Automating Creativity: Enhancing Generalized Nets with Algorithmic Drawing

Angel Ivanov Dimitriev[1,2] and Georgi Ilkov Terziev[2]

[1] Dept. of Bioinformatics and Mathematical Modelling, Institute of Biophysics and Biomedical Engineering, Bulgarian Academy of Sciences, Acad. G. Bonchev Str. Bl. 105, 1113 Sofia, Bulgaria
[2] Faculty of Mathematics and Informatics, Sofia University "St. Kliment Ohridski", 5 James Bourchier Blvd., Sofia 1164, Bulgaria
adimitriev@fmi.uni-sofia.bg georgiterziev2002@abv.bg

Abstract. New, general approach to the feeble modal topological structure is introduced and these structures are illustrated with five groups of examples using intuitionistic fuzzy objects.

Keywords: Generalized nets · Algorithmic drawing.

1 Introduction

Generalized nets (GNs) are a powerful tool used in the modeling and analysis of complex systems [1,2]. They extend the concept of Petri nets, providing a more flexible and expressive framework for representing dynamic processes. However, a significant challenge faced by researchers and practitioners is the manual effort required to draw these nets. Traditionally, all generalized nets presented in academic articles and books are painstakingly drawn by manually specifying the coordinates. This method, while precise, is time-consuming and prone to human error.

The same issue persists in GN simulators, where users must input the exact coordinates to visualize the nets. This manual approach not only slows down the modeling process but also poses a barrier to efficient experimentation and rapid prototyping.

In response to this challenge, we propose a novel algorithm designed to automate the drawing of generalized nets. Our algorithm significantly accelerates the process by automatically generating clear and understandable visual representations of GNs from their abstract descriptions. By eliminating the need for manual coordinate specification, our approach simplifies the creation and manipulation of generalized nets, enabling users to focus more on analysis and less on the tedious aspects of diagram creation.

In this article, we will delve into the details of our algorithm, demonstrate its application, and highlight its advantages over traditional methods. Through this, we aim to pave the way for more efficient and error-free GN modeling, ultimately enhancing the utility and accessibility of generalized nets in various fields.

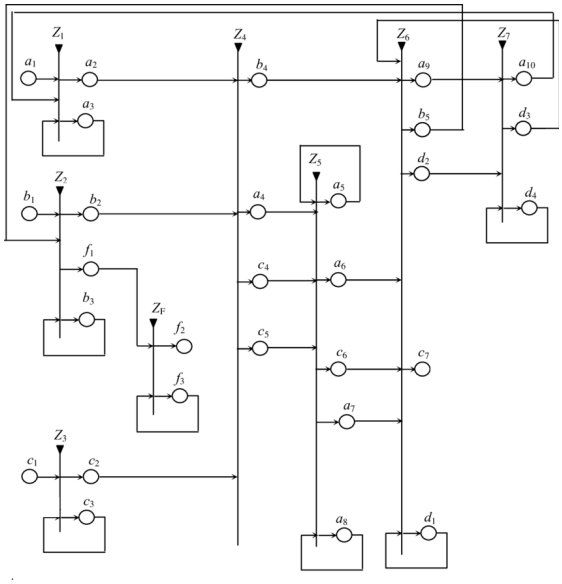

Fig. 1. Example of a generalized net drawing.

2 Current Practices in GN Visualization

Visualizing generalized nets (GNs) involves the precise placement of various components—places, transitions, and arcs—in a manner that is both clear and accurate. Traditionally, this visualization process is done manually, requiring users to specify exact coordinates for each element. This section outlines the common practices and rules followed when manually drawing GNs.

2.1 Directionality of Arrows

In most generalized nets, the arrows indicating the flow between transitions and places are oriented from left to right. This left-to-right directionality helps maintain a clear and intuitive understanding of the net's flow and logic. The layout adheres to a convention where each transition's outputs are depicted on its right side, reinforcing a straightforward reading direction. Backward transitions, where necessary, are incorporated thoughtfully to avoid confusion but are used sparingly.

2.2 Positioning of Output Places

A key aspect of GN visualization is the precise placement of output places. For clarity and consistency, all output places are positioned immediately to the right of their corresponding transitions. This alignment ensures that the flow of the net is easily traceable, with each transition leading directly to its outputs without ambiguity.

2.3 Dual-Role Places (Input and Output)

Some places in a generalized net serve dual roles, acting as both inputs to and outputs from transitions. When a place fulfills this dual function, it is conventionally placed at the bottom of the transition. This practice helps in maintaining a clean layout by minimizing the crossing of arcs and avoiding visual clutter. By positioning dual-role places at the bottom, the diagram remains organized, and the relationships between different components are clearly delineated.

2.4 Additional Practices

While the above rules form the core guidelines for GN visualization, several additional practices are commonly followed to enhance readability and coherence:

- **Uniform Spacing:** Consistent spacing between elements is maintained to ensure that the diagram is visually balanced and easy to interpret.
- **Labeling:** Clear and concise labeling of places and transitions is essential. Labels are typically positioned close to the corresponding elements without overlapping other parts of the net.
- **Color Coding:** When necessary, color coding is used to distinguish different types of transitions or places, adding another layer of clarity to the diagram.
- **Minimizing Crossings:** Efforts are made to minimize the crossing of arcs, as intersecting lines can lead to confusion. This is achieved through careful planning of element placement and flow direction.
- **Consistent Orientation:** The overall orientation of the net is kept consistent throughout the diagram. This means maintaining a left-to-right flow and avoiding sudden changes in direction unless absolutely necessary.

These practices, while effective in creating clear and understandable GN diagrams, are labor-intensive and prone to errors. The manual specification of coordinates for each element can be time-consuming and challenging, particularly for complex nets. This underscores the need for automated solutions that can streamline the visualization process, ensuring accuracy and efficiency.

In the subsequent sections, we will introduce our algorithm designed to automate the drawing of generalized nets. This algorithm adheres to the established visualization practices while significantly reducing the manual effort required, offering a more efficient and error-free approach to GN visualization.

3 Graph Drawing

Graph drawing is a fundamental aspect of visualizing and understanding complex networks and relationships between entities. It is a field that merges mathematics, computer science, and visualization techniques to represent graphs in a two-dimensional or three-dimensional space [3]. Effective graph drawing is crucial for analyzing structures such as social networks, biological networks, and data flow diagrams, among others.

The main goal of graph drawing is to place nodes and edges in such a way that the structure of the graph is easily comprehensible, minimizing edge crossings, and maintaining a clear representation of relationships. Various algorithms exist to achieve optimal layouts, such as force-directed layouts, hierarchical layouts, and circular layouts. The choice of algorithm often depends on the nature of the graph and the specific application for which the drawing is intended.

3.1 Graph Drawing with GraphViz

GraphViz is a powerful open-source tool designed specifically for graph visualization [4]. It provides a suite of layout engines that cater to different types of graphs, making it versatile for various graph drawing needs. With GraphViz, users can easily define graphs in a text-based format using the DOT language, which is then processed to generate visual representations.

The primary layout algorithms available in GraphViz include:

- **dot**: A hierarchical layout algorithm ideal for directed graphs, which organizes nodes in layers.
- **neato**: A force-directed layout algorithm that places nodes based on a spring model, making it suitable for undirected graphs.
- **fdp**: Similar to `neato`, but uses a different approach to minimize edge crossings.
- **twopi**: A radial layout algorithm that arranges nodes in concentric circles, often used for showing radial hierarchies.
- **circo**: A circular layout algorithm, particularly useful for cyclic structures.

Using GraphViz, one can generate high-quality graph drawings for a wide range of applications. It supports output in various formats, including PNG, PDF, and SVG, making it convenient for integration into publications and presentations. Moreover, the flexibility of GraphViz allows for customization of node shapes, colors, edge styles, and other visual aspects, enabling the creation of aesthetically pleasing and informative graphs.

4 Modeling a Generalized Net as an Oriented Graph

Generalized nets (GNs) are a formalism used to model complex systems where processes are interconnected through various states and transitions. To visualize and analyze the structure of a generalized net, it can be effectively modeled as an oriented graph. This representation provides clarity in understanding the relationships between the different components of the net.

In the context of graph theory:

- **Places** in the generalized net are represented as **nodes (vertices)**. Each place corresponds to a specific state or condition in the system.
- **Transitions** in the generalized net are also represented as **nodes (vertices)**. These nodes represent the processes or actions that cause movement from one state to another.

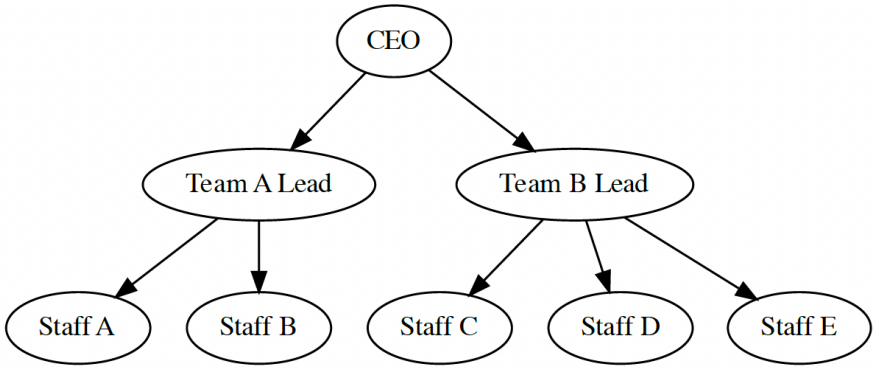

Fig. 2. An example of a graph generated using GraphViz's `dot` layout engine.

– **Edges (arcs)** are directed connections between nodes, representing the flow or movement within the net. An edge is drawn from a place node to a transition node if the place is an input for the transition. Similarly, an edge is drawn from a transition node to a place node if the place is an output of the transition.

This oriented graph model allows us to visualize the flow of tokens (representing entities or information) through the generalized net, highlighting how they move from place to place via transitions. The orientation of the edges ensures that the directionality of the process is maintained, reflecting the inherent logic of the generalized net.

4.1 Example

Consider a generalized net with five places and three transitions:

– **Places**: l_1, l_2, l_3, l_4, l_5
– **Transitions**: Z_1, Z_2, Z_3

The connections in the generalized net are as follows:

– $l_1 \to Z_1$
– $Z_1 \to l_2, l_3$
– $l_2 \to Z_2$
– $Z_2 \to l_4$
– $l_3, l_4 \to Z_3$
– $Z_3 \to l_5$

In this representation, both places and transitions are modeled as **nodes** in the graph. Specifically:

- Nodes corresponding to places (l_1, l_2, l_3, l_4, l_5) represent the different states or conditions in the generalized net.
- Nodes corresponding to transitions (Z_1, Z_2, Z_3) represent the actions or processes that lead to state changes.

Edges are directed connections between these nodes, illustrating the flow or movement within the net:

- There is a directed edge from node l_1 to node Z_1, indicating that l_1 is an input to the transition Z_1.
- From Z_1, there are directed edges to both l_2 and l_3, representing the possible outputs of the transition.
- Similarly, l_2 has a directed edge leading to Z_2, which then directs to l_4.
- Finally, there are edges from both l_3 and l_4 to Z_3, with Z_3 having an edge leading to l_5.

The following figures illustrate the generalized net and its corresponding oriented graph representation:

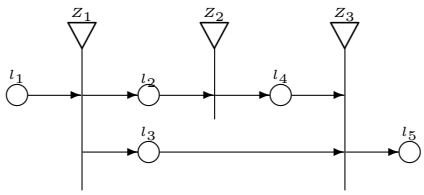

Fig. 3. The generalized net with places l_1, l_2, l_3, l_4, l_5 and transitions Z_1, Z_2, Z_3.

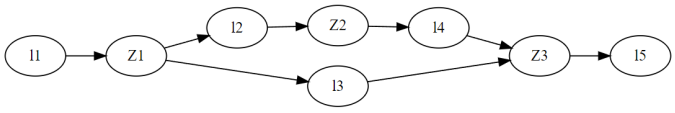

Fig. 4. The oriented graph representation of the generalized net. Nodes represent places and transitions, while directed edges indicate the flow between them.

This example illustrates how a generalized net can be effectively represented as an oriented graph, with places and transitions as nodes and directed edges

representing the flow between them. This approach makes the relationships between the components of the generalized net clear, facilitating the analysis and understanding of the system's behavior.

5 Enhancing Generalized Net Visualization with GraphViz

To effectively visualize a generalized net using GraphViz, it's important to set specific constraints and parameters that ensure the net is both aesthetically pleasing and accurately represents the intended structure. Below, we outline key steps and considerations to achieve this:

5.1 Setting Global Parameters

At the beginning of the GraphViz file, we define some global settings to establish the overall layout and style of the graph:

- `rankdir=LR`: This directive ensures that the graph is drawn with a left-to-right orientation, aligning with the common representation of generalized nets where transitions are depicted horizontally with input places on the left and output places on the right.
- `splines=ortho`: This setting forces all edges to be orthogonal, meaning they are parallel to the X and Y axes. This is crucial for generalized nets, as it preserves the clean, grid-like appearance where connections run straight and perpendicular, simplifying the visual structure.

5.2 Defining Transition Nodes

Transition nodes in a generalized net are unique in that they often represent actions or processes and are usually depicted as rectangles or thick lines. To achieve this in GraphViz:

- `shape=rect`: This sets the shape of the transition nodes to rectangles, making them visually distinct from the circular or elliptical nodes representing places.
- `height={count of output places}, width=0.05`: The height of each rectangle is dynamically set based on the number of output places it connects to, while the width is kept narrow (0.05) to resemble a thick line. This approach emphasizes the transition as a process leading to multiple potential outcomes.

5.3 Grouping Output Places with Subgraphs

In GraphViz, output places are typically drawn adjacent to their corresponding transition. To ensure that these places are kept together and maintain their relative positions:

– We define them within a `subgraph cluster`. By grouping the transition node and its associated output places in a subgraph, we create a logical grouping that GraphViz respects during layout processing. This reduces the likelihood of these nodes being rearranged or separated from each other during the graph drawing, thereby maintaining the visual coherence of the generalized net.
– `rank=same`: To ensure that all output nodes for a transition are aligned on the same Y coordinate, we use the `rank=same` constraint within the subgraph cluster. This ensures that the output places appear horizontally aligned, making the graph cleaner and easier to interpret.

5.4 Handling Input Edges with Invisible Nodes

One of the challenges in GraphViz is ensuring that all edges enter the right side of transition nodes consistently. To address this, we introduce the concept of an invisible transition node:

– For each input edge to a transition node, an `invisible_node` is created and positioned to the left of the transition node.
– Each input place is connected to its corresponding `invisible_node`, rather than directly to the transition node.
– The `invisible_node` is then connected to the transition node with a visible arrow, ensuring that the arrow originates from the right of the transition.
– Both the `invisible_node` and the transition node are placed in a subgraph cluster to maintain their relative positioning, preventing rearrangement during layout processing.

This technique allows for a cleaner and more organized layout, where all input connections to a transition appear as if they enter from a single, unified direction on the right. This not only enhances the visual clarity of the graph but also ensures that the logical flow of transitions is easily interpreted. By using invisible nodes and carefully managing their placement, the generalized net is rendered with a clear and consistent structure, making it more intuitive to analyze and understand the system's flow.

6 Algorithm for Generating GraphViz Strings for Generalized Nets

In this section, we present the pseudocode for generating a GraphViz string from a generalized net (GN). This algorithm is designed to automate the visualization of GNs, ensuring that the resulting graph adheres to predefined layout parameters. The core function, `generateGraphVizString`, is supported by several auxiliary functions that handle the specific tasks of mapping transitions and places, generating invisible nodes, and creating the necessary connections between elements in the GN.

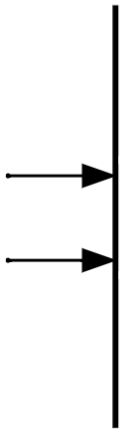

Fig. 5. Trasition visualization with invisible nodes before it.

The pseudocode provided below outlines the essential steps and logic used in the algorithm, offering a clear and concise representation suitable for implementation.

6.1 Generating Invis Nodes

The function `genInvisNodes` creates invisible nodes that are used to align transitions within the graphical layout.

Function 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

6.2 Connecting Invis Nodes to Transitions

The function `genEdgeBetweenInvisNodesAndTransition` creates edges from invisible nodes to the transitions, ensuring they are aligned and correctly connected.

Function genEdgeBetweenInvisNodesAndTransition(result, genNet, transition)

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

6.3 Generating Transitions

The function `genTransitionsString` generates the necessary structure for each transition in the GN, including its invisible nodes and connections.

Function 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
```

6.4 Generating Outgoing Places and Edges

Two separate functions handle the generation of outgoing places and edges. These ensure that the connections between transitions and places are created properly.

Function genOutgoingPlacesFromTransition(genNet, result, transition)

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

Function genOutgoingEdgesFromTransition(genNet, result, transition)

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

6.5 Generating Output Edges from Places

Finally, the function `genOutputEdgesFromPlaces` creates the edges connecting places to invisible nodes, ensuring that the transitions maintain proper alignment.

Function 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
```

6.6 Main GraphViz Generation Function

The main function, `genGraphVizString`, ties all the previous functions together to generate the complete GraphViz representation of the Gener net.

Function genGraphVizString(genNet)

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

7 Postprocessing

In this chapter, we describe the postprocessing techniques applied to the SVG (Scalable Vector Graphics) output generated by the algorithm. Initially, the algorithm produces a graph where transitions are represented as very thin rectangular nodes, and positions are depicted as circular nodes. However, to visually adapt this output into the form of a generalized net (GN), several modifications are made during the postprocessing phase.

The goal of the postprocessing is to modify the SVG so that it better aligns with the generalized net format. The following changes are made:

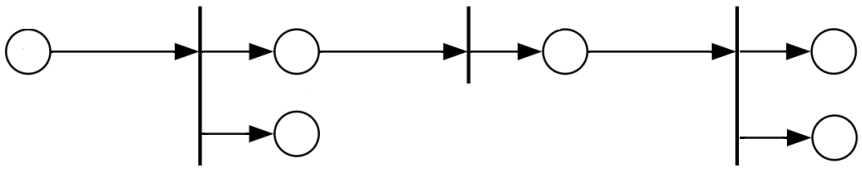

Fig. 6. An example of a rendered graph (generalized net) with Graphviz

7.1 Transitions

Each transition, originally a thin rectangle, is updated by placing a black triangle on top of it. The triangle serves as a marker for the transition, giving it more prominence in the diagram. Furthermore, the name of each transition is placed above the triangle for easy identification.

7.2 Positions (Places)

For positions, represented by circles, the postprocessing algorithm adds the names of the positions above each circle. This is necessary because the graph layout tool (Graphviz) used in the initial SVG generation does not natively support labeling the positions in this way. Therefore, the postprocessing step manually inserts these labels to ensure clarity in the final visual representation.

7.3 Rendering the Generalized Net

After applying the above changes, the resulting SVG closely resembles the structure of a generalized net. The transitions are now clearly marked and labeled, and the positions (places) have appropriate labels above their respective circles. This transformation provides a more intuitive and accurate graphical representation of the system being modeled, making it easier to interpret and analyze.

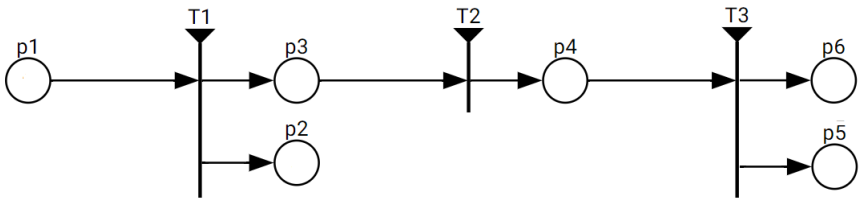

Fig. 7. An example of a rendered generalized net

7.4 Pseudocode of the Postprocessing Step

Function 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
```

7.5 Pseudocode for the Main Drawing Function

Function drawGenNet(genNet)

```
graphVizStr ← genGraphVizString(genNet)
svg ← graphViz.render(graphVizStr)
postprocessSVG(svg)
```

7.6 Conclusion

The postprocessing phase plays a crucial role in adapting the initial SVG output into a more user-friendly and visually intuitive generalized net representation. By adding the triangles and labels, the graph gains clarity and structure that would otherwise be missing.

References

1. Atanassov, Krassimir T. Generalized Nets. Singapore: World Scientific, 1991.
2. Atanassov, Krassimir T. On Generalized Nets Theory. Sofia: "Prof. Marin Drinov" Academic Publishing House, 2007.
3. Kamada, Tomihisa, and Kawai, Satoru. An algorithm for drawing general undirected graphs. Information Processing Letters 31(1): 7–15, 1989.
4. Ellson, John, Gansner, Emden R., Koutsofios, Eleftherios, North, Stephen C., and Woodhull, Gordon. Graphviz and Dynagraph – Static and Dynamic Graph Drawing Tools. In M. Jünger and P. Mutzel (eds.) Graph Drawing Software, pp. 127–148. Springer, 2004.