

Simulation of Generalized Nets by using GNDraw

Nikolay Ikonov¹, Peter Vassilev², Simeon Ribagin²

¹Institute of Mathematics and Informatics

Bulgarian Academy of Sciences

8 Acad. G. Bonchev Str., 1113 Sofia, Bulgaria

e-mail: `nikonomov@math.bas.bg`

²Institute of Biophysics and Biomedical Engineering

Bulgarian Academy of Sciences

105 Acad. G. Bonchev Str., 1113 Sofia, Bulgaria

e-mails: `peter.vassilev@gmail.com`

`sim_ribagin@mail.bg`

Abstract: This article describes basic simulation of a Generalized Net. This is added as an extension to the existing application for creating Generalized Nets, `GNDraw`, which is written in the Java programming language.

Keywords and phrases: Generalized Net, Simulation, Software, Java.

2000 Mathematics Subject Classification: 68Q85.

1 Introduction

Generalized Nets (see [1, 2]) are an apparatus for modelling of parallel and concurrent processes, developed as an extension of the concept of Petri nets and some of their modifications.

The application `GNDraw` was written specifically for the creation of Generalized Nets by using simple text commands [3]. Naturally, this is extended with a simulation of a Generalized Net, which has the following capabilities: priority of transitions, places and tokens; capacity of places; integer, float and string function types for the characteristics of places and tokens.

The simulation has to be defined by the following commands after the ones for drawing the Generalized Net:

```

simulation
transition:name:priority
place:name:priority:capacity:func_type:func_descr
token:name:start_place:priority:func_type:init_value

```

Each definition for transition, place, token has to be placed on its own row after the word `simulation`. The name for transition and place has to reflect the name from the definition of the net, while the name of the token has to be unique. Priority and capacity have to be natural numbers. Starting place `start_place` of the token has to be a valid name of a place.

Function type `func_type` can be integer or float, everything else is considered a string. Initial value `init_value` should be a number, otherwise it is loaded as string. Function description can be `+2`, `-3`, `+4.1` (plus sign is optional), otherwise it is appended to the token characteristics.

Default values: priority of transition, place, token have default values of 0, capacity of place – 10, `func_type` of place and token – integer, `init_value` of token – 0, `func_descr` of place – 1.

2 Simulation

Let's focus on this basic example:

```

# Example 1
transitions
Z1 : l1 -> l2
Z2 : l2 -> l3
Z3 : l3 i[1] -> l4
Z4 : l4 -> l5
columns
Z1; Z2, Z3; Z4
adjustY
Z1@1; Z4@2
simulation
place:l2:2:1:integer:+2
place:l4:2:1:integer:+4
token:core1:l1:5:integer:0
token:core2:l1:3:integer:0

```

The Generalized Net is defined from the word `transitions` to the word `simulation`. Then the simulation definitions follow: place l_2 has priority 2, capacity 1 and function +2; place l_4 has function +4; two tokens have starting place of l_1 , and token `core1` enters the net first, due to its higher priority.

The initial view of the net is shown on Figure 1. The function of l_1 is 1, the initial value of both tokens is zero, therefore both tokens receive value 1.

Let us advance with one iteration, shown on Figure 2. The priority of `core1` is greater than the priority of `core2`, so `core1` moves first. Since the capacity of l_2 is only one token, then `core1` moves to l_2 , while `core2` stays at l_1 . The function of l_2 is +2, so `core1` now has value 3.

Second iteration is shown on Figure 3. Now `core1` moves to l_3 , and `core2` moves to l_2 . The function of l_3 is 1, therefore the value of `core1` changes to 4, while the value of `core2` changes to 3.

Third iteration on Figure 4: `core1` moves to l_4 , and `core2` moves to l_3 ; the function of l_4 is +4, the value of `core1` becomes 8; the function of l_3 is 1, the value of `core2` becomes 4.

Fourth iteration on Figure 5: `core1` moves to l_5 (its value is now 9), and `core2` moves to l_4 (its value is now 8).

And last iteration on Figure 6: `core2` also moves to l_5 . Final value of `core1` is 9, same for `core2`.

3 Objects for Simulation

The Generalized Net consists of several objects: transitions, places, tokens, arcs. This application has implemented objects for transition, place and token. They exhibit the following properties:

- Transition – name of the transition, list of input places, list of output places, priority.
- Place – name of the place, name of the transition that has this place as output (left transition), name of the transition that has this place as input (right transition), priority, capacity, function type, function description, list of tokens presently at this place, list of tokens that have passed through this place.
- Token – name of the token, priority, function type, initial value, status of the token, name of the place where the token is presently at, list of places that the token has passed through.

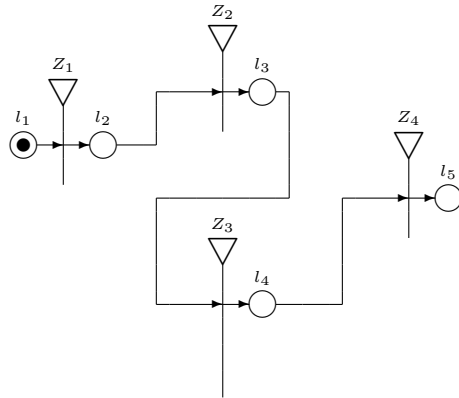


Figure 1: Initial view of the net.

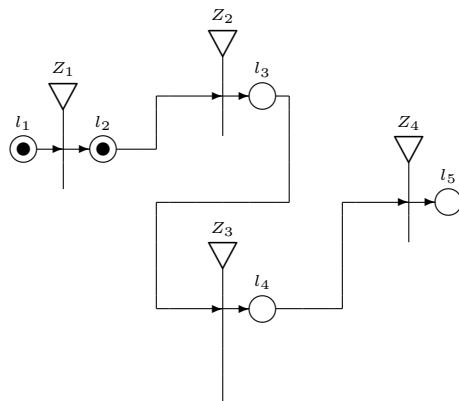


Figure 2: First iteration.

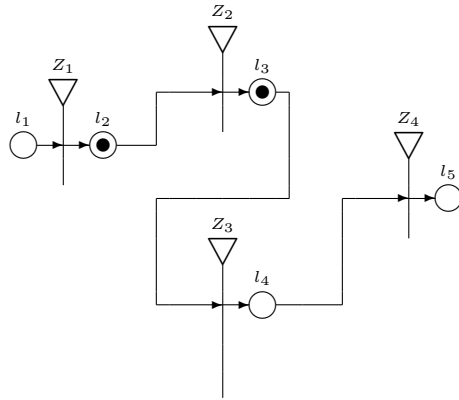


Figure 3: Second iteration.

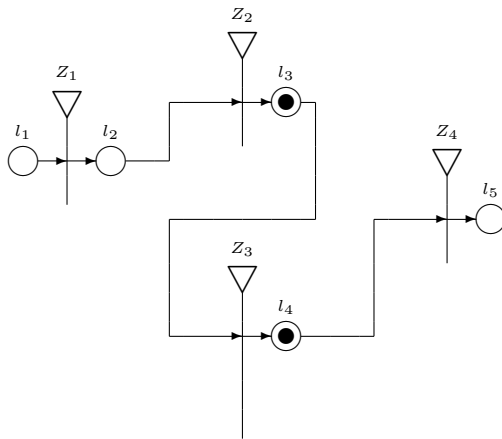


Figure 4: Third iteration.

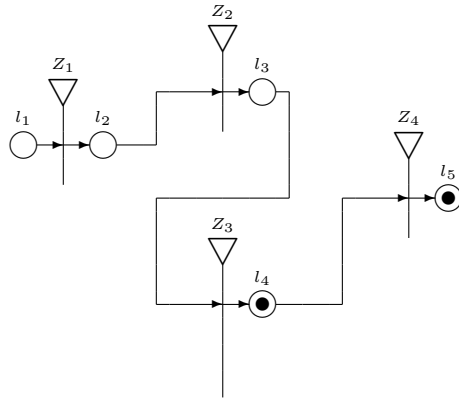


Figure 5: Fourth iteration.

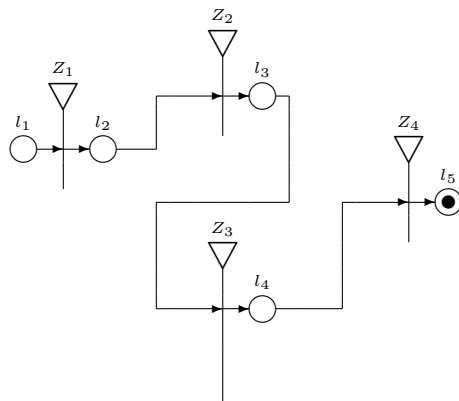


Figure 6: Last iteration.

4 Algorithm for Simulation

The algorithm for simulation attempts to move the tokens from their starting place through the net, until all tokens reach an exit place or the number of maximum iterations is reached.

The first step of the algorithm is to activate the tokens: all tokens are ordered by their priority, then for each token is checked if the token count at the starting place of the token is lower than the capacity:

```
if (place.getTokenCount < place.getCapacity) {  
    place.setTokenHere(token.getName);  
    saveChars(token, place);  
    token.setStatus(1);  
}
```

Then the token count at the place is increased, the characteristics of the place are saved to the token characteristics, and the token is activated.

The second step is to create a list with the active tokens, ordered by their priority. On each iteration, this list is created anew, to filter out tokens that have reached an exit place.

The third step is to start the simulation: for each active token take the place it is presently at (current position). If that place is connected to a transition on the right, then the token might be able to move. Otherwise the place is an exit place, and the token is set as inactive.

Token movement is based on priority: the total priority is the sum of the priority of the current place and the priority of the transition that the place connects to. The token moves to the transition with the highest total priority.

The token is now ready to move: make a list with the output places of that transition, and order them by their priority, select the one with the highest priority, and set it to outPlace.

```
if (outPlace.getTokenCount < outPlace.getCapacity) {  
    outPlace.setTokenHere(token.getName);  
    token.setPlace(outPlace.getName);  
    saveChars(token, outPlace)  
}
```

If the token count is less than the capacity, then move the token: add the token name to the history of outPlace, set the name of outPlace as the current position of the token, save the characteristics of outPlace to the token.

After all tokens have been processed, synchronize the token movement between the places. Each place has a list of tokens presently at it, and some tokens may have moved from the place to another one during the iteration. Check the current position of each token at this place: if the current position of the token is not equal to the name of this place, then remove the token from the list of tokens for the place.

Then the next iteration begins at the third step above. The simulation ends when there are no more active tokens or the maximum iterations are reached, which are defined as a parameter in the user interface of the application.

5 Second Example

This example shows priority of a place and a transition:

```
# Example 2
transitions
Z1 : l1 -> l2
Z2 : l2 l5 -> l3
Z3 : l3 i[1] l8 -> l4
Z4 : l4 -> l5 l6
Z5 : l7 -> l8
columns
Z1, Z5; Z2, Z3; Z4
adjustY
Z5@3
simulation
transition:Z2:4
place:l2:2:1:integer:+2
place:l4:2:1:integer:+4
place:l6:3:2:integer:+1
place:l8:3:2:integer:+1
token:core1:l1:3:integer:0
token:coreA:l7:2:integer:0
token:coreB:l7:7:integer:0
token:coreC:l7:1:integer:0
```

The third iteration of the example is shown on Figure 7. The question is: why the token moves to l_5 instead of l_6 , since l_6 has priority 3, and by default

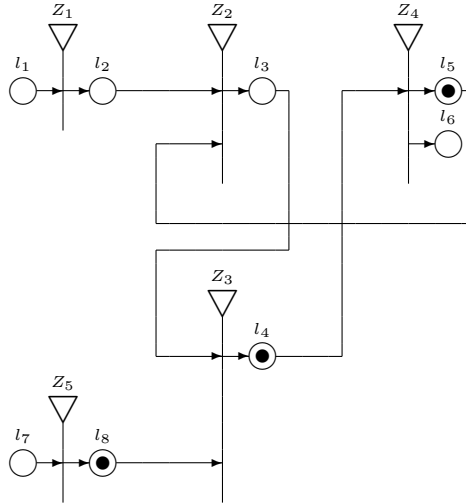


Figure 7: Third iteration of second example.

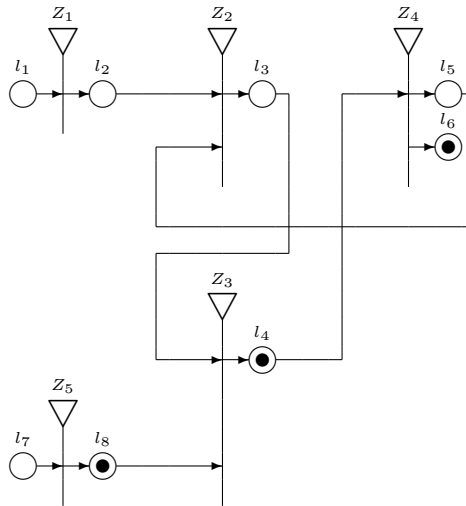


Figure 8: Fourth iteration of second example with increased priority of l_6 .

priority of a place is zero? Priority of a place is the sum of its own priority and the priority of the transition it connects to. Place l_5 connects to transition Z_2 , which has priority 4, therefore the total priority of l_5 is 4, and that is greater than the priority of l_6 .

The priority of l_6 has been increased to 6, and the fourth iteration is shown on Figure 8. This is the final state of the net, since there are 4 active tokens, capacity of l_6 is 2 tokens only, of l_4 is 1 and of l_8 is 2. There are two tokens, `core1` and `coreB`, at l_6 , one token `coreA` at l_4 , and one token `coreC` at l_8 .

6 Third Example

This example shows all capabilities of the algorithm:

```
# Example 3
transitions
Z1 : l1 -> l2
Z2 : l2 l5 -> l3
Z3 : l3 l8 l11 -> l4 l9
Z4 : l4 -> l5 l6
Z5 : l7 -> l8
Z6 : l9 -> l10 l11
columns
Z1, Z5; Z2, Z3; Z4, Z6
adjustY
Z5@2; Z6@1
simulation
transition:Z2:4
transition:Z4:7
place:l4:2:1:integer:+4
place:l6:3:2:integer:+1
place:l7:4:2:integer:+1
place:l9:8:1:integer:+1
token:core1:l1:3:integer:0
token:coreA:l7:2:integer:0
token:coreB:l7:7:integer:0
token:coreC:l7:1:integer:0
token:coreD:l7:5:integer:0
```


This example has 4 tokens that have to enter net at place l_7 . Since capacity of l_7 is 2, only `coreB` and `coreD` enter the net, based on their priority.

The third iteration on Figure 9 has `core1` at l_3 , `coreB` at l_5 , and `coreD` at l_4 . The tokens rotate between these three places.

The tokens move through the lower circuit Z_6 , when priority of Z_4 is decreased to 5 or less, as shown on Figure 10. The third iteration has `coreB` at place l_{10} , `coreD` at l_9 , and `core1` at l_3 , which stays there for this iteration, since capacity of l_9 is 1. The priorities of l_{10} and l_{11} are equal, one of them is selected at random, based on the internal data structure.

7 GNDraw

The application GNDraw is written in the Java programming language [4]. It can be launched from the file `GNDraw.jar` [5]. The user interface is shown on Figure 11: left panel is for input of the Generalized Net, center panel for viewing and exporting the net, right panel is for simulation.

The simulation panel has a parameter for maximum iterations, controls for switching between the iterations, and a panel for displaying the token movement. Each iteration displays different states of the Generalized Net, that were saved during simulation.

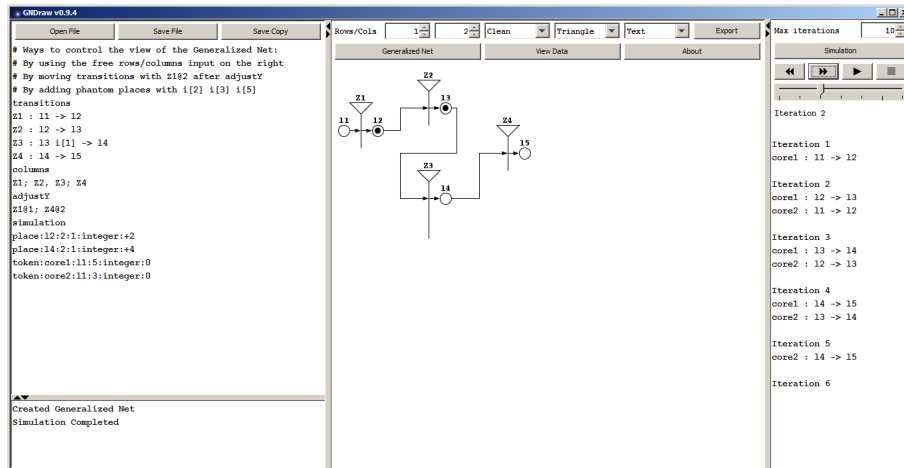


Figure 11: User interface.

8 Conclusion

Generalized Nets can be used for the construction of models of both theoretical and real processes. These models must be simulated in order to determine the behavior of the modeled process. For that purpose an extension of a previously developed software application GNDraw was designed. The new extension to the application enables basic simulation, in addition to the existing construction of a Generalized Net model through a graphical user interface. Advanced simulation techniques are planned as future development.

Acknowledgements

P. Vassilev explained in detail the theory behind the simulation of a Generalized Net, tested the program and provided suggestions, which were implemented. S. Ribagin tested the program and found some issues, which were corrected in version 0.9.4.

This work is partially supported by the Bulgarian National Science Fund under Grant Ref No DN-02-10 “New Instruments for Knowledge Discovery from Data, and their Modelling”.

References

- [1] Atanasov, K., Theory of Generalized Nets (An algebraic aspect), *Advances in Modelling & Simulation*, AMSE Press, Vol. 1, 1984, No. 2, 27–33.
- [2] Atanasov, K., *On Generalized Nets Theory*, “Prof. M. Drinov” Academic Publishing House, Sofia, 2007.
- [3] Ikonov, N., GNDraw – Software Application for Creating Generalized Nets, *Issues in Intuitionistic Fuzzy Sets and Generalized Nets*, in press.
- [4] <https://java.com/>
- [5] <http://justmathbg.info/files/math/GNDraw094.zip>