places and transitions in the livelock can be identified easily by “tracing” back the directed arcs to the corresponding duplicate node.
3. State Coverability:

The coverability tree directly encodes, which net state $y$ can be covered when starting in a certain initial state: If there is a node $x$ in the coverability tree, which dominates $y$ ($x(p_i) \geq y(p_i)$ for all $i$) then $y$ can be covered.

If the net state $x$ contains the $\omega$-symbol at least once, there is a loop in between $y$ and $x$. Depending on the particular values of $y(p_i)$, $i = 1 \ldots n$, that need to be covered we can determine the number of loops involved in this path until $y$ is covered.
4. Conservation

We consider a certain weighting vector \( c = [c_1, c_2, \ldots, c_n] \) with non-negative components. We need to observe that if \( x(p_i) = \omega \) for some \( p_i \), then we must have \( c_i = 0 \), if the corresponding Petri net is to be conservative. This is because \( \omega \) does not represent a fixed integer value.

Conversely, we can pose the following problem: Given a Petri net, is there any weighting vector such that the conservation condition is satisfied. To solve this problem, we set \( c_i = 0 \) for all unbounded places \( p_i \). Next, suppose there are \( b \leq n \) bounded places and let the coverability tree consist of \( r \) nodes and let the value of the constant sum in the conservation condition be \( C \). We then have a set of \( r \) equations of the form

\[
\sum_{i=1}^{b} c_i x(p_i) = C \quad \text{for each state } x \text{ in the coverability tree}
\]

with \((b+1)\) unknowns, namely, the \( b \) positive weights for the bounded places and the constant \( C \). Thus, the conservation problem reduces to a standard algebraic problem for solving this set of linear equations. If one or more solutions exist, it is possible to determine them with well-known techniques.
1. Proposition: A marked Petri net MPN cannot be reconstructed unambiguously from a coverability tree, if the $\omega$-symbol occurs in the tree and therefore accumulates model behavior.

Example:
3. Linear-Algebraic Techniques

3. Linear-Algebraic Techniques
In addition to the algorithmic Petri net analysis based on the coverability tree, the state equation can be analyzed by linear-algebraic techniques. The state equation was defined in lecture 4 on slide 13. Thus, conclusions about both, reachability and conservation can be drawn.

1. Reachability:

**Proposition:** Let MPN be a marked Petri net with incidence matrix $A$ and initial state $x_0$.

If the state equation

$$x = x_0 + vA$$

cannot be solved concerning a net state $x$ of interest under the constraint of a “firing count vector” $v$ with non-negative integer components, then the net state $x$ is not reachable (sufficient condition).

Necessary, but not sufficient condition for the reachability of a net state $x$ is, that there exists a solution of the state equation and all components of $v$ are non-negative integers.
Example of analysis of the state equation

Is state $x = [0 \ 1 \ 0 \ 0]$ reachable?

$$vA = x - x_0 \iff \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} A = \begin{bmatrix} -1 & 1 & 0 & 0 \end{bmatrix}$$ cannot be solved!

$\Rightarrow \ x \text{ is not reachable}$

Is state $x = [0 \ 0 \ 0 \ 1]$ reachable?

$$vA = x - x_0 \iff \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} A = \begin{bmatrix} -1 & 0 & 0 & 1 \end{bmatrix} \text{ can be solved!}$$

$$v = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}$$

Vector components are non-negative, but $t_1$ and $t_2$ cannot fire in the initial state!!!
Analysis of the state equation (II): Conservation

2. Conservation property:

**Proposition:** Let $A$ be the incidence matrix of a Petri net. If there exists a weighting vector $c = [c_1, c_2, \ldots, c_n]$, whose entries satisfy $c_i \geq 0$ for all $i$, and such that

$$Ac^T = 0, \quad x = x_0 + vA \quad \text{with} \quad xc^T = x_0c^T + vAc^T \quad \text{then we can write}$$

$$xc^T = x_0c^T.$$

Since the last equality holds for all states $x$ that are reachable from $x_0$ this means that the Petri net is conservative with respect to this vector $c$ for any choice of the initial state $x_0$. 


Open Questions?