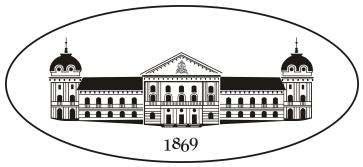

Bulgarian Academy Of Sciences
Institute Of Biophysics And Biomedical Engineering

Software Product for the Implementation of Generalized Net Models and Its Applications

ABSTRACT OF A DISSERTATION

for the award of educational and scientific degree "Doctor"
in the professional field:
4.6 "Informatics and Computer Science"
doctoral programme "Informatics"

Angel Ivanov Dimitriev

Scientific supervisors:

Acad. DSc. Krassimir Atanassov
Assoc. Prof. Nora Angelova

Sofia, 2026

Abstract of a dissertation thesis

Author: Angel Ivanov Dimitriev

Title: Software Product for the Implementation of Generalized Net Models and Its Applications

The dissertation thesis was discussed and admitted to defense at a meeting of the Scientific Unit of IBPhBME – BAS on _____.

The public defense will be held on _____ at _____ in _____.

The dissertation thesis contains:
182 pages, 85 figures, 25 code fragments, and 19 formulas.

The bibliography includes: 104 references.

The structure of the dissertation includes: Introduction, 4 chapters, Conclusion, Author's statement of contributions, List of publications related to the dissertation topic, and Bibliography.

- The **Introduction** substantiates the relevance of the topic and defines the object and the subject of the research.
- **Chapter 1** presents the theoretical foundations of Generalized Nets (GNs), analyzes the existing software for simulation, and formulates the aim and objectives of the dissertation thesis.
- **Chapter 2** is devoted to newly developed algorithms for automatic drawing of GNs and transformation of graphical images into *TeX* format.
- **Chapter 3** describes the architecture, design, and implementation of the developed software simulator *OnlineGN*.
- **Chapter 4** examines the creation of OM models for processes to which OM have not been applied so far or for which other approaches have been used, as well as the applications of the simulator, with each of the specified models being simulated and verified through it.
- **The Conclusion** summarizes the achieved results and outlines directions for future work.

Contents

Introduction

The Theory of Generalized Nets (GNs) provides a powerful mathematical apparatus for modeling and simulating complex processes characterized by parallelism and logical interdependence. Since its inception, the GN apparatus has established itself as a flexible tool in areas such as industrial production, computer systems, and artificial intelligence. However, with the growing complexity of models, the need for accessible and highly efficient software solutions for their description and study has become increasingly evident.

Despite the existing developments, there is still a lack of tools that enable work in an online environment, automatic generation of graphical representations, and straightforward integration into scientific publications. The main goal of the present work is the creation of the software system ***OnlineGN*** – a web-based simulator that combines the theoretical foundations of GNs with modern web technologies. Within the dissertation, new algorithms for automatic visualization and export of models are proposed, turning ***OnlineGN*** into a comprehensive modeling environment. Through a series of simulations of classical problems and real manufacturing scenarios, the applicability and effectiveness of the developed tool are demonstrated.

Chapter 1. Generalized Nets: Theoretical Foundations and Simulation Software

1.1. Notes on the Theory of Generalized Nets

1.1.1. On the Concept of a Generalized Net

Generalized Nets (GNs) were first presented in 1982 in a report at an international conference, with the text published in 1984 [1]. They emerged as a natural continuation of efforts to model parallel processes, which began with the research of *C. A. Petri* in 1962 and led to the formulation of Petri nets (PNs) [2].

In their classical form, PNs are bipartite directed graphs with two types of vertices: *transitions* (discrete events) and *places* (conditions), between which *tokens* move. The orientation toward parallelism led to numerous extensions of PNs (more than twenty by 1982), which add: (i) temporal aspects; (ii) coloring of tokens for distinguishability; (iii) guard conditions (predicates) over groups of tokens for synchronized firings; and others.

Against this background, GNs provide a richer framework for the specification of concurrency, synchronization, and conditional dependencies, while preserving the intuitive graph semantics of PNs and extending their expressive power for modeling complex parallel processes. Detailed studies of GNs are given in [7, 8, 9, 10], while a small part of their applications is considered in [11, 12, 13, 14].

1.1.2. Informal Description of the Notation in Generalized Nets

In the dissertation, the main notation and rules in GNs are summarized. Places and transitions are connected by *arcs* (arrows), which determine the possible directions of movement. Some places contain *tokens* (dots), analogously to classical Petri nets [17]. An entering token carries *initial characteristics*, and when passing through the net it may acquire additional characteristics. Since the initial characteristics may remain valid, a *history* accumulates for the token. From the point of view of implementation, this set of characteristics can be regarded as an associative list "name $\rightarrow$ value". A place may optionally have a *characteristic function*, which assigns new values to the characteristics of the tokens entering it.

The connectivity in GNs is simplified with respect to degrees: each place is connected to at least one arc and may have at most one incoming and one outgoing arc. Without an incoming arc, a place is *input*, and without an outgoing arc – *output*. Each transition has at least one input and one output place ($m, n \geq 1$); it is possible that, for a given transition, the same place plays both roles [8]. When a token reaches an input place of a transition, it becomes *potentially active*; if the conditions for transfer to the output places are satisfied, the transition is *active*.

The conditions associated with transitions are not described by a single predicate, but by an *indexed matrix* (IM) – a matrix of predicates that simultaneously specifies when movement along the different directions is possible and determines the capacities of the corresponding arcs. Thus, the IM provides finer control in comparison with standard Petri nets [8].

Time in the definition of GNs is *discrete*: successive steps are considered, numbered by an integer variable that increases by one. It is permissible to fix in parallel an *absolute time scale* with an initial moment and a horizon of functioning [8]. For multiple connected processes, separate scales may be used for each of them, or a common one for synchronization.

GNs are sufficiently expressive to describe both the extensions of Petri nets and models with the computational power of a Turing machine class (in particular, finite automata) [8]. The GN extensions known so far are *conservative*: for each of them there exists a standard GN that equivalently describes its functioning and result [8, 9].

1.1.3. Formal Definition of Generalized Nets

In this section, the dissertation considers a strictly mathematical definition of the GN components. The formal apparatus is based on set theory and the apparatus of IMs, defining the structure, temporal parameters, and token dynamics. Formally, each transition is described by the ordered 7-tuple

$$Z = \langle L', L'', t_1, t_2, r, M, \Box \rangle,$$

where r is the IM that specifies the transition conditions, and M is the IM that specifies the capacities of the arcs of the transition; the transition is activated at moment t_1, provided that the truth value of the Boolean expression is "true". The generalized net is given by the ordered 4-tuple

$$E = \langle \langle A, \pi_A, \pi_L, c, f, \theta_1, \theta_2 \rangle, \langle K, \pi_K, \theta_K \rangle, \langle T, t^0, t^* \rangle, \langle X, \Phi, b \rangle \rangle,$$

where the transitions and places, priorities, capacities, time and memory functions, and characteristic functions are defined. The GN components are classified according to the static structure, dynamic nature, temporal essence, and memory of the GN, and the notion of an *empty GN* is introduced.

1.1.4. Algorithms for Token Movement in a Transition and in a Generalized Net

In this section, the dissertation presents an extended version of the most general algorithm known so far for the functioning of a transition in a GN [8, 7, 9, 10].

1.2. Current Software for Simulation of Generalized Nets

1.2.1. Main Shortcomings of Current Simulators

This section of the dissertation considers the main shortcomings of current simulators, which simultaneously affect architecture, user experience, and maintenance. These include **fragmentation and incomplete integration**, **portability and dependency problems**, as well as **language layer and compatibility**. These and other reasons motivate the development of *OnlineGN* as a web-based, integrated, and platform-independent environment.

1.2.2. Analysis of Existing Software Solutions

Existing software solutions for modeling and simulation of GNs are considered, describing the architecture, components, and interaction modes in ***GN Lite*** (incl. *GNTicker*, *TickerServer/GNTP*, *GN IDE*, *GNVis*, *XGN2SVG*, *GNProfy*, *GNDraw*), as well as the approach of ***GoGN*** for developing models by means of a *C#* library. The emphasis is on the technologies used, the degree of integration between the tools, and the practical aspects of working with the *XGN* format.

1.2.3. Main Shortcomings of Current Simulators

The main shortcomings of current simulators are summarized below, as they affect architecture, user experience, and maintenance simultaneously.

- **Fragmentation and incomplete integration.**
  Early versions of *GNTicker* are console-based, while interactivity is compensated through *TickerServer*/*GNTP* and separate auxiliary tools outside the main modeling–simulation cycle.
- **Portability and dependency problems.**
  Some of the tools are aimed at *Windows/.NET*, and under *Unix*-like operating systems they often rely on *Mono*, which increases installation and maintenance effort.
- **Language layer and compatibility.**
  GNTCFL increases the learning curve; *GNJS* aims to reduce it, but the prototypes do not provide full compatibility with *GNTP*, which limits integration.
- **Scalability and performance.**
  The algorithms are correct, but not always optimal for very large or highly connected nets, which leads to longer execution times and higher resource consumption.
- **Limitations of the tooling and ecosystem.**
  XGN2SVG produces static output; *GNVis* is a separate *.NET* module; *GNProfy* is focused on sequence diagrams and older *XMI* versions; *GoGN* requires programming and compilation instead of declarative modeling.

1.3. Aim and Tasks of the Dissertation

1.3.1. Formulation of the Aim

The aim of the present dissertation is to develop a web-based simulator for GNs – *OnlineGN*, which provides an interactive environment for creating, visualizing, editing, sharing, and simulating models, and to validate its applicability by developing, simulating, and verifying real GN models.

1.3.2. Tasks for Achieving the Aim

To achieve the stated aim, the following tasks have been formulated:

Task 1. Development and implementation of an algorithm for automatic drawing of GNs from their description in a structured textual format.

Task 2. Development and implementation of an algorithm for converting an *SVG* image of a GN into *TeX* format, for easy inclusion of visualizations in scientific publications.

Task 3. Formulation of requirements for a web-based solution.

Task 4. Design of the architecture of the *OnlineGN* system and definition of the main modules (net storage, visualization, editing, simulation, import/export, *API*).

Task 5. Implementation of a web-based graphical interface for visualization, editing, and personalization of the generated image (places, transitions, connections, and visual parameters).

Task 6. Implementation of a convenient interface for editing predicates in the indexed matrix and the characteristic functions of the places.

Task 7. Implementation of a mechanism for online storage and sharing of nets via a unique hyperlink (link), based on centralized database storage.

Task 8. Creation and formalization of new GN models of real processes that have not so far been described by means of GNs.

Task 9. Simulation and verification of the developed models in *OnlineGN* through scenarios and experiments.

The realization of the stated tasks will lead to the creation of a functional web-based simulator, accompanied by new algorithms for automatic drawing and conversion to *TeX*. The practical applicability of the development will be confirmed by validation examples and by the formalization of new GN models of real processes.

In summary, the dissertation addresses the need for automation and effective collaboration when working with the GN apparatus. The *OnlineGN* project offers a complete, flexible, and easy-to-use solution, aligned with contemporary software standards and the needs of the research community.

Chapter 2. Newly Introduced Algorithms for Generalized Nets

2.1. Algorithm for Automatic Drawing of Generalized Nets

The chapter presents the solution of **Task 1** through the development of an algorithm for automatic drawing of GNs, described in detail in [39].

2.1.1. Introduction

When working with GNs, a significant difficulty is the need for substantial manual effort in drawing them. Traditionally, the coordinates of the elements are specified precisely, which is time-consuming and carries a risk of human error. The same problem is also observed in GN simulators, where manual input of coordinates slows down modeling and limits rapid experimentation.

In response, an algorithm for automated drawing is proposed, which generates clear visual representations of GNs solely from their abstract description and eliminates the need for manual specification of coordinates. In this way, the creation and manipulation of GNs are facilitated, and attention is directed toward analysis rather than toward the technical details of rendering.

2.1.2. Current Practices in the Visualization of Generalized Nets

The visualization of GNs requires precise positioning of places, transitions, and arcs for the sake of clarity and structural correctness. Traditionally, this is done interactively through manual specification of coordinates, following established conventions for arrangement and readability.

Directionality of arrows.
Arrows are usually oriented from left to right, with the outputs of the transitions drawn on the right. When feedback connections are needed, they are added in a way that does not disrupt the overall clarity.

Positioning of output places.
Output places are positioned immediately to the right of the corresponding transitions, which facilitates tracing the flow and the logic of the net.

Places with a dual role (input and output).
When a place is both an input and an output, it is positioned in the lower part of the transition, with the aim of producing a more orderly diagram and minimizing crossings.

Additional practices.
For better readability, uniform spacing, clear labels, color differentiation when needed, minimization of crossings, and consistent orientation of the net are applied.

Although these practices support clear visualization, manual specification of coordinates remains labor-intensive and error-prone, which motivates automated solutions. In the following parts, an algorithm for automatic drawing of GNs is presented, consistent with the established conventions and reducing both manual effort and the risk of errors.

2.1.3. Drawing Graphs

The visualization of graphs aims at a clear presentation of structure through suitable placement of vertices and edges with minimal crossings [31]. For this purpose, *Graphviz* is often used – a tool for automatic generation of graph images from a *DOT* description, with various layout algorithms (e.g. *dot*, *neato*) [32].

2.1.4. Modeling a Generalized Net as a Directed Graph

For the purposes of visualization and structural analysis, a GN can be represented as a directed graph in which **places** and **transitions** are regarded as **vertices**, while the **arcs** describe directed connections between them. An arc $l \to Z$ is drawn when place l is an input to transition Z, and an arc $Z \to l$ when l is an output of Z. In this way, the direction of the arcs reflects the flow of tokens in the net.

Example.
A GN is considered with places l_1, l_2, l_3, l_4, l_5 and transitions Z_1, Z_2, Z_3, where the connections are: $l_1 \to Z_1$, $Z_1 \to l_2, l_3$, $l_2 \to Z_2$, $Z_2 \to l_4$, $l_3, l_4 \to Z_3$, $Z_3 \to l_5$. The GN is shown in **Figure 5**, and the corresponding directed graph – in **Figure 6**.

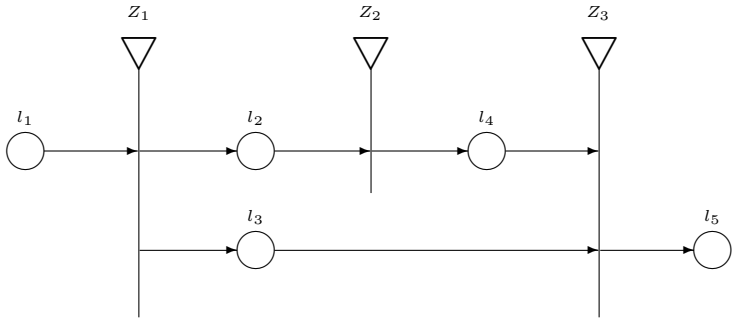

Figure 5 *– A generalized net with places l_1, l_2, l_3, l_4, l_5 and transitions Z_1, Z_2, Z_3.*

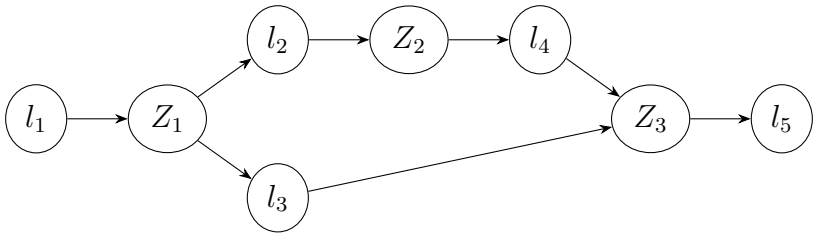

Figure 6 *– Directed graph corresponding to the GN. The vertices represent places and transitions, and the directed edges represent the flow between them*

2.1.5. Visualization of Generalized Nets with *Graphviz*

For correct and readable visualization of GNs with *Graphviz*, parameters and constraints are specified that preserve the standard arrangement of the elements.

Setting global parameters.
rankdir=LR is used for left-to-right orientation and *splines=ortho* for orthogonal arcs, which preserves the grid-like appearance of GNs and facilitates tracing the flow.

Defining the vertices for transitions.
Transitions are modeled as rectangular vertices (*shape=rect*) with a small fixed width, while the height is set according to the number of output places in order to reflect their connection with multiple outputs.

Grouping the output places with *subgraph*.
The transition and its output places are grouped in a *subgraph* (e.g. *cluster*), using *rank=same* to align the outputs and preserve their proximity to the transition during automatic arrangement.

Managing the input arcs through "invisible" vertices.
In order to obtain a single input point and for the inputs to "enter" from the left, auxiliary "invisible" vertices are added in front of the transition, to which the input places are connected, and then these vertices are connected to the transition itself, while their relative positioning is fixed through grouping.

2.1.6. Algorithm for Generating *Graphviz* Strings from Generalized Nets

This section describes the algorithmic logic for generating a *Graphviz* string from a GN (*GN*). The algorithm has been created for automating GN visualization and guarantees that the resulting graph follows predefined layout parameters. The main function, *generateGraphvizString*, uses several helper functions, which are responsible for mapping transitions and places, generating invisible vertices, and creating the necessary arcs between the elements of the GN.

The structure of the pseudocode, summarizing the main steps and the logic of the algorithm, is defined as follows:

Generating invisible vertices.
The function *genInvisNodes* generates invisible vertices used for alignment of the transitions in the graphical layout of the graph.

Algorithm 1 genInvisNodes(result, genNet, transition)

```
for each inputPlace in transition do
   result ← result + "invis_node_" + transition.name + "_" +
indexOf(inputPlace) + "[shape = point, width = 0.01, height = 0.01]" +
newLine
end for
```

Connecting the invisible vertices to the transitions.
The function *genEdgeBetweenInvisNodesAndTransition* generates arcs from the invisible vertices to the transitions, providing correct alignment and connectivity.

Algorithm 2 genEdgeBetweenInvisNodesAndTransition(result, genNet, transition)

```
for each inputPlace in transition do
   result ← result + "invis_node_" + transition.name + "_" +
indexOf(inputPlace) + " -> " + transition.name + ":w" + newLine
end for
```

Generating transitions.
The function *genTransitionsString* generates the necessary configuration for each transition in the GN, including the invisible vertices and the connections among them.

Algorithm 3 genTransitionsString(genNet, result)

```
for each transition in genNet.transitions do
   result ← result + "subgraph cluster_" + transition.name + "{"
   result ← result + "style=invis"
   result ← result + "subgraph cluster_" + transition.name + "_0" +
"{"
   result ← result + transition.name + "[shape=rect,
height=max(count(outputPlaces), count(inputPlaces)), width = 0.005]"
   call genInvisNodes(result, genNet, transition)
   call genEdgeBetweenInvisNodesAndTransition(result, genNet,
transition)
   call genOutgoingPlacesFromTransition(result, genNet, transition)
   call genOutgoingEdgesFromTransition(result, genNet, transition)
    result ← result + "}"
end for
```

Generating output places and arcs.
Two separate functions are responsible for generating the output places and the corresponding arcs. In this way, it is guaranteed that the arcs between transitions and places are created correctly.

Algorithm 4 genOutgoingPlacesFromTransition(genNet, result, transition)

```
result ← result + "{rank = same;"
for each outputPlace in transition do
    result ← result + outputPlace.name + ";"
end for
result ← result + "}"
```

Algorithm 5 genOutgoingEdgesFromTransition(genNet, result, transition)

```
for each outputPlace in transition do
    result ← result + transition.name " -> " outputPlace.name
end for
```

Generating outgoing arcs from the places.
Finally, the function *genOutputEdgesFromPlaces* generates arcs connecting the places to the invisible vertices, thus ensuring the correct alignment of the transitions.

Algorithm 6 genOutputEdgesFromPlaces(genNet, result)

```
for each place in genNet do
    invisNodeIndex ← first invisible node index for the transition
    result ← result + place.name + " -> " + "invis_node_" +
"transitionName" + "_" + invisNodeIndex + "[arrowhead=none]"
end for
```

Main function for generating a *Graphviz* string.
The main function *genGraphvizString* combines all previous functions, generating the full *Graphviz* string representing the GN.

Algorithm 7 genGraphvizString(genNet)

```
result ← " digraph G "
result ← result + "rankdir=LR;"
result ← result + "splines=ortho;"
call genTransitionsString(result, genNet)
call genOutputEdgesFromPlaces(result, genNet)
result ← result + "}"
```

The result of applying the described algorithm for generating a *Graphviz* string is shown in **Figure 8**. The visualization reflects the automatically generated GN structure before subsequent graphical processing.

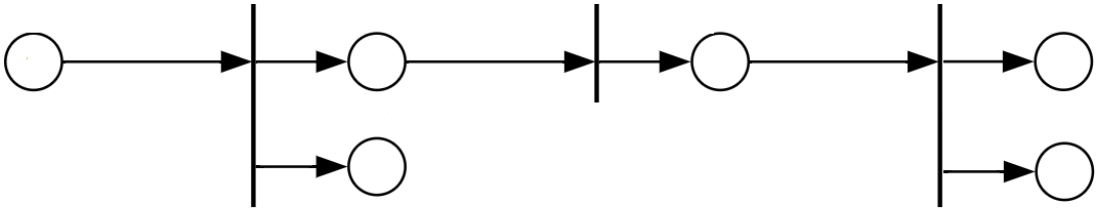

Figure 8 *– Example of a visualized graph (generalized net) using Graphviz.*

Subsequent processing of the generated image.
This paragraph describes the techniques applied to the *SVG* (*Scalable Vector Graphics*) file generated by the algorithm. Initially, a graph is created in which transitions are represented as very thin rectangular vertices and places – as circular vertices. In order to adapt this visual representation to the GN form, a series of modifications is performed during subsequent processing.

The goal is for the resulting image to correspond more closely to the GN form. The specific changes are described below:

- **Transitions:** Each transition, initially defined as a narrow rectangle, is subjected to additional processing – a black triangle is added in its upper part, which serves as a graphical identifier and improves its visibility in the overall structure. Above the triangle, a label with the name of the transition is visualized, guaranteeing its unambiguous identification in the model.

- **Places:** For the places, represented by circles, their names are added above each circle. This is necessary because the tool for generating the initial *SVG* (*Graphviz*) does not support this way of labeling by default. Therefore, the required labels are inserted manually in order to ensure clarity in the final visual representation.

- **Visualization of the generalized net:** After the application of the above changes, the resulting *SVG* closely resembles a GN. The transitions are clearly marked and labeled, while the places have their names legibly placed above their circles. This approach leads to a more intuitive and accurate graphical representation of the system being modeled, facilitating its interpretation and analysis.

  The final result after applying the described steps for subsequent *SVG* processing is shown in **Figure 9**. The resulting image follows the visual conventions of GNs and allows intuitive perception of structure and flow.

- **Example of the resulting image:**

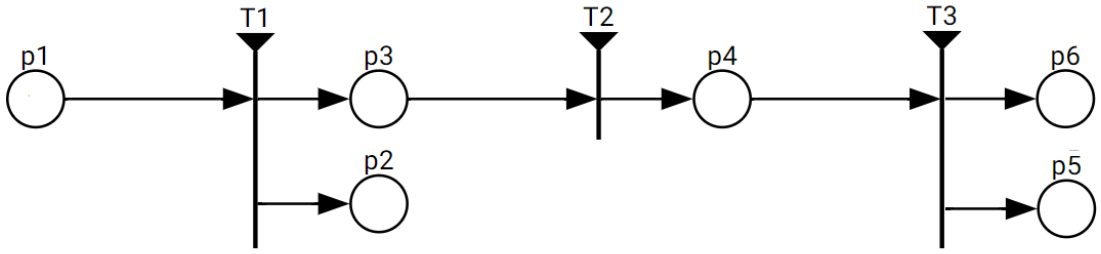

Figure 9 *– Example of a generalized net after visualization processing.*

- **Pseudocode of the processing step:**

Algorithm 8 postprocessSVG(graph)

```
svg ← loadSVG(graph)
for each transition in svg do
    points ← getTransitionPoints(transition)
    topPoint ← findTopMostPoint(points)
    addBlackTriangle(topPoint)
    label ← getTransitionLabel(transition)
    placeLabelAboveTriangle(topPoint, label)
end for
for each position in svg do
    circle ← findCircle(position)
    centerPoint ← getCenterPoint(circle)
    label ← getPositionLabel(position)
    placeLabelAboveCircle(centerPoint, label)
end for
outputSVG ← saveUpdatedSVG(svg)
return outputSVG
```

- **Pseudocode of the main drawing function:**

Algorithm 9 drawGenNet(genNet)

```
graphvizStr ← genGraphvizString(genNet)
svg ← Graphviz.render(graphvizStr)
postprocessSVG(svg)
```

2.2. Algorithm for Converting an *SVG* Image of a Generalized Net into *TeX* Format

This section presents an algorithm for automatic conversion of an *SVG* image of a GN into *TeX* code, thereby solving **Task 2**.

2.2.1. Introduction and Aim of the Algorithm

SVG is a convenient vector format for generation and editing, but in scientific texts *TeX* is most commonly used, where figures must fit cleanly and consistently with the style of the document. Therefore, a method is introduced for transferring an *SVG* image of a GN into *TeX* without manual redrawing.

The algorithm extracts places, transitions, and connections from the input *SVG* and generates an equivalent diagram through `\put`, `\circle`, `\line`, and `\vector`. The resulting code is inserted directly into a document and minimizes the need for manual corrections.

2.2.2. Description of the Format of the Input *SVG* File

The algorithm works with an *SVG* having a predefined structure. Each element has a `class` attribute indicating whether it is a place, a transition, or an arc. The GN components are represented as `<g>` groups:

- ***Places***: `class` following the pattern `"node place {id}"`. Inside there are `<ellipse>` (visualization) and `<title>`, `<text>` (identifier).
- ***Transitions***: `class` following the pattern `"node transition {id}"`. The group contains two `<polygon>` elements (vertical line and triangle), marked with `class "triangle"`, as well as `<title>` and `<text>` for the identifier.
- ***Edges***: `class` following the pattern `"edge {originId}__{destinationId}"`. The group contains a `<path>` (geometry of the arc) and a `<polygon>` (arrowhead).

2.2.3. Extracting the Coordinates from the Input *SVG* File

Only the coordinates required for construction in *TeX* are parsed:

- ***Places***: `cx, cy` from `<ellipse>` and `x, y` of the label from `<text>`.
- ***Transitions***: coordinates of the **two** `<polygon>` elements and `x, y` of the label from `<text>`.
- ***Edges***: from `<path>`, the `d` attribute is taken and parsed into a sequence of points `(x_k,y_k)` through the commands `M` and `L`. The arrowhead is parsed from `<polygon points="...">` as points `(x_j,y_j)`.

Finally, the **global transformation** (e.g. `translate` and `scale`) is also extracted, and it is applied to all coordinates.

2.2.4. Global Transformations and Normalization of Coordinates

The coordinates in *SVG* are often subjected to a common transformation, which must be applied in order to obtain the correct scale and arrangement in *TeX*. The typical format is:

`transform="translate(tx, ty) scale(s)"`

The following are extracted:

$$\textit{globalTranslate}_x = tx, \quad \textit{globalTranslate}_y = ty, \quad \textit{globalScale} = s$$

For each point (x, y), the following is applied:

$$\widehat{X} = x \cdot s + tx, \quad \widehat{Y} = y \cdot s + ty \tag{1}$$

After that, a common reduction factor is used:

$$X' = \widehat{X} \cdot \underbrace{\tfrac{6}{10}}_{\text{LENGTH\_SCALE\_FACTOR}}, \quad Y' = \widehat{Y} \cdot \underbrace{\tfrac{6}{10}}_{\text{LENGTH\_SCALE\_FACTOR}} \tag{2}$$

Normalization to `picture` is needed because in *TeX* the origin is at the bottom left, whereas in *SVG* the y-axis grows downward, and because it is convenient for the coordinates to lie within the canvas. After computing the *bounding box* $(X_{\min}, X_{\max}, Y_{\min}, Y_{\max})$, each point is transformed as follows:

$$x_{\textit{TeX}} = (X' - X_{\min}) \cdot c, \qquad y_{\textit{TeX}} = (Y_{\max} - Y') \cdot c \tag{3}$$

The dimensions of the canvas are set so as to fit the whole figure:

$$W = (X_{\max} - X_{\min}) \cdot c, \qquad H = (Y_{\max} - Y_{\min}) \cdot c \tag{4}$$

2.2.5. Normalization of Sizes and Labels

After normalization, sizes are unified and labels are corrected, since the scale and fonts in *SVG/Graphviz* and *TeX* differ.

Fixing the radius of the places.
The radius of the places is fixed through a constant circle diameter:

$$\texttt{\textbackslash circle\{1.0\}} \quad \Rightarrow \quad r_{\textit{TeX}} = 0.5. \tag{5}$$

Correction of the labels of the places.
The label is shifted in the direction from the center toward its original position:

$$L_{\textit{new}} = C + k\,(L - C), \tag{6}$$

where k is `placeLabelOffsetFactor`.

Correction of the labels of the transitions.
The label of a transition is shifted upward by a constant:

$$y_{\textit{new}} = y_{\textit{old}} - \texttt{transitionLabelVerticalOffset}. \tag{7}$$

2.2.6. Clipping the Arcs to the Boundaries of the Elements

In order for the arcs not to enter the places/transitions, the endpoints are clipped to the boundary of the corresponding element. The arc is $(p_0, p_1, \ldots, p_{n-1})$, and only p_0 (in the direction toward p_1) and p_{n-1} (in the direction away from p_{n-2}) are corrected.

Clipping the endpoints of arcs at places.
For a place (a circle with radius r_{TeX}), the following is used:

$$P' = C + r \cdot \frac{(P - C)}{\|P - C\|}. \tag{8}$$

Clipping the endpoints of the arc at transitions.
The transition is approximated by a rectangle $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$. The direction is determined from the adjacent point Q through:

$$\Delta x = x_q - x_p, \qquad \Delta y = y_q - y_p. \tag{9}$$

and the criterion:

$$\text{horizontal, if } |\Delta x| \geq |\Delta y|, \qquad \text{vertical, if } |\Delta y| > |\Delta x|. \tag{10}$$

After that, the point is placed on the corresponding side:

$$P' = \begin{cases} (x_{\max},\, y_p), & \text{if } |\Delta x| \geq |\Delta y| \text{ and } \Delta x > 0, \\ (x_{\min},\, y_p), & \text{if } |\Delta x| \geq |\Delta y| \text{ and } \Delta x < 0, \\ (x_p,\, y_{\max}), & \text{if } |\Delta y| > |\Delta x| \text{ and } \Delta y > 0, \\ (x_p,\, y_{\min}), & \text{if } |\Delta y| > |\Delta x| \text{ and } \Delta y < 0. \end{cases} \tag{11}$$

Preserving the correct endpoint of the arrowhead.
After clipping, the last segment is generated as `\vector`, so that the arrowhead coincides with the new endpoint.

2.2.7. Pseudocode of the Algorithm

In a more general form, the steps can be synthesized as follows:

```
function svgToTeX(svgString):
1) DOM = parse SVG string into an XML DOM object
2) mainG = find <g id="graph0"> and parse
transform="translate(tx, ty) scale(s)"
globalTranslate = (tx, ty)
globalScale = s
3) Define LENGTH_SCALE_FACTOR = 0.6   # (2/3) shrink
4) Define pxToTeX = 0.1              # 1px -> 0.1 TeX units
```

```
# 5) Identify PLACES
for each g with class="node place":
read ellipse(cx, cy)
apply global transform + LENGTH_SCALE_FACTOR => (x', y')
store nodeShapes[placeName] = circle with center=(x', y')
and forced radius=5px

# 6) Identify TRANSITIONS
for each g with class="node transition":
read polygon points => bounding box => xMin, xMax, yMin, yMax
apply transform => store nodeShapes[transitionName] = rect

# 7) Capture LABELS for places and transitions
for each place:
read original label coords => (lx, ly)
apply transform => (lx', ly')
offset label => (lx'', ly'') = (cx', cy')
+ alpha*(lx'-cx', ly'-cy')
for each transition:
shift label upward => labelY' = labelY - someVerticalOffset

# 8) Identify EDGES
for each g with class="edge":
read origin, destination from class or <title> text
parse path (M x0 y0 L x1 y1 ...)
apply transform => rawPoints
# arrow polygon => transform as well

# 9) CLIP edge endpoints
for first and last rawPoints:
if shapes[origin] == circle => move boundary
if shapes[destination] == rect => clamp to bounding box
(similarly for the other end)

# 10) Compute bounding box over all x' and y'
xMinAll, xMaxAll, yMinAll, yMaxAll

# 11) Convert to TeX coords => (x_T, y_T)
x_T = (x' - xMinAll)*pxToTeX
y_T = (yMaxAll - y')*pxToTeX

# 12) Build a \begin{picture}(width, height) with lines for edges,
 circles for places, etc.

 return final TeX code as a string
```

After executing these steps, a *TeX* fragment is generated that can be be directly inserted into a scientific document, in accordance with the accepted convention for drawing GNs.

Chapter 3. *OnlineGN*: Simulator for Generalized-Net Models

The present chapter introduces the new GN simulator developed within the dissertation work – *OnlineGN*. The system requirements, architectural decisions, and the technology stack used are considered. The system is a standalone software component, and in future stages it is intended to expand compatibility through work with file formats generated by other simulators. The main functionalities are demonstrated in a video demonstration [42]. The source code of the application is published in [41].

3.1. Requirements and Goals of the System

The section defines functional and non-functional requirements for the web-based solution, which serve as criteria for the design and evaluation of *OnlineGN*.

3.1.1. Functional Requirements

OnlineGN visualizes GNs in a two-dimensional space (places, transitions, and connections) and performs simulation by applying Algorithm B (see **1.1.4.**), while respecting the correct processing order of transitions/states/tokens and supporting changes of characteristics, token splitting, and token merging. Two execution modes are implemented: trace mode (step by step) and continuous execution until a final state (or a stopping criterion).

The system allows restart of the simulation, interactive modification of coordinates of elements and connections (including points along a connection), as well as editing of characteristics of places and transitions with reflection in the simulation. Detailed information about the tokens at each step is provided, as well as a log of the operations within a single time step, describing the processing order according to priorities and capacities.

Loading/saving of nets in *JSON* format is supported, as well as storage and retrieval from a database, generation of a hyperlink for access to a stored net, generation of a representation in *TeX* format, addition of user-defined *JavaScript* functions, and dynamic change of the simulation speed. Automatic deletion of nets that have not been accessed for a predefined period is also implemented.

3.1.2. Non-Functional Requirements

The non-functional requirements cover performance, reliability, security, usability, and scalability. Thresholds are defined for response time up to 2 seconds under light load and recovery time after failure up to 30 minutes, and the choice is justified with respect to established practices and criteria for responsiveness and availability [43]. In addition, the system does not collect personal data and is aligned with the requirements of *GDPR*.

3.1.3. Constraints and Assumptions of the Environment

The system is deployed using the free cloud services of ***Render*** (for the server) and ***Atlas*** (for the database), and is therefore designed in accordance with the limited resources

provided by these services.

3.2. Architecture of the System

The section describes the architectural organization of *OnlineGN* and the interaction between the main components, corresponding to the implementation of **Task 4**.

3.2.1. General Architecture and Interaction Scenarios

OnlineGN is implemented as a three-layer architecture: data layer (DB), server layer (*API*/services), and client layer (user application), where the client carries out the main part of the business logic and the server provides auxiliary services. The database stores nets and states and is responsible for data consistency; the server acts as an intermediary between the client and the DB and performs supporting tasks (e.g. periodic deletion of outdated records); the client implements the *UI/UX*, visualization, simulation, and interactive modifications.

Typical scenarios are described: loading a net from the DB and starting a simulation, storing a net with a returned hyperlink, as well as periodic deletion of unused nets by the server.

3.2.2. Physical/*Deployment* Architecture and Cloud Infrastructure

The deployment is implemented through ***Render*** and ***MongoDB Atlas***. The server instance is maintained in the infrastructure of *Render* and delivers the client application to the browser, while the client–server communication is over *HTTPS*. The database instance is maintained in *Atlas*; the server communicates with the DB over *HTTPS* and authenticates with a username and password [51, 52].

3.2.3. Technology Stack and External Dependencies

The database is *MongoDB* (version 8.0), where the data are stored as documents with a *JSON*-like structure and a physical representation in *BSON*.

The server side is implemented with *Node.js* and *Express.js* for a *REST API*, with unified routing, validation, and centralized error handling. The used *Node.js* version is 22.x (*LTS*), selected because of its predictable life cycle and long-term security updates; the approach supports a low migration cost to the next *LTS* version. For the server, external libraries such as `body-parser`, `cors`, `cron`, `express`, `is-svg`, and `mongoose` are used, and for development – `nodemon`.

The client application is developed with *Angular* (version 16) on top of *TypeScript* and *SCSS*, following good practices for maintainability and future transition to the next *LTS* version [53, 54]. In addition to the core *Angular* packages, *Angular Material* and *CDK* are used for *UI* components, `ngx-toastr` for notifications, as well as `d3` and `d3-graphviz` for interactive visualization, animations, and rendering of *Graphviz DOT* in the browser [55].

3.3. Design and Modules of the Application

In the context of **Task 4**, the modular organization of *OnlineGN* is presented. The main functional components (storage, visualization, editing, simulation, import/export, and *API*) and the interaction between them are described. The modular design aims at clear separation of responsibilities (*separation of concerns*), facilitated maintenance, and scalability.

3.3.1. Data Processing Module (*GenNet Persistence*)

GenNet Persistence implements the persistence of net data and provides the basic operations for working with records. The following submodules are included:

- ***Loading*** – loading a net by unique identifier, while updating the information about last use upon access;
- ***Saving*** – storing/updating the data of a net so that it is available for subsequent retrieval;
- ***Expiration*** – automatic deletion of nets that have not been accessed in the last 90 days.

3.3.2. *JSON* Format for Description of Generalized Nets

The dissertation describes a developed *JSON* format for unified, machine-readable representation of GNs, suitable for direct interpretation by the simulator. The data format is defined by the *GenNetExportModel* interface. A single document describes the structure of the net, the initial configuration of the tokens, execution parameters, user-defined functions, and a pre-generated visualization. The format is organized into four main sections: `genNetRaw`, `settings`, `code`, and `svg`.

Root structure.
The root object contains:

- ***`genNetRaw`*** – declarative description of the GN;
- ***`settings`*** – settings of the simulation environment;
- ***`code`*** – user *JavaScript* code;
- ***`svg`*** – pre-generated visualization of the net.

In this way, a clear distinction is ensured between structural data and behavioral specification.

Section `genNetRaw`.
The `genNetRaw` section contains the main data for the net and includes the collections `places`, `transitions`, and `tokens`.

- ***places*** – an array of objects of type *Place*, through which the identifier, name, transformation function of the characteristics, merge function, priority, and capacity of the place are specified;
- ***transitions*** – an array of objects of type *Transition*, through which the identifier, name, split function, priority, and the connections between the input and output places through objects of type *TransitionItem* are specified;
- ***tokens*** – an array of objects of type *Token*, defining the initial configuration of the simulation through identifier, name, current place, priority, and initial characteristics.

The dissertation gives example *JSON* representations of objects of type *Place*, *Transition*, and *Token*.

Section *settings*.
The `settings` section contains parameters that influence execution without changing the structural model. In the considered implementation, the object `simulationSettings` is included, through which the duration of one simulation step is specified by means of `stepDurationMS`.

Section *code*.
The `code` section contains a string with user *JavaScript* code used in the simulation. In order for the code to be accessible in the context of the simulator, the object ***this.globalObj*** is defined at the outermost level, serving as a namespace for user definitions.

Within ***this.globalObj*** there are:

- ***tokens*** – a reference to the current collection of tokens;
- ***functions*** – functions invoked by their names from the fields in ***genNetRaw***;
- ***globals*** – additional values used as auxiliary context during execution of the simulation.

Thus, the format allows declarative specification of the net, while the concrete behavior and the conditions during the simulation are defined centrally and linked through the names of the functions. The dissertation also presents an example *JSON* model of a GN with two places, one transition, and one token.

Section *svg*.
The `svg` section contains an *SVG* string for visualization of the net. If such a string is not provided, the program generates an *SVG* visualization through the automatic drawing algorithm described in **2.1.**.

3.3.3. Graphical Visualization Module (*SVG/Graphviz*)

The module for GN visualization in the client application is discussed: loading a ready *SVG* or generating an *SVG* from a *Graphviz* string, drawing/redrawing elements, animations upon changes, processing user events, and saving the current state as *SVG*.

3.3.4. Simulation Module for the Movement of Tokens

The *Simulation* module is described, which implements execution of the simulation according to the rules of GNs and synchronizes the logical changes with the visualization: a single move (*Single move*), full simulation (*Full simulation*), and restart (*Reset*) through a visualization update submodule (*Update visualization*).

3.3.5. *API* and Script Interface

The server provides two endpoints for access from the client.

- `GET /api/genNets/:id` – retrieves a net by identifier and returns `status code 200` on success or `status code 404` if the net is not found
- when the net is found, an object with fields `id`, `content`, and `lastFetchedAt` is returned
- `POST /api/genNets` – stores the submitted net in the database
- in the request, the field `content` is submitted, and upon successful storage `status code 201` and `id` are returned
- upon unsuccessful storage, `status code 400` is returned

At this point in the dissertation, example requests and *JSON* responses from the *API* are given.

3.3.6. Module for Importing and Exporting Nets

The import and export module provides loading and saving of the net data in different formats, in accordance with the interface described in **item 3.3.5.**. Export to a *JSON* file, *TeX* format, and *JSON* format in a database is supported, as well as import from a file and from stored *JSON* data in the database. Functionally, the module is divided into two submodules: *Import* (loading) and *Export* (saving).

Submodule *Import*.

Import includes:

- ***JSON*** – reading and validation of input data in *JSON* format against the interface from **item 3.3.5.**, where upon successful validation the necessary objects for visualization are created; it is also used when loading from the server, since the data arrive in *JSON*;

- ***File*** – processing of files provided by the user, where for valid *JSON* the content is passed to the *JSON* submodule.

Submodule *Export*.

Export includes:

- ***TeX*** – conversion of the current visualization of the net into *TeX* format for integration into scientific documents;
- ***JSON*** – generation of a *JSON* representation of the currently loaded net (including data for tokens and visualization) according to the format from **item 3.3.5.**; it is also used when storing in the database;
- ***File*** – creation of a file for download by the user, where the data are generated by the *JSON* submodule and written into a file.

3.3.7. Module for Visualization of Dynamic Processes and Animation

For visualization of token movement in the simulator, the *d3* library is used, where each token is represented by an *SVG* group (graphical symbol and text). The movement is implemented along trajectories defined by means of *SVG path* elements, while the animation is executed through interpolation: the path length is computed (`getTotalLength()`), a transition is defined with `attrTween` and parameter $t \in [0, 1]$, and the current position is obtained through `getPointAtLength(t * l)` and applied through the transformation *translate*. After completion of the transition, the element is removed from the *DOM*, which frees resources and reflects the traversal of the token along the arc.

3.4. Algorithms and Implementation Details

The main algorithms and the key implementation decisions used in the realization of *OnlineGN* are described.

3.4.1. Developed Algorithm 1: Automatic Drawing of GNs

Algorithm 1 is implemented programmatically in the *OnlineGN* system as a separate implementation, which automates GN drawing on the basis of the structural data of the net. The implementation generates coordinates and a graphical representation, which is used directly by the visualization module.

3.4.2. Developed Algorithm 2: Conversion of *SVG* to *TeX*

Algorithm 2 is implemented in *OnlineGN* for automatic conversion of the current *SVG* visualization into a *TeX* representation. The implementation transforms the main graphical primitives and textual elements, providing export in *TeX* format for inclusion in scientific documents.

3.5. Interface and User Experience

The chapter presents the implementation of **Task 5** and **Task 6**, considering the design of the user interface and the main principles of the user experience in *OnlineGN*. The emphasis is on navigation, visual components, and the way in which the interface supports interactive work with the simulator, including visualization, personalization, and editing of GN parameters.

3.5.1. Navigation and Visual Components of the Web Client

The client application combines the main functionalities in one primary workspace: visualization of the net, control of the simulation, and access to operations for loading/saving and exporting. The interface provides a direct view of the current state of the net and of the simulation step, as well as convenient access to the main actions through functional buttons and dialogs.

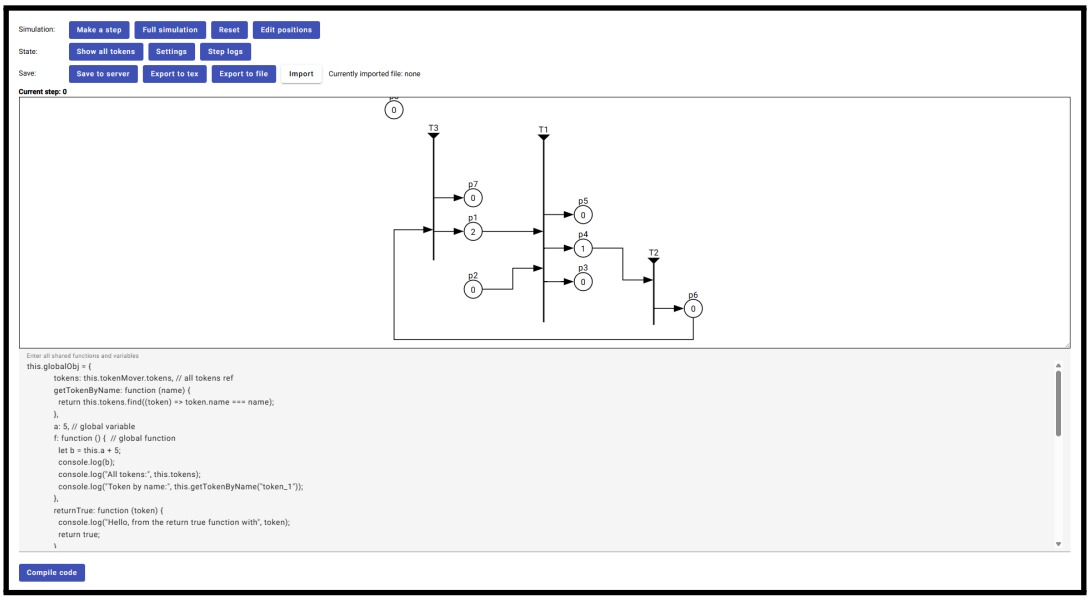

Figure 16 – Main workspace and controls.

For the implementation of **Task 6**, an editing mode for the elements of the net has been introduced, through which a place/transition/arc is selected and its coordinates and parameters are modified. This enables interactive correction of the visualization and adjustment of the net objects directly in the workspace.

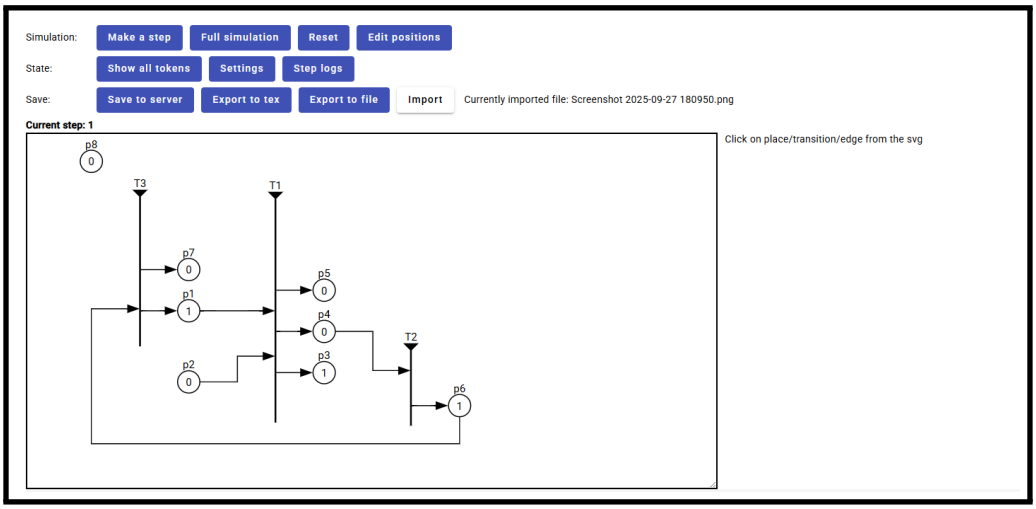

Figure 22 *– Coordinate editing mode ("Edit positions").*

Predicates and movement rules are visualized through specialized components for editing and viewing transitions, including a representation of the predicate matrix, which reflects the conditions for passage between the input and output places of a transition.

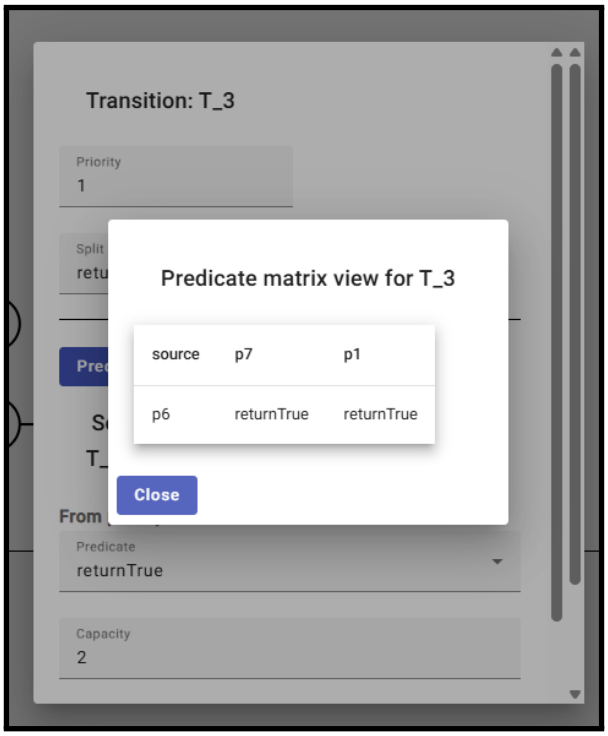

Figure 44 *– Predicate matrix of a specific transition.*

3.5.2. Help System, Documentation, and Localization

A help system is provided, complementing the web client through embedded documentation and contextual help. Capabilities for interface localization are also ensured, which facilitates the use of the simulator in an educational and research context.

Chapter 4. Applications of Generalized Nets and the *OnlineGN* Simulator

The present chapter considers four new GN models of processes that had not previously been described by means of the GN apparatus. Each of the developed models has been simulated and verified through the software environment *OnlineGN*. The exposition and the simulations carried out in these four sections constitute the solution of **Task 8** and **Task 9**.

4.1. Simulation of Finite Automata through Generalized Nets

The dissertation considers how a finite automaton can be described and simulated with the help of GNs. The correspondence between the states and transitions of the automaton and the corresponding GN elements is constructed so that the model becomes an operational scheme. A concrete example of a finite automaton is also considered, for which the corresponding GN is constructed and a real simulation is realized with *OnlineGN*. In the development and analysis, the approach proposed in [56] is followed.

In [57] a way of representing automata through GNs is considered, but the main emphasis is on the theoretical proof of existence; here, a construction directly applicable to simulation in GN software environments is given, with concrete structural constraints and optimizations.

4.1.1. Basic Notions of Finite Automata

Finite alphabets and languages.
The standard notions of a *finite alphabet* Σ, a *word* $\alpha \in \Sigma^*$, and a *language* $L \subseteq \Sigma^*$ are used, where the empty word is denoted by ε.

Deterministic Finite Automaton (DFA).
A DFA is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_{start}, F)$ [58, 59], where $\delta : Q \times \Sigma \to Q$ is total. A word α is accepted if and only if $\delta^*(q_{start}, \alpha) \in F$.

Nondeterministic Finite Automaton (NFA).
An NFA is a 5-tuple $\mathcal{N} = (Q, \Sigma, \Delta, q_{start}, F)$ [58, 60], where $\Delta : Q \times \Sigma \to \mathcal{P}(Q)$. A word α is accepted if and only if $\Delta^*(\{q_{start}\}, \alpha) \cap F \neq \varnothing$.

Comparison between DFA and NFA.
A DFA follows a single computation path, whereas an NFA allows branching (parallel) paths; the two models recognize the same class of languages (Rabin–Scott theorem [61], *powerset construction*).

4.1.2. Representation of Nondeterministic Finite Automata through Generalized Nets

GNs provide a flexible framework for simulation of parallel processes [8, 9], which makes them suitable for modeling the way words are accepted by an NFA. The GN is constructed so as to cover all possible parallel computation paths: each *state* q_i is associated with a transition Z_i, and each *arc* $q_i \to q_j$ is represented as a *place* (output for Z_i and input for Z_j). A special start place p_{start} and a final transition Z_{final} are introduced, which realizes exiting the net upon acceptance.

Tokens and characteristics.
The active objects are *tokens*, each token corresponding to a possible computation path in the NFA. As a characteristic, the remainder of the input word (`word`) is carried; when passing through a place, the first symbol is removed (except for the initial movement from p_{start}). When branching occurs in the NFA, the token is duplicated into several copies with the same current remainder of the word.

Indexed matrices and predicates.
The construction is formalized through *indexed matrices* and predicates of the form $S(x, y)$ (for compatibility between the next input symbol and an admissible transition in the NFA), as well as the predicate E (empty characteristic "word") as the condition for passage through Z_{final}.

Theorem. Every NFA can be represented by a corresponding GN (see [8, 9]).

4.1.3. Detailed Example of a Nondeterministic Finite Automaton Represented as a Generalized Net

An example NFA with two final states (q_3 and q_5) is considered, which is nondeterministic because of two arcs labeled *b* outgoing from the same state. On the basis of this automaton, the corresponding GN is constructed with one transition for each state and an additional final transition Z_{final}.

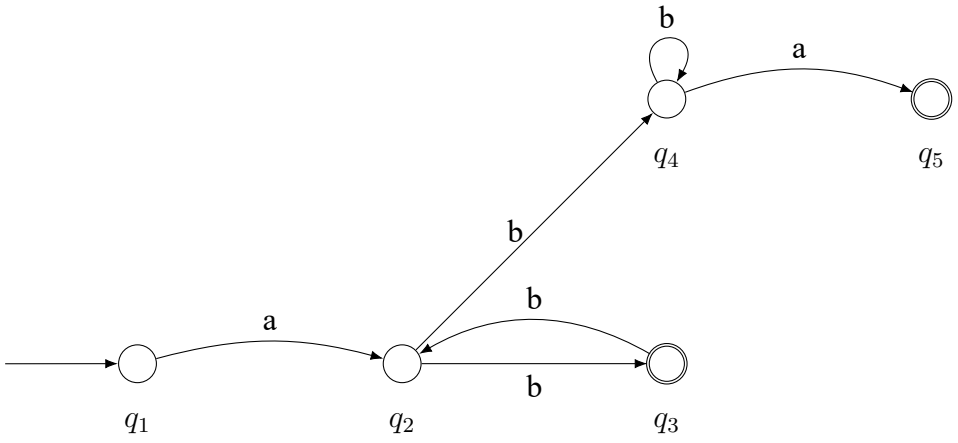

Figure 52 *– Example NFA with two final states: q_3 and q_5.*

4.1.4. Simulation of Multiple Nondeterministic Finite Automata through a Single Generalized Net

It is shown that a finite set of NFAs can be represented through a single GN, by combining the GNs corresponding to each NFA from the set, using the property that $E_1 \cup E_2$ is a GN whenever E_1 and E_2 are GNs [8]. This allows simultaneous checking of input strings through multiple NFAs by means of one single GN.

4.1.5. Simulation of a Single Nondeterministic Finite Automaton with the Help of *OnlineGN*

A real simulation in *OnlineGN* of the constructed GN is demonstrated, which recognizes the word `abbba`. In the simulation, a slightly modified notation is used (since subscripts are not supported), e.g. q_{i-j} is written as `qi-j`. The GN starts with a single token in `p-start`, carrying the characteristic `abbba`; during the successive steps, the movement of the tokens and the development of the characteristic (reading the symbols) are visualized. Upon acceptance, the token reaches the final place, and the characteristic becomes the empty string.

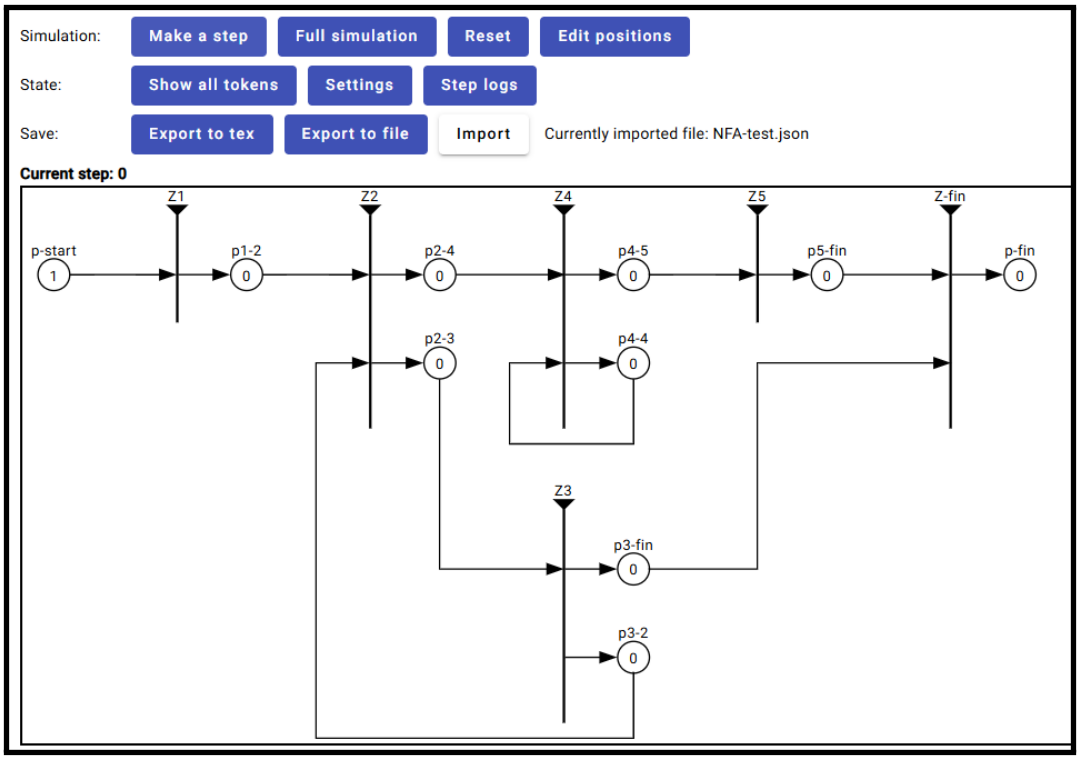

Figure 54 *– Initial configuration of the generalized net with the starting place (p_{start}).*

Figure 55 *– Token in p_{start} with characteristic $abbba$.*

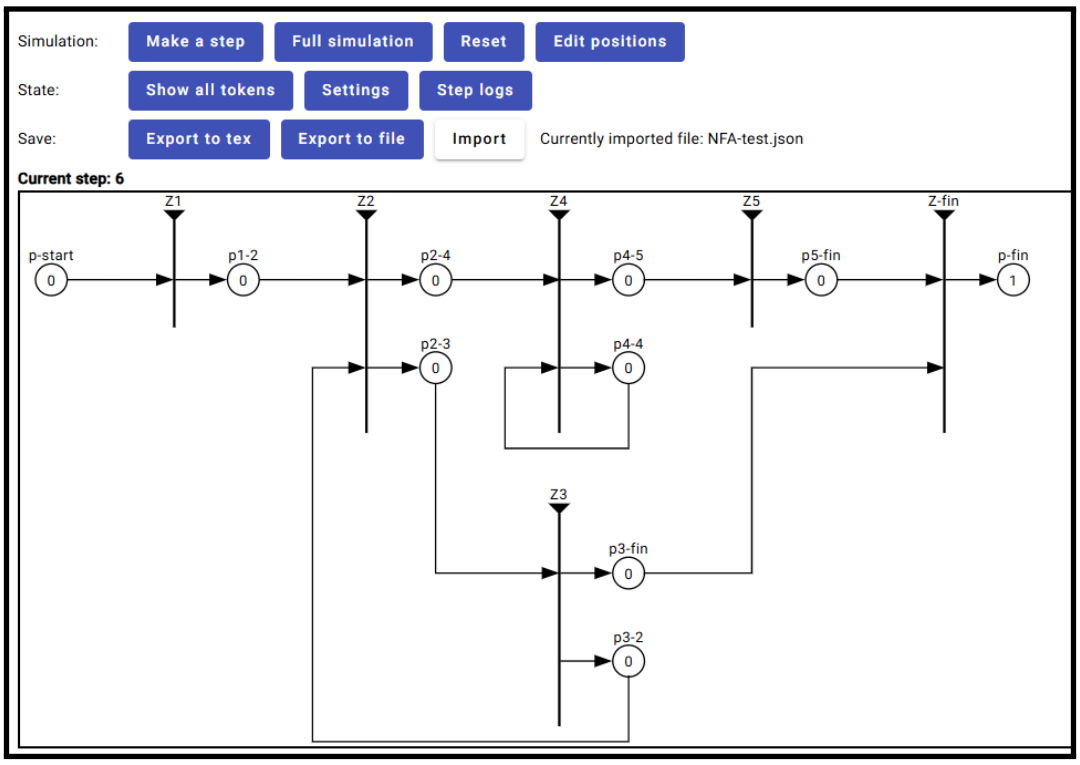

Figure 58 *– The token has reached the final place p_{fin}.*

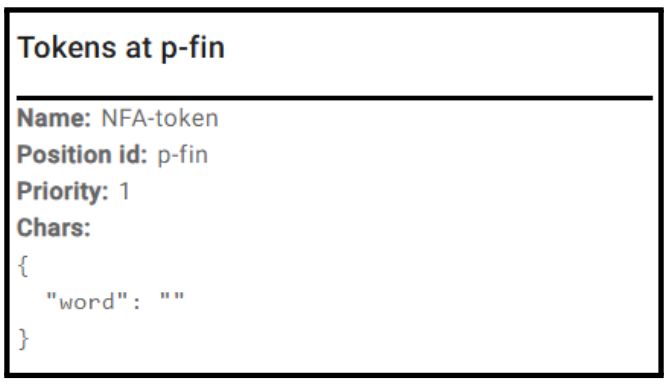

Figure 59 *– The token's characteristic becomes the empty string upon acceptance.*

4.1.6. Conclusion

The dissertation considers how GNs can be used for simulation of DFAs and NFAs by associating states with transitions and arcs with places, thus capturing the parallel computation paths. The architecture based on tokens and characteristics allows traceable simulation, while *OnlineGN* is used to demonstrate the execution step by step and to show acceptance clearly (final place and empty characteristic).

4.2. Parallelization of the Towers of Hanoi Problem through Generalized Nets

A *parallel* version of the classical Towers of Hanoi problem is considered, in which, in one step, simultaneous movement of more than one ring is allowed. The problem is formalized by means of the GN apparatus, and the simulation realization in *OnlineGN* demonstrates the executability of the model and the effect of parallelism in exponential combinatorial processes.

4.2.1. Description of the Towers of Hanoi Problem

The classical Towers of Hanoi problem [63] consists of three rods and n rings of different sizes, initially arranged on the first rod; a larger ring may not be placed on a smaller one (**Figure 60**). The goal is to transfer all rings from the first to the third rod while respecting the rule. When exactly one ring is moved per step, a minimum of $2^n - 1$ steps is required [64, 65]. The present chapter describes the process through GNs.

4.2.2. Parallelization of the Towers of Hanoi: Conceptual Overview

The classical constraints are: **(i)** at each step only one ring is moved and **(ii)** a larger ring may not be placed on a smaller one. In the parallel variant, **(i)** is relaxed: in one *parallel step*, a packet of up to k rings can be *lifted* simultaneously (from the tops of an arbitrary combination of rods), after which they are *placed* one by one according to the priority rule (first the largest, last the smallest). The lift–place cycle counts as one parallel step.

4.2.3. Configuration of the System

- **Rods.**
  Three rods R_1 (source), R_2 (auxiliary), R_3 (target), each with a stack of rings in strictly decreasing order of size from bottom to top.
- **Rings.**
  n rings, labeled from 1 (smallest) to n (largest).
- **Air buffer.**
  After lifting, the selected up to k rings are “in the air”, while the internal order of the rings coming from the same rod is preserved.

4.2.4. Definition of a Parallel Step

Each step has two phases:

1. **Lift phase.** Up to k rings are selected from the tops of the rods and removed *simultaneously*.
2. **Placement phase.** Placement according to the *priority rule*: (1) release in strictly decreasing order of size; (2) placement on an arbitrary rod without violating the local order by size.

The minimum number of parallel steps is denoted by $H(n, k)$.

Theorem 1. $H(n, k) = 2^{\lceil \frac{n}{k} \rceil} - 1$.

The proof groups the n rings into $M = \lceil n/k \rceil$ blocks of size at most k, which are treated as "super-rings", and reduces the parallel model to the classical recursion on M elements (the upper and lower bounds coincide).

Observation (Optimality of block movements in the parallel variant of the Towers of Hanoi).
An optimal strategy always transfers exactly k consecutive rings from one rod to another; each block behaves as a "super-ring", whereby the problem is reduced to the classical Towers of Hanoi on $\lceil n/k \rceil$ super-rings.

4.2.5. Generalized-Net Model for the Parallel Towers of Hanoi

A GN model is introduced for simulation of the k-parallel rule. The GN contains n ρ-tokens ("rings") and k χ-tokens ("hands"). The χ-tokens allow, in one parallel step, at most k ρ-tokens to be lifted simultaneously and placed according to the priority rule. The net is constructed so as to realize the optimal block strategy and to be executable in *OnlineGN*.

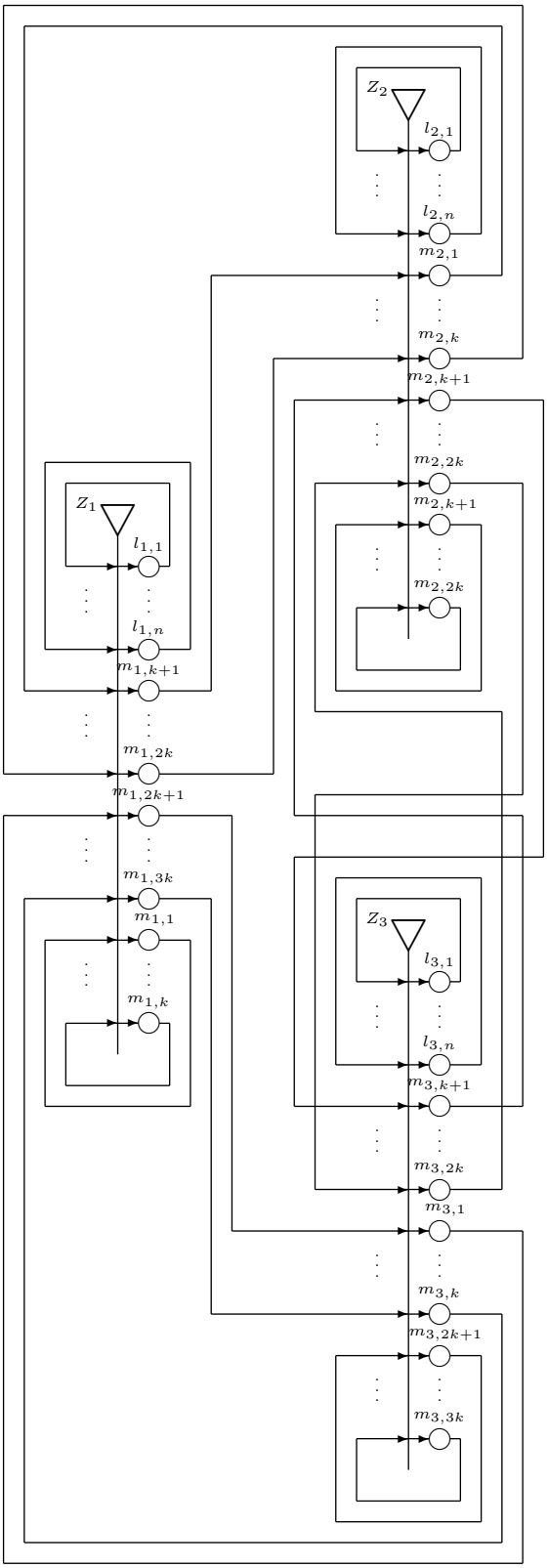

Figure 61 *– Generalized Net for the Tower of Hanoi: $n\rho$-tokens and $k\chi$-tokens.*

Tokens and movements.

The following conventions are used:

- ρ-token: ring; initial characteristic – the size.
- χ-token: “hand”; characteristic – the index of the “hand”.

A distinction is made between a *move of the puzzle* and a *net move*. Each move of the puzzle is realized by **two net moves**: (i) positioning the ρ- and χ-tokens at the output places of the transition of the source rod; (ii) transferring them to the input–output places of the transition of the target rod. In this way, the GN simulates the transfer of ρ-tokens between rods through a sequence of net moves.

Transitions and places.

The GN contains **three transitions** Z_1, Z_2, Z_3 (one for each rod). For the ρ-tokens, n **input–output places** $l_{i,j}$ ($i \in \{1,2,3\}$, $j = 1,\dots,n$) are used; a ρ-token in $l_{i,j}$ means “a ring of size j is on rod i”. For the χ-tokens, **holding places** $m_{i,j}$ ($j \in [1,3k]$) are used, which encode the current/next rod for taking and transferring, depending on the parity of the net step. The general form of the GN is shown in **Figure 61**.

Indexed matrices.

The predicates over the indexed matrices of the GN are:

- $U_{x,y}$: “a ρ-token is moved from x to y in this step of the puzzle”;
- $W_{x,y,i,j}$: selection of a χ-token for moving a ρ-token and ensuring the correct position in the block mode;
- P_i: “a ρ-token has characteristic weight i” (correct positioning after the move);
- S_x: placing the χ-tokens in output places according to the forthcoming move.

The indexed matrices for odd and even net moves are given in **Figures 63 and 64**.

	$l_{a,1}$	$\cdots$	$l_{a,n}$	$m_{a,(a-1)*k+1}$	$\cdots$	$m_{a,a*k}$	$m_{a,(b-1)*k+1}$	$\cdots$	$m_{a,b*k}$	$m_{a,(c-1)*k+1}$	$\cdots$	$m_{a,c*k}$
$l_{a,1}$	F	$\cdots$	F	F	$\cdots$	F	$W_{a,b,1,1}$	$\cdots$	$W_{a,b,1,k}$	$W_{a,c,1,1}$	$\cdots$	$W_{a,c,n,k}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$l_{a,n}$	F	$\cdots$	F	F	$\cdots$	F	$W_{a,b,n,1}$	$\cdots$	$W_{a,b,n,k}$	$W_{a,c,n,1}$	$\cdots$	$W_{a,c,n,k}$
$m_{a,(a-1)*k+1}$	F	$\cdots$	F	F	$\cdots$	F	$U_{a,b}$	$\cdots$	F	$U_{a,c}$	$\cdots$	F
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$m_{a,a*k}$	F	$\cdots$	F	F	$\cdots$	F	F	$\cdots$	$U_{a,b}$	F	$\cdots$	$U_{a,c}$
$m_{b,(a-1)*k+1}$	F	$\cdots$	F	F	$\cdots$	F	$U_{a,b}$	$\cdots$	F	$U_{a,c}$	$\cdots$	F
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$m_{b,a*k}$	F	$\cdots$	F	F	$\cdots$	F	F	$\cdots$	$U_{a,b}$	F	$\cdots$	$U_{a,c}$
$m_{c,(a-1)*k+1}$	F	$\cdots$	F	F	$\cdots$	F	$U_{a,b}$	$\cdots$	F	$U_{a,c}$	$\cdots$	F
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$m_{c,a*k}$	F	$\cdots$	F	F	$\cdots$	F	F	$\cdots$	$U_{a,b}$	F	$\cdots$	$U_{a,c}$

Figure 63 *– Indexed matrix of Z_X for the odd network moves (beginning of a puzzle move).*

	$l_{a,1}$	$\cdots$	$l_{a,n}$	$m_{a,(a-1)*k+1}$	$\cdots$	$m_{a,a*k}$	$m_{a,(b-1)*k+1}$	$\cdots$	$m_{a,b*k}$	$m_{a,(c-1)*k+1}$	$\cdots$	$m_{a,c*k}$
$l_{a,1}$	F	$\cdots$	F	F	$\cdots$	F	F	$\cdots$	F	F	$\cdots$	F
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$l_{a,n}$	F	$\cdots$	F	F	$\cdots$	F	F	$\cdots$	F	F	$\cdots$	F
$m_{a,(a-1)*k+1}$	F	$\cdots$	F	F	$\cdots$	F	F	$\cdots$	F	F	$\cdots$	F
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$m_{a,a*k}$	F	$\cdots$	F	F	$\cdots$	F	F	$\cdots$	F	F	$\cdots$	F
$m_{b,(a-1)*k+1}$	P_1	$\cdots$	P_n	S_a	$\cdots$	F	S_b	$\cdots$	F	S_c	$\cdots$	F
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$m_{b,a*k}$	P_1	$\cdots$	P_n	F	$\cdots$	S_a	F	$\cdots$	S_b	F	$\cdots$	S_c
$m_{c,(a-1)*k+1}$	P_1	$\cdots$	P_n	S_a	$\cdots$	F	S_b	$\cdots$	F	S_c	$\cdots$	F
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$m_{c,a*k}$	P_1	$\cdots$	P_n	F	$\cdots$	S_a	F	$\cdots$	S_b	F	$\cdots$	S_c

Figure 64 *– Indexed matrix of Z_X for the even network moves (end of a puzzle move).*

Theorem 2. *The GN describing the Hanoi puzzle with n ρ-tokens and k χ-tokens transfers the ρ-tokens from the first to the third rod in $2^{\left\lceil \frac{n}{k} \right\rceil} - 1$ moves of the puzzle.*

The proof is a direct application of **Theorem 1** for $N = n$, $K = k$, through grouping the ρ-tokens into blocks of size at most k.

4.2.6. Simulations with *OnlineGN*

The developed GN model has been simulated with *OnlineGN*. Due to the size of the net, screenshots are used that show the active segments. For the example $n = 3$, $k = 2$, the initial configuration has the ρ- and χ-tokens at Z_1 (**Figure 65**), after which intermediate positioning

toward m-places directed to Z_2 is observed (**Figure 66**), and a final configuration with all ρ-tokens in Z_3 (**Figure 67**). For all tested pairs (n, k), the reported number of steps coincides with $2^{\lceil n/k \rceil} - 1$.

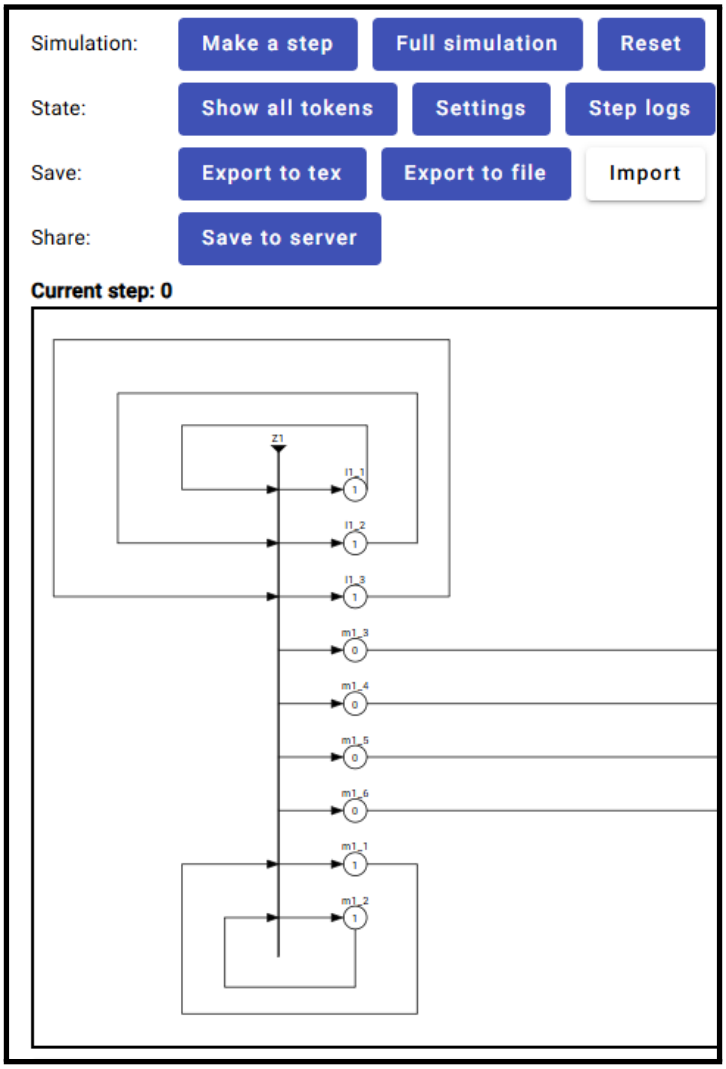

Figure 65 *– Generalized Net for the Tower of Hanoi with 3 ρ-tokens and 2 χ-tokens in OnlineGN.*

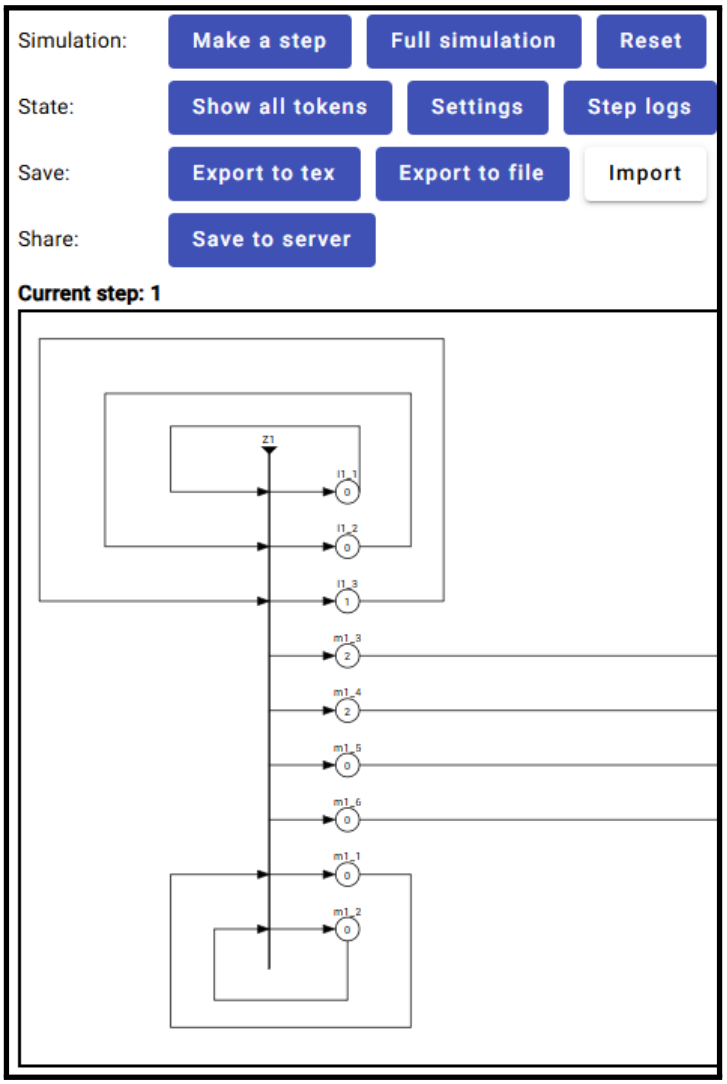

Figure 66 – *The ρ- and χ-tokens are positioned for movement to the next transition (rod).*

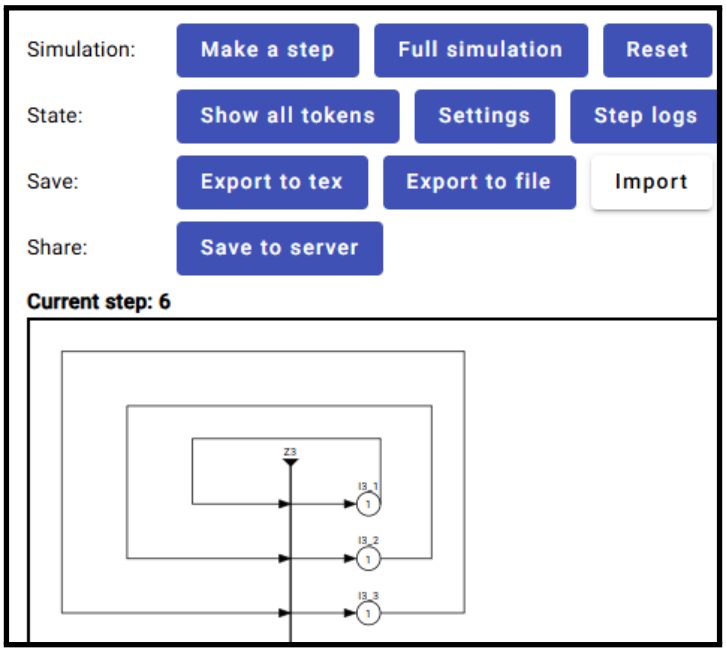

Figure 67 *– Final configuration: all ρ-tokens occupy the output positions of Z_3 (Tower of Hanoi interpretation: third rod).*

4.2.7. Other Studies on Parallel Solving of Problems

Other parallel solutions of exponential problems are considered.

4.2.8. Conclusions from the Results

The developed GN is a direct, executable model of the k-parallel rule of the Towers of Hanoi. When executed in ***OnlineGN***, the simulation generates the expected number of moves $2^{\left\lceil \frac{n}{k} \right\rceil} - 1$ for all tested n and k, which confirms the correctness and numerical validity of the model. Increasing k shortens the critical path, but the gains decrease because of the cost of synchronization (lifting and placing under a global constraint on the order), i.e. the process is *synchronization-limited*.

4.3. Generalized-Net Model of the Production of Gas and Polymeric Products in a Petroleum Refinery

The dissertation considers a GN model of the production of fuel gas, LPG (*LPG*), propylene, and polypropylene, and its software simulation through *OnlineGN*. The construction of the model and the details of the formalization are presented in [78].

4.3.1. Introduction to the Production of Products in a Petroleum Refinery

Modeling the processes of production of refined products in a petroleum refinery supports production planning and analysis of efficiency. In the literature, partial models of

separate aspects of the problem can be found [79, 80, 81], whereas in GNs the cause–effect dependencies are specified through the predicates of the transitions and the characteristics of the tokens.

In previous studies [82, 83] it has been shown that the production of automotive gasoline [82] and diesel fuel [83] can be modeled through GNs; the present chapter complements this line with the production of fuel gas, *LPG*, propylene, and polypropylene and its verification through *OnlineGN*.

4.3.2. Technological Scheme for the Production of Fuel Gas, LPG, Propylene, and Polypropylene in a Petroleum Refinery

The technological production chain for fuel gas, LPG, propylene, and polypropylene is presented in **Figure 69** (in the dissertation it is used as a basis for formalization in GNs).

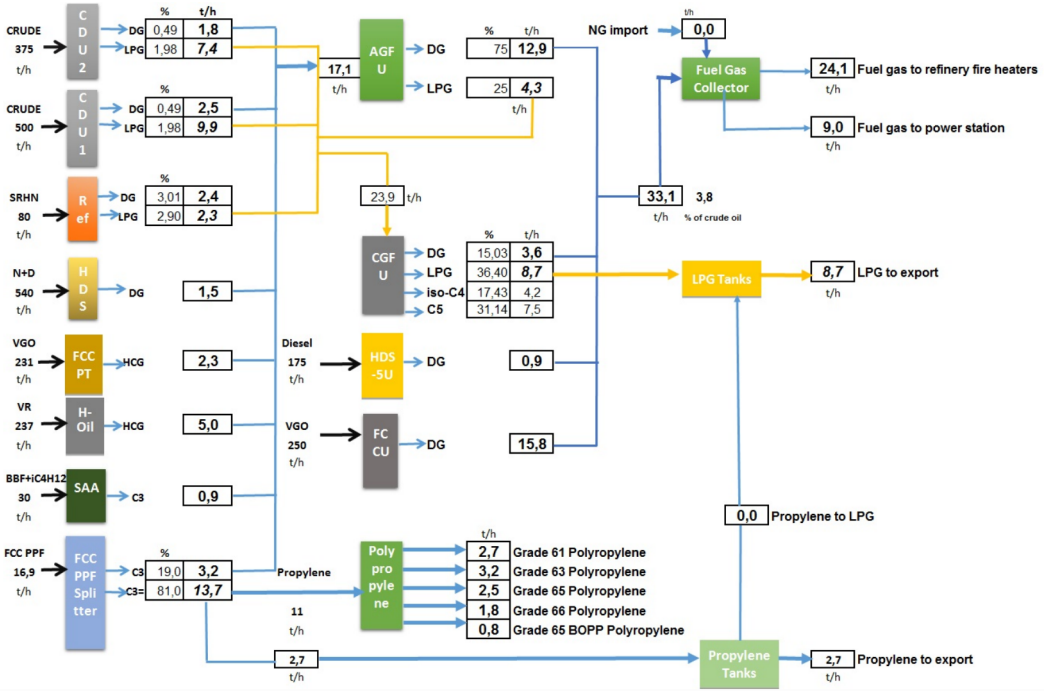

Figure 69 *– Processing scheme of fuel gas, LPG, propylene, and polypropylene production in a petroleum refinery.*

The production of petroleum gas products depends on the origin of the crude oil and the operating regimes in the process units. Depending on market requirements, the liquid propylene may be exported as polymer-grade propylene or used as a component of LPG; the polypropylene unit produces various grades of polypropylene products.

4.3.3. Main Results: Generalized-Net Model

The dissertation considers the GN model (**Figure 70**), which describes the technological chain under consideration through places, transitions, and tokens with characteristics. The model contains 17 transitions, 55 places, and 47 token types, including: streams of raw

materials/intermediate fractions (σ_i), process units (ρ_i), and products (α, β, γ, δ, ε), according to [78].

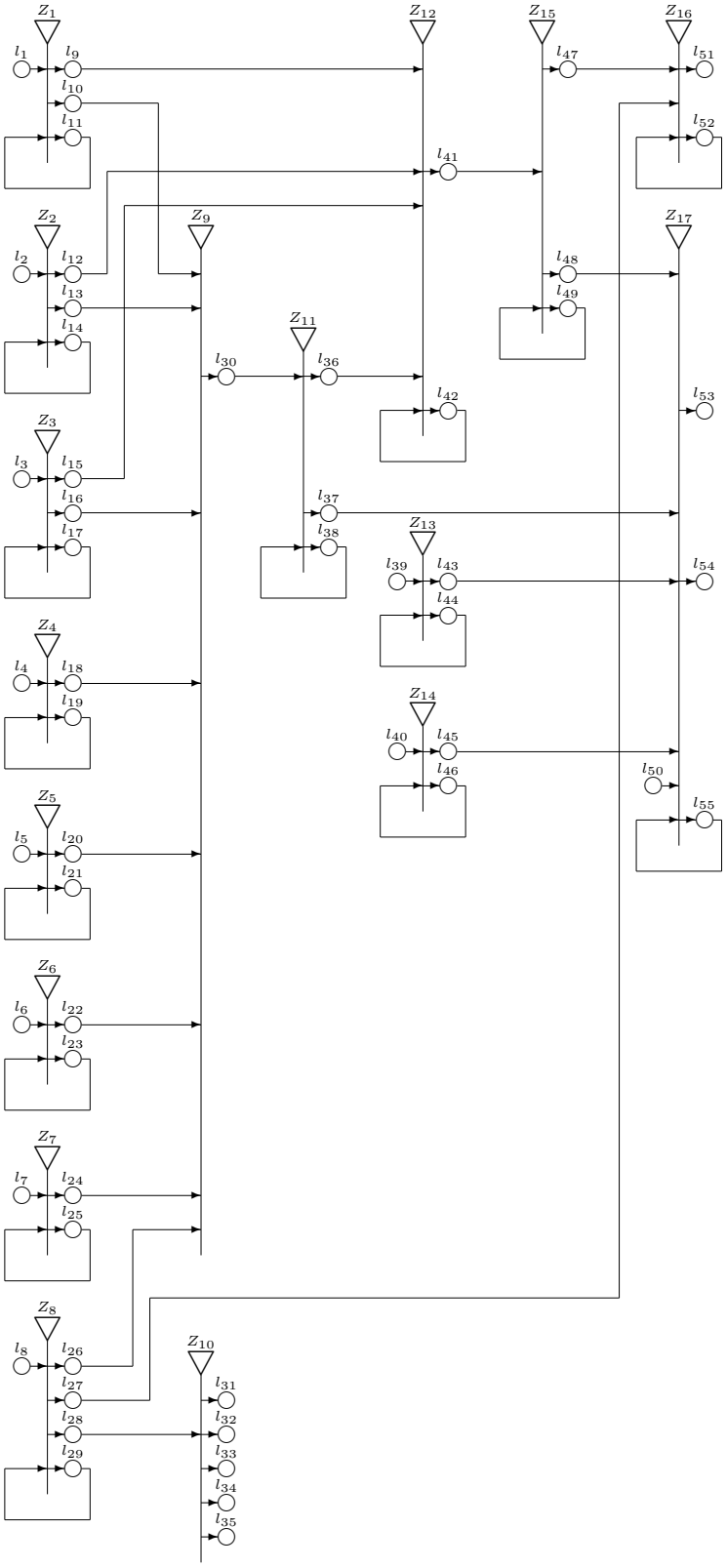

Figure 70 *– Generalized net model.*

The logic of the process is defined through indexed matrices with predicates for each transition (e.g. conditions of the type “there is a request for product ...”), which determine the movement of tokens from the input to the output places and the change of their characteristics during simulation.

4.3.4. Remarks on the Results and the Generalized-Net Model

The production of fuel gas, *LPG*, propylene, and polypropylene is a complex parallel process with multiple technological units and constraints, which can be represented through GNs.

4.3.5. Simulation with *OnlineGN*

In order to verify in practice whether the proposed GN model correctly describes the process of production of fuel gas, *LPG*, propylene, and polypropylene in a petroleum refinery, we implemented and simulated it in *OnlineGN*. In this way, we obtain not only a visualization of the structure (places, transitions, and connections), but also the possibility to trace how the tokens “move” and how their characteristics change when passing through the separate units.

Due to the size of the model, its visual representation is divided into three parts: upper, middle, and lower. Here, only the upper part is shown in **Figure 71**.

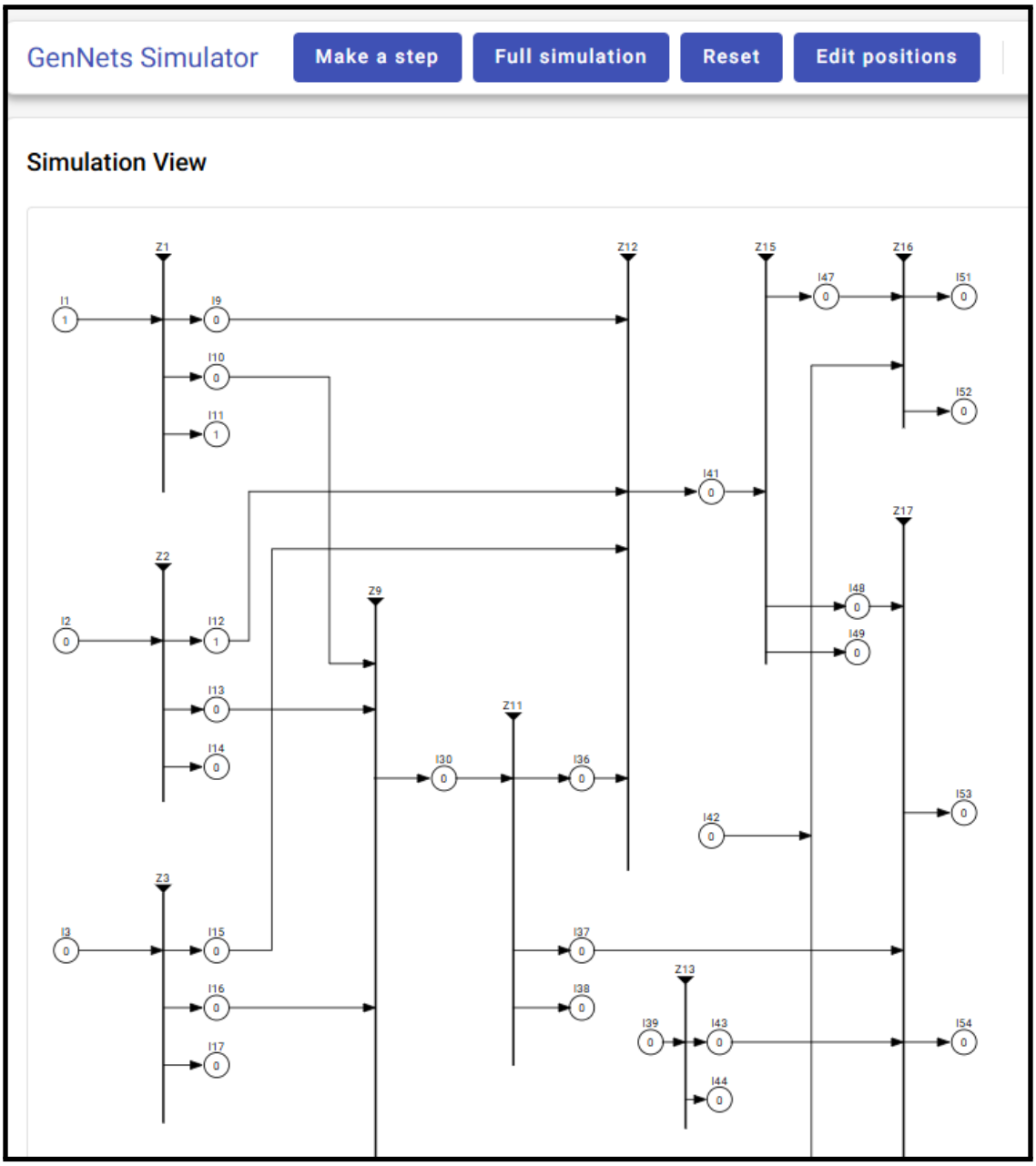

Figure 71 *– Upper part of the generalized net model in OnlineGN.*

The next important step is the logic of the transitions: for each transition, indexed matrices are defined in which the predicates from the theoretical description are embedded. It is precisely these predicates that "decide" whether a given token can pass from the input to the output places of the transition. In **Figure 74** an example is shown with an indexed matrix for transition Z_1, which in the model corresponds to *CDU-2*.

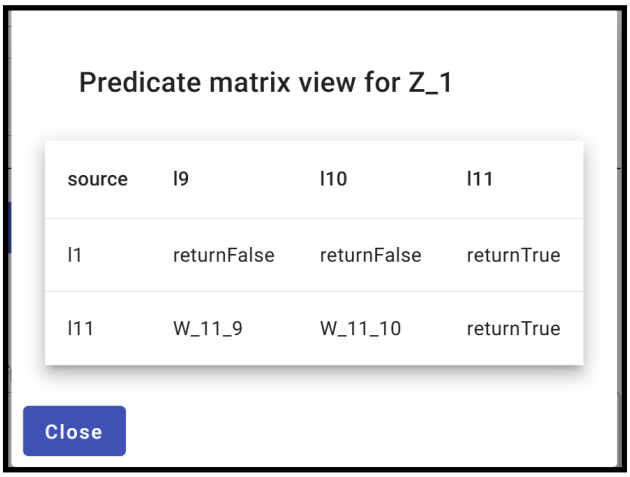

Figure 74 *– Indexed matrix with predicates for transition Z_1 (CDU-2).*

In this concrete example, the predicates reflect conditions of the type "there is a request for LPG product from *CDU-2*" and "there is a request for fuel gas product from *CDU-2*". In other words, transition Z_1 allows tokens to continue only when the corresponding requests/conditions described in the model are present. Analogously, indexed matrices are introduced for the remaining transitions, corresponding to the predicates described for the concrete process units and streams.

The initial characteristics of the tokens are set according to the scenario from the article. In **Figure 75** a token is shown in the input place l_1, which enters Z_1 (*CDU-2*), together with its initial characteristic. This characteristic contains the necessary attributes for the simulation (e.g. type of request and quantities), so that during execution it can be clearly traced what the token "carries" and how it changes along the chain.

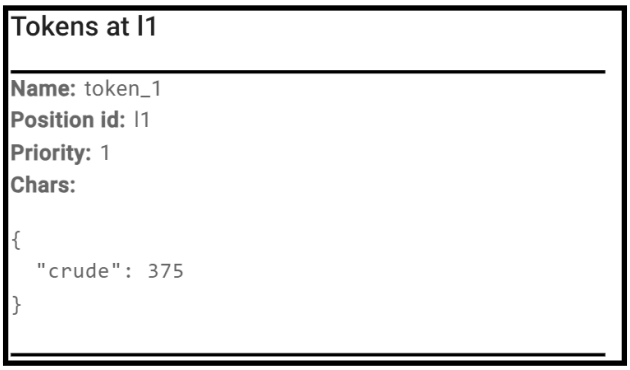

Figure 75 *– Token in the input place l_1 with initial characteristic (example of input to Z_1).*

After successful execution of the simulation, the observed output characteristics coincide with the expected theoretical results. As a concrete illustration, at the output place l_{53} (designated as *"LPG TANKS"*) a token is obtained whose characteristic contains the amount of *LPG* predicted by the model. This is shown in **Figure 76**.

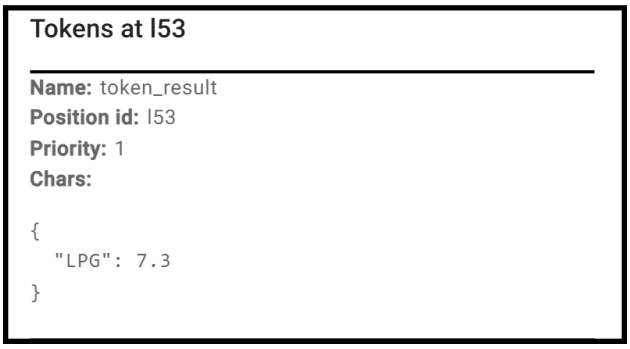

Figure 76 *– Token in the output place l_{53} ("LPG TANKS") with characteristic after simulation.*

In summary, the simulation results confirm that the specified predicates and initial characteristics work correctly and that the GN model adequately describes the production process under consideration. This gives grounds for accepting the model as validated with respect to the theoretical scenario and as usable for subsequent experiments and analyses.

4.3.6. Conclusions on the Model and the Simulation

The production of gaseous products in a petroleum refinery (fuel gas, LPG, and propylene, including as raw material for polypropylene) is a complex parallel process in which correct reflection of the cause–effect relations is key; in GNs this is specified naturally through the predicates of the transition conditions. The present chapter presents a GN model and its software realization and simulation through *OnlineGN*. Obtaining results corresponding to the expected behavior of the process confirms the correctness of the model and demonstrates the applicability of GNs in modeling and simulation of similar production chains.

4.4. Generalized-Net Model of the Production of Heavy Petroleum Products in a Petroleum Refinery

Here, a GN model of the production of heavy petroleum products in a petroleum refinery and its simulation through *OnlineGN* are considered. The construction of the model is described in detail in [89].

4.4.1. Introduction to Generalized-Net Models for Petroleum Products in a Petroleum Refinery

The dissertation considers that the production processes running in parallel in a petroleum refinery can be modeled and simulated through GNs [8]. In the present chapter, the focus is on the processing of heavy residual fractions and the production of heavy petroleum products. Heavy oil represents the residual petroleum fraction after atmospheric distillation [91], with boiling temperature above 360 °C and relative density above 0.933 (*API* < 20) [92]. The aim is for the process to be modeled with the help of GNs and for the model to be simulated through *OnlineGN*.

4.4.2. Processing Scheme for Production of Different Types of Heavy Fuel Oil and Road Pavement Bitumen in a Petroleum Refinery, Modeled through Generalized Nets

The dissertation considers the production of three types of fuel oil with different sulfur content (*Fuel oil* 0.5% *S*, *Fuel oil* 1.0% *S*, *Fuel oil* 2.5% *S*) and the production of road bitumen in *BU*, as shown in **Figure 77**. The specifications and requirements for the fuel oils are presented in [89]. The vacuum fractions (*VR*, *SRVGO*) are obtained in *VDU-1*/*VDU-2* from the atmospheric residue (*CDU*) [94].

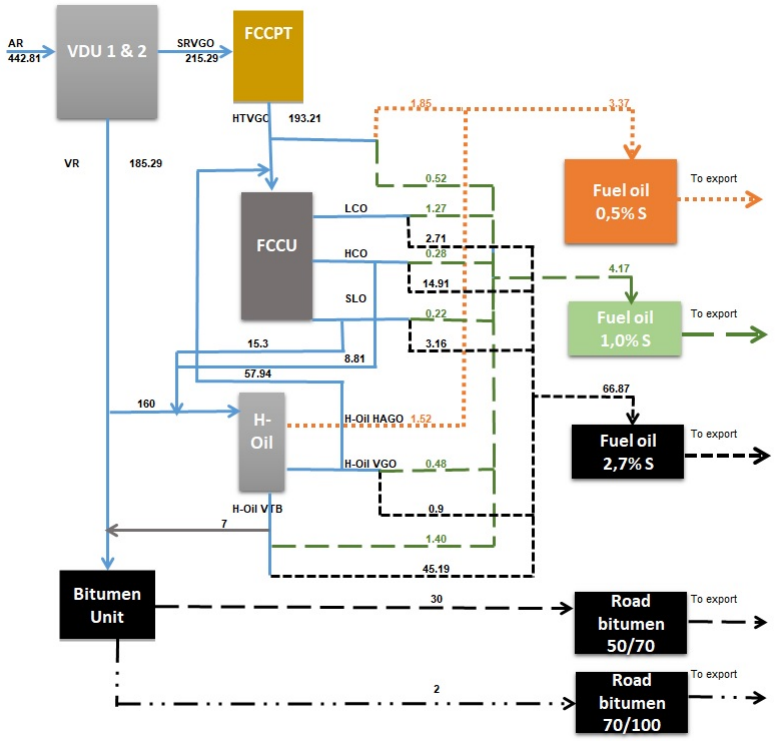

***Figure 77** – Processing scheme for production of different grades of heavy fuel oil and road pavement bitumen in a petroleum refinery to be modeled using generalized nets. The numbers in the diagram are related to the quantity of the heavy oil streams in the dimension of t/h. Different colors are used to differentiate the three grades of produced fuel oils: orange for fuel oil 0.5% S, green for fuel oil 1.0% S, and black for fuel oil 2.5% S.*

4.4.3. Results from Modeling the Production of Heavy Petroleum Products in a Petroleum Refinery with the Help of Generalized Nets

The GN model (see **Figure 78** in the dissertation) contains 8 transitions, 35 places, and 8 token types. The technological units *VDU*, *FCCPT*, *FCCU*, *H-Oil*, and *BU*, as well as the outputs for 0.5% *S*, 1.0% *S*, and 2.5% *S*, are modeled. At the initial moment, tokens $\alpha_0, \beta_0, \gamma_0, \delta_0, \epsilon_0, \zeta_0, \eta_0, \theta_0$ are placed respectively in places $l_1, l_9, l_{17}, l_{26}, l_{29}, l_{31}, l_{33}, l_{35}$ with characteristics corresponding to *AR*, *SRVGO*, *SRVR*, *FCC* feedstock, bitumen feedstock, and final products. The logic of the process is specified through indexed matrices and predicates (e.g. $W_{5,2}, W_{5,3}, W_{5,4}$ for requests to *VDU*), where during execution the tokens are merged and split depending on the truth values of the predicates and the specified quantities $q_i \in [0, Q_i]$.

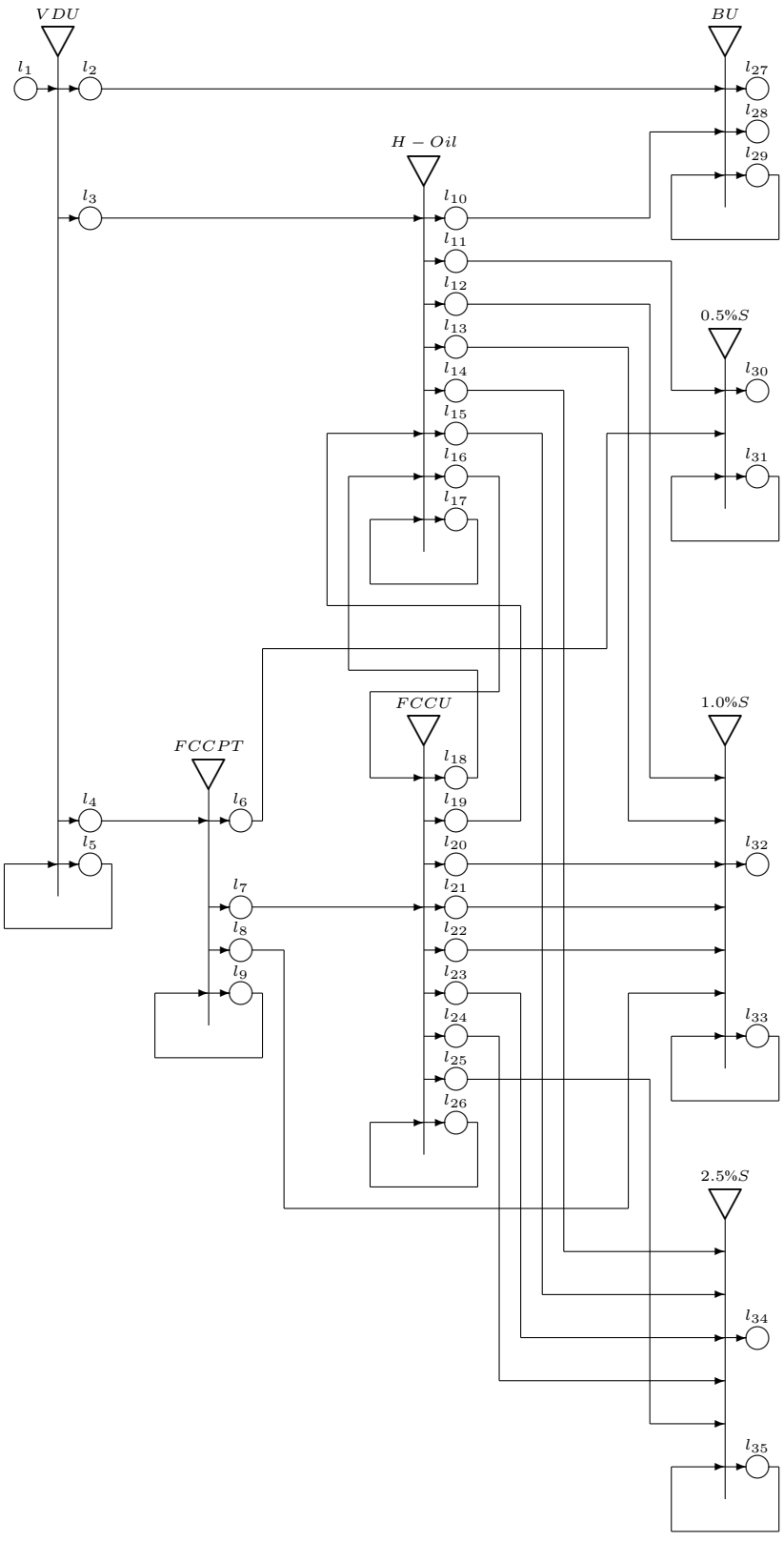

Figure 78 *– A GN model of the manufacturing of heavy oil products in "LUKOIL Neftohim Burgas" refinery.*

4.4.4. Simulation of the Model with *OnlineGN*

The dissertation considers implementation and execution of the GN model in *OnlineGN* for the purpose of verification and derivation of quantitative results. **Figure 79** shows the GN scheme integrated in the simulator. For the considered scenario, the initial token in place l_1 is atmospheric residue (*AR*) with characteristic value 442.81, and the requests to the subsequent units are specified through the characteristic functions (*code*) and the indexed matrices of the transitions. The results are derived from the characteristics of the tokens in the output places (e.g. l_{32} for 1.0 mass.% *S* and l_{34} for 2.5 mass.% *S*).

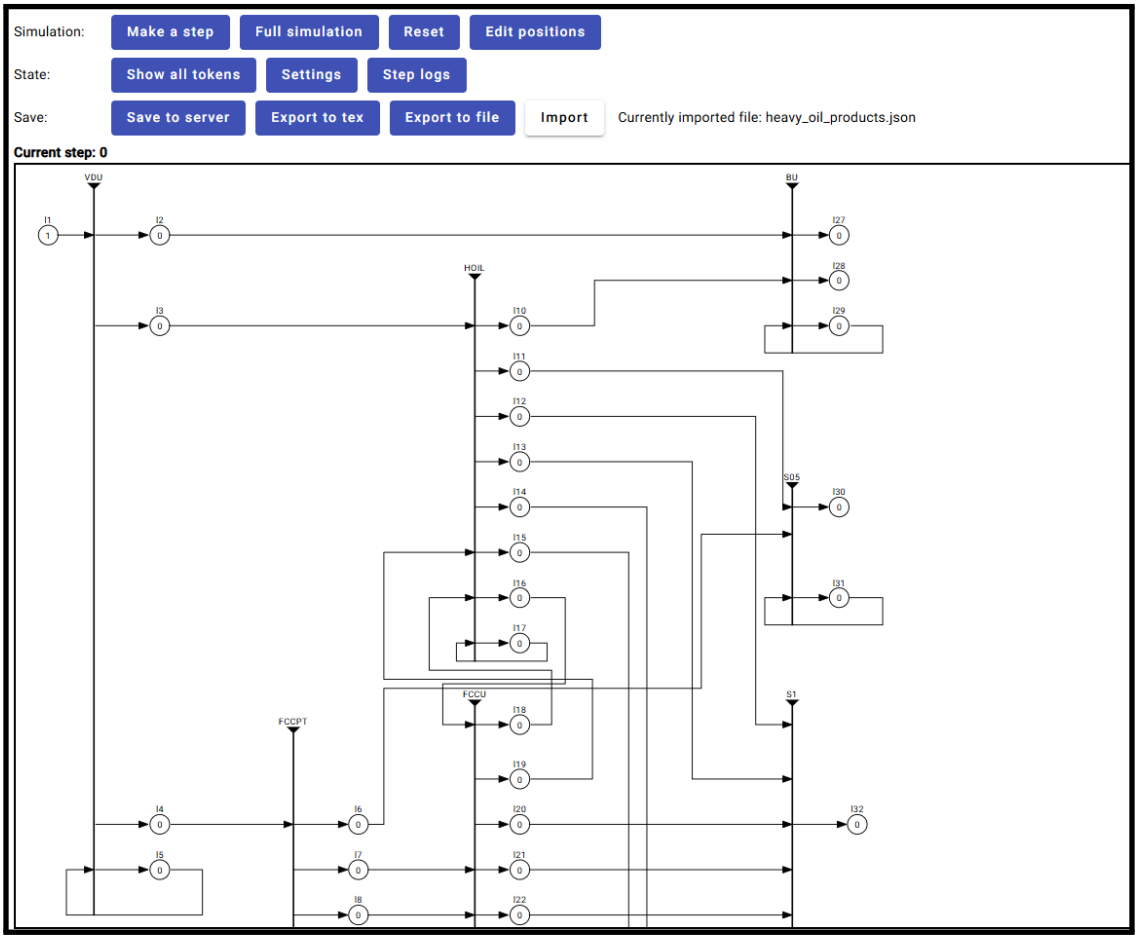

Figure 79 *– Generalized net (GN) model of the heavy oil products processing chain in the refinery.*

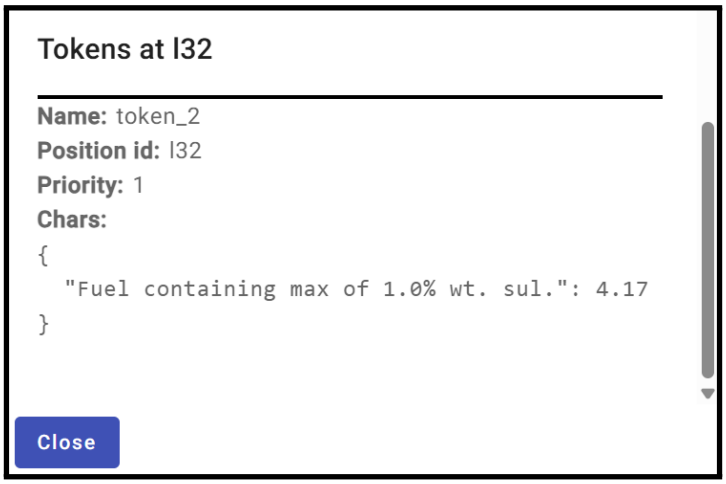

Figure 84 *– Obtained quantity of fuel oil with a maximum of 1.0 wt.% sulfur (place l_{32}).*

4.4.5. Results for the Model

As can be seen in **Figure 77**, the production of the three classes of heavy fuel oil and of the classes of road pavement bitumen is a complex parallel process involving *VDU*, *FCCPT*, *FCCU*, *H-Oil*, and *BU*, within which refined heavy petroleum products and final product streams are obtained.

4.4.6. Conclusions

The obtained results show that the processes of production of different classes of heavy petroleum products in a refinery can be successfully modeled through GNs, while the parallel nature and the cause–effect dependencies are represented naturally. The simulation carried out through ***OnlineGN*** leads to results corresponding to the expected behavior of the system, which serves as verification of the correctness of the model and confirms the applicability of GNs in modeling and simulation of production processes in a petroleum refinery.

Conclusion

In the present dissertation, the main goal set was achieved – the development of a new simulator for GNs functioning in an online environment, which substantially facilitates and accelerates work with GNs. The implemented solution supports key activities such as model creation, subsequent editing (including changes of functions and visualization), and efficient sharing of nets. During the development, modern and widely established technologies were used, aligned with the requirements for reliability, extensibility, and usability in a web environment.

Alongside the software result, the dissertation also produced new contributions related to the theory and application of GNs. New algorithms were developed, aimed at supporting the modeling and presentation of GNs. First, an algorithm was proposed for automatic drawing of a GN from its description, which shortens the time needed to obtain a correct and clear visualization and limits the need for manual corrections. Second, an algorithm was developed for exporting the net in *TeX* format, so that after edits and visual adjustments in the simulator, the model can be directly included in scientific publications and other academic materials.

In addition, new GN models were created in different application areas, which demonstrate the expressive power of the apparatus and validate the capabilities of the developed simulator. Among them are models related to the representation of finite automata, a model of a mathematical game, as well as models of industrial processes, including scenarios related to the production of gas and petroleum products. All proposed models were implemented and studied through simulations with the simulator developed in the dissertation, which provides practical verification of the functionality and applicability of the system.

Author's Reference

Reference of the Contributions in the Dissertation

Scientific Contributions

The following scientific contributions have been achieved in the present dissertation:

1. **Algorithms for automation and visualization of Generalized Nets (GNs):**
   - An algorithm for automated drawing of GNs on the basis of their abstract description has been developed, eliminating the need for manual specification of coordinates.
   - An algorithm has been created for transforming vector images (*SVG*) of GNs into *TeX* format through identification of graphical primitives and coordinate transformation.
2. **Parallel solution of an exponential problem (Towers of Hanoi):**
   - A new parallel variant of the Towers of Hanoi problem has been formalized, allowing simultaneous movement of up to k rings.
   - A theorem has been proved for the minimum number of parallel steps, $H(n,k) = 2^{\lceil \frac{n}{k} \rceil} - 1$, and a constructive algorithm has been implemented for solving the problem.
3. **Generalized-net modeling and simulation of the parallel exponential Towers of Hanoi problem:**
   - An original GN model of the parallel Towers of Hanoi has been created, using ρ-tokens for the objects and χ-tokens for the resources.
   - The applicability of the formalism has been demonstrated through simulations in the *OnlineGN* environment, showing effective management of concurrent processes and logical constraints.

Scientific–Applied Contributions

The following results have been achieved from a scientific–applied point of view:

1. **Design, implementation, and application of the software system *OnlineGN*:**
   - An innovative web-based platform has been developed for modeling, simulation, and visualization of GNs, offering a centralized environment without the need for local installation.
   - A convenient and interactive interface has been developed for changing each GN component in the simulator, as well as for tracking details of the execution of the net.
   - Functionality has been implemented for export from *SVG* to *TeX*, which optimizes the preparation of graphical materials for academic publications.

2. **A new method for representation of finite automata through GNs:**

    - A methodology has been developed for describing deterministic and nondeterministic finite automata through GNs, allowing the capture of parallel computation paths.

3. **Modeling of existing processes in a petroleum refinery through GNs:**

    - GN models have been developed for real technological processes in a petroleum refinery, including the production of gas, polymeric, and heavy petroleum products.

4. **Verification of the new GN models through *OnlineGN*:**

    - All newly created models in the dissertation have been verified through simulations in *OnlineGN*, which simultaneously verifies both the correctness of the models and the capabilities of the system.

Publications Related to the Dissertation

1. Dimitriev, A.; Terziev, G. *Automating Creativity: Enhancing Generalized Nets with Algorithmic Drawing*. In: *Proceedings of the 22nd International Workshop on Intuitionistic Fuzzy Sets and Generalized Nets*, Warsaw, Poland, October 18, 2024 (in press).

2. Dimitriev, A. *Representing Finite-State Automata with Generalized Nets: Simulation via OnlineGN*. *Proceedings of the Jangjeon Mathematical Society*, vol. 29, no. 1, pp. 156–167, Jan. 2026. doi: https://doi.org/10.17777/pjms.2026.29.1.011.

3. Dimitriev, A.; Atanassov, K.; Angelova, N. *Parallel Towers of Hanoi via Generalized Nets: Simulated with OnlineGN*. *Software*, vol. 4, no. 4, art. 23, 2025. doi: https://doi.org/10.3390/software4040023.

4. Stratiev, D. D.; Dimitriev, A.; Stratiev, D.; Atanassov, K. *Modeling the Production Process of Fuel Gas, LPG, Propylene, and Polypropylene in a Petroleum Refinery Using Generalized Nets*. *Mathematics*, 2023, **11**, 3800. doi: https://doi.org/10.3390/math11173800.

5. Stratiev, D. D.; Dimitriev, A.; Stratiev, D.; Atanassov, K. *Generalized Net Model of Heavy Oil Products' Manufacturing in Petroleum Refinery*. *Mathematics*, 2023, **11**, 4753. doi: https://doi.org/10.3390/math11234753.

Bibliography

[1] Atanassov, K. *Theory of generalized nets (an algebraic aspect)*. AMSE Review, 1984, vol. 1, no. 2, pp. 27–33.

[2] Petri, C.-A. *Kommunication mit Automaten*. PhD thesis, University of Bonn, Bonn, Germany, 1962. (Also published as: Schriften des Instituts für Instrumentelle Mathematik, no. 2, Bonn, 1962.)

[7] Atanassov, K. *Introduction to the Theory of Generalized Nets*. Burgas: Pontica Print, 1992, pp. 7–10. (In Bulgarian)

[8] Atanassov, K. *Generalized Nets*. Singapore: World Scientific, 1991, pp. 9–14.

[9] Atanassov, K. *On Generalized Nets Theory*. Sofia: "Prof. Marin Drinov" Publishing House, 2007, pp. 11–14.

[10] Atanassov, K.; Sotirova, E. *Generalized Nets*. Sofia: Prof. Marin Drinov Publishing House of the Bulgarian Academy of Sciences, 2017. 172 pp. ISBN 978-954-322-881-2. (In Bulgarian)

[11] Orozova, D. *Generalized Net Models of Intelligent Learning Environments*. Sofia: Prof. Marin Drinov Publishing House of the Bulgarian Academy of Sciences, 2011. 234 pp. ISBN 978-954-322-481-4. (In Bulgarian)

[12] Georgiev, P. R. *Generalized Net Models of Knowledge-Based Systems*. Dissertation for the award of the educational and scientific degree "Doctor", Specialized Scientific Council on Computer Science and Computer Engineering, Sofia, 1998. Scientific specialty: 02.21.05 "Artificial Intelligence Systems". Scientific supervisor: K. Atanassov. (In Bulgarian)

[13] Stefanova-Pavlova, M. M. *Modeling of Flexible Automated Manufacturing Systems by Generalized Nets*. Dissertation for the award of the educational and scientific degree "Doctor", Specialized Scientific Council on Computer Science and Computer Engineering, Sofia, 2001. Scientific specialty: 01.01.12 "Informatics". Scientific supervisor: K. Atanassov. (In Bulgarian)

[14] Aladjov, H. Ts. *Generalized Net Models of Learning and Self-Organization and Their Application to the Control of Motor Activity*. Dissertation for the award of the educational and scientific degree "Doctor", Specialized Scientific Council on Computer Science and Computer Engineering, Sofia, 2002. Scientific specialty: 02.21.05 "Artificial Intelligence Systems". Scientific supervisor: K. Atanassov. (In Bulgarian)

[17] Starke, P. *Petri-Netze*. Berlin: VEB Deutscher Verlag der Wissenschaften, 1980, pp. [1–13].

[31] Kamada, T.; Kawai, S. *An algorithm for drawing general undirected graphs*. Information Processing Letters, 1989, vol. 31, no. 1, pp. 7–15.

[32] Ellson, J.; Gansner, E. R.; Koutsofios, E.; North, S. C.; Woodhull, G. *Graphviz and Dynagraph – Static and Dynamic Graph Drawing Tools*. In: Jünger, M.; Mutzel, P. (eds.), *Graph Drawing Software*, Springer, 2004, pp. 127–148.

[39] Dimitriev, A.; Terziev, G. *Automating Creativity: Enhancing Generalized Nets with Algorithmic Drawing*. In: *Proceedings of the 22nd International Workshop on Intuitionistic Fuzzy Sets and Generalized Nets*, Warsaw, Poland, 18 Oct 2024 (in press).

[41] Angel Dimitriev. *phd-thesis-onlinegn*. Angel Dimitriev's GitHub repository, 2026. Available online: `https://github.com/Angeld55/phd-thesis-onlinegn` (accessed 12 Mar 2026).

[42] *Introduction to OnlineGN – A Simulator for Generalized Nets*. Video demonstration, published 12 Feb 2026. Available online: `https://youtu.be/sM_Cp-pt2TU?si=kl58ruSYlCRxdJir` (accessed 12 Feb 2026). (In Bulgarian)

[43] Nielsen, J. *Response Times: The 3 Important Limits*. Nielsen Norman Group, online article, 1993. Available online: `https://www.nngroup.com/articles/response-times-3-important-limits/` (accessed 24 Dec 2025).

[51] Render Team. *Web Services*. Official documentation. Available online: `https://render.com/docs/web-services` (accessed 12 Feb 2026).

[52] MongoDB Team. *What is MongoDB Atlas?* Official documentation. Available online: `https://www.mongodb.com/docs/atlas/` (accessed 12 Feb 2026).

[53] Angular Team. *What is Angular?* Official documentation. Available online: `https://angular.dev/overview` (accessed 12 Feb 2026).

[54] TypeScript Team. *The TypeScript Handbook*. Official documentation. Available online: `https://www.typescriptlang.org/docs/handbook/intro.html` (accessed 12 Feb 2026).

[55] D3 Contributors. *d3-transition: Animated transitions for D3 selections*. npm package documentation. Available online: `https://www.npmjs.com/package/d3-transition` (accessed 12 Feb 2026).

[56] Dimitriev, A. *Representing finite-state automata with generalized nets: Simulation via OnlineGN*. Proceedings of the Jangjeon Mathematical Society, 2026, vol. 29, no. 1, pp. 156–167. doi: 10.17777/pjms.2026.29.1.011.

[57] Atanassov, K. T. *Generalized Nets and Finite Automata*. AMSE Review, Series "Modelling and Simulation", 1985, vol. 2, no. 2, pp. 1–7.

[58] Hopcroft, J. E.; Motwani, R.; Ullman, J. D. *Introduction to Automata Theory, Languages, and Computation*. 3rd ed., Addison–Wesley, 2006, pp. [45–47].

[59] Kozen, D. C. *Automata and Computability*. Springer, 1997, pp. [7–9].

[60] Papadimitriou, C. H. *Computational Complexity*. Addison–Wesley, 1994, pp. [13–14].

[61] Rabin, M. O.; Scott, D. *Finite Automata and Their Decision Problems*. IBM Journal of Research and Development, 1959, vol. 3, no. 2, pp. 114–125.

[63] Wikipedia contributors. *Tower of Hanoi*. Wikipedia, The Free Encyclopedia, 2025. Available online: `https://en.wikipedia.org/w/index.php?title=Tower_of_Hanoi&oldid=1270762908` (accessed 12 Feb 2026).

[64] Frame, J. S. *Solution of the Tower of Hanoi Problem*. American Mathematical Monthly, 1945, vol. 52, no. 10, pp. 596–604.

[65] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. *Introduction to Algorithms*. 3rd ed., Cambridge, MA, USA: MIT Press, 2009, pp. [4–10].

[78] Stratiev, D. D.; Dimitriev, A.; Stratiev, D.; Atanassov, K. *Modeling the Production Process of Fuel Gas, LPG, Propylene, and Polypropylene in a Petroleum Refinery Using Generalized Nets*. Mathematics, 2023, vol. 11, art. 3800.

[79] Zhou, J.; Reniers, G.; Zhang, L. *A weighted fuzzy Petri-net based approach for security risk assessment in the chemical industry*. Chemical Engineering Science, 2017, vol. 174, pp. 136–145.

[80] Infosys Limited. *Predictive Analytics and Dynamic Optimization: The Sweet Spot in Refinery Planning*. White paper, 2018. Available online: `https://www.infosys.com/industries/oil-and-gas/insights/documents/sweet-spot-refinery-planning.pdf` (accessed 14 Dec 2022).

[81] Zhang, L.; Yuan, Z.; Chen, B. *Refinery-wide planning operations under uncertainty via robust optimization approach coupled with global optimization*. Computers & Chemical Engineering, 2021, vol. 146, art. 107205. doi: 10.1016/j.compchemeng.2020.107205.

[82] Stratiev, D. D.; Zoteva, D.; Stratiev, D. S.; Atanassov, K. *Modelling the Process of Production of Automotive Gasoline by the Use of Generalized Nets*. In: *Uncertainty and Imprecision in Decision Making and Decision Support: New Advances, Challenges, and Perspectives*, Lecture Notes in Networks and Systems, vol. 338, Springer, 2022, pp. 349–365. doi: 10.1007/978-3-030-95929-6_27.

[83] Stratiev, D. D.; Stratiev, D.; Atanassov, K. *Modelling the Process of Production of Diesel Fuels by the Use of Generalized Nets*. Mathematics, 2021, vol. 9, art. 2351. doi: 10.3390/math9192351.

[89] Stratiev, D. D.; Dimitriev, A.; Stratiev, D.; Atanassov, K. *Generalized Net Model of Heavy Oil Products' Manufacturing in Petroleum Refinery*. Mathematics, 2023, vol. 11, art. 4753.

[91] Kaiser, M. J.; De Klerk, A.; Gary, J. H.; Handwerk, G. E. *Petroleum Refining: Technology, Economics, and Markets*. 6th ed., Boca Raton, FL, USA: CRC Press, 2020, pp. 1–722.

[92] Merdrignac, I.; Espinat, D. *Physicochemical characterization of petroleum fractions: The state of the art*. Oil & Gas Science and Technology – Rev. IFP, 2007, vol. 62, pp. 8–29.

[94] Gaikwad, R. W.; Warade, A. R.; Bhagat, S. L.; Bhasarkar, J. B. *Optimization and simulation of refinery vacuum column with an overhead condenser*. Materials Today: Proceedings, 2022, vol. 57, pp. 1593–1597. doi: 10.1016/j.matpr.2021.12.175.