Proceedings of the Jangjeon Mathematical Society
29 (2026), No. 1, pp. 155 – 167

www.jangjeon.or.kr
http://dx.doi.org/10.17777/pjms2026.29.1.011

Representing Finite-State Automata with Generalized Nets: Simulation via OnlineGN

Angel Dimitriev[1,2]
[1]Department of Bioinformatics and Mathematical Modelling,
Institute of Biophysics and Biomedical Engineering,
Bulgarian Academy of Sciences,
105 Acad. G. Bonchev Str., 1113 Sofia, Bulgaria
e-mail: `angel_dimitriev@abv.bg`

[2]Faculty of Mathematics and Informatics,
Sofia University "St. Kliment Ohridski",
5 James Bourchier Blvd., Sofia 1164, Bulgaria
e-mail: `adimitriev@fmi.uni-sofia.bg`

ABSTRACT: This article introduces a new method for simulating Finite State Automata and Nondeterministic Finite State Automata by employing Generalized Nets. A detailed schema is presented, illustrating how states and transitions can be directly represented as Generalized Net places, transitions, and tokens. An example demonstrating how to simulate an FSA is provided using the OnlineGN software.

Keywords and phrases. Generalized nets, Nondeterministic Finite State Automata.

1 Introduction

Non-deterministic finite automata (NFAs) are fundamental to formal language theory, but creating an executable framework that can simulate one NFA and scale to coordinate entire sets of interacting NFAs quickly becomes hard. Generalized Nets (GNs) offer a leaner alternative: their concurrent semantics allow a single GN to capture every branch of an NFA and even to host several automata at once inside one model.

In this article we introduce a direct translation from any NFA to an equivalent GN and prove that one GN can simultaneously simulate any finite collection of automata while preserving the language of each. The result highlights the efficiency of GNs and their practical value for large-scale, multi-process systems.

2 A Brief Note on Generalized Nets

Generalized Nets (GNs) are recognized as a broad extension of Petri Nets (PNs) and their other variants [1, 2]. In this work, we employ a simplified version of GNs, incorporating only the minimal set of GN components required for our model.

Each transition in this simplified GN is shown in Figure 2, and is formally defined by:

$$Z = \langle P', P'', R \rangle,$$

where P' and P'' are finite, non-empty sets of places, and R is an index matrix (see [3]) composed of predicate elements:

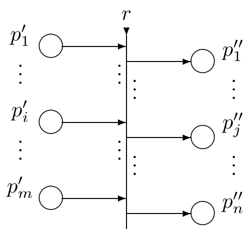


Figure 1: Structure of a GN transition.

$$R = \begin{array}{c|ccccc} & p''_1 & \dots & p''_j & \dots & p''_n \\ \hline p'_1 & & & & & \\ \vdots & & & & & \\ p'_i & & & r_{i,j} & & \\ \vdots & & & & & \\ p'_m & & & & & \end{array}$$

Each element $r_{i,j}$ is a predicate associated with the i-th input place and j-th output place ($1 \leq i \leq m$, $1 \leq j \leq n$). When $r_{i,j}$ is evaluated to *true*, a token from p'_i moves to p''_j.

A simplified GN is then denoted by:

$$E = \langle A, K, X, \Phi \rangle,$$

where A is the set of transitions of the form shown above, K is the set of GN tokens, X is the collection of initial characteristics assigned to tokens upon entry, and Φ is the function that updates each token's characteristic as it transitions from an input place to an output place within a given transition.

3 Essential Concepts of Finite State Automata.

In this section, we introduce the fundamental concepts of *finite state automata*, including both deterministic and non-deterministic variants. We begin by defining finite alphabets, strings (words), and languages, and then present the formal definitions of *deterministic finite automata* (DFA) and *non-deterministic finite automata* (NFA). We also discuss how an automaton accepts a word and how we describe the language recognized by a finite automaton.

3.1 Finite Alphabets and Languages

A *finite alphabet* is a finite set of symbols, usually denoted by Σ. A *string* (or *word*) over Σ is a finite sequence of symbols from Σ. The empty string (with no symbols) is denoted by ε. A *language* L over Σ is any set of strings over Σ, i.e., $L \subseteq \Sigma^*$ (Σ^* is the set of all string with symbols from Σ).

3.2 Deterministic Finite Automaton (DFA)

A *deterministic finite automaton* (DFA) is a 5-tuple [4, 5]

$$\mathcal{A} = (Q,\ \Sigma,\ \delta,\ q_{start},\ F),$$

where:

- Q is a finite set of *states*.
- Σ is a finite *alphabet*.
- $\delta : Q \times \Sigma \to Q$ is the *transition function*, which must be total (i.e., defined for every pair (q, a) with $q \in Q$ and $a \in \Sigma$).
- $q_{start} \in Q$ is the *start state*.

- $F \subseteq Q$ is the set of *accepting* (or *final*) states.

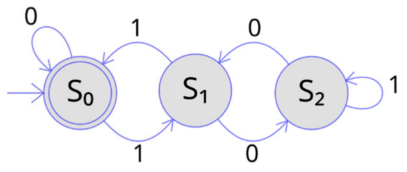


Figure 2: Example of a deterministic finite automaton.

3.2.1 When Does a DFA Accept a Word?

Let $\alpha = a_0a_1 \cdots a_{n-1}$ be a word over the alphabet Σ. The DFA $\mathcal{A}$ *accepts* (or *recognizes*) α if there exists a sequence of states $q_0, q_1, \ldots, q_n \in Q$ such that:

- $q_0 = q_{start}$,
- $\delta(q_i, a_i) = q_{i+1}$ for each $0 \le i < n$,
- $q_n \in F$.

Equivalently, we can use the extended transition function $\delta^* : Q \times \Sigma^* \to Q$, defined inductively by

$$\delta^*(q, \varepsilon) = q \quad \text{and} \quad \delta^*(q, \alpha a) = \delta\big(\delta^*(q, \alpha), a\big).$$

Then $\mathcal{A}$ accepts α if and only if $\delta^*(q_{start}, \alpha) \in F$.

3.2.2 Language of a DFA

The *language recognized* by the DFA $\mathcal{A}$ is

$$L(\mathcal{A}) \;=\; \{\, \alpha \in \Sigma^* \mid \mathcal{A} \text{ accepts } \alpha \}.$$

3.3 Non-deterministic Finite Automaton (NFA)

A *non-deterministic finite automaton* (NFA) is a 5-tuple [4,6]:

$$\mathcal{N} = (Q,\, \Sigma,\, \Delta,\, q_{start},\, F),$$

where:

- Q is a finite set of *states*.
- Σ is a finite *alphabet*.
- $\Delta : Q \times \Sigma \to \mathcal{P}(Q)$ is the *transition function*, which gives a *set of possible next states* for each state-symbol pair.
- $q_{start} \in Q$ is the *start state*.
- $F \subseteq Q$ is the set of *accepting* (or *final*) states.

3.3.1 When Does an NFA Accept a Word?

Let $\alpha = a_0a_1 \cdots a_{n-1}$ be a word. The NFA $\mathcal{N}$ *accepts* α if there exists:

- A start state $q \in Q_{start}$,
- A sequence of states $q_0, q_1, \ldots, q_n$ with $q_0 = q$,

such that for every i with $0 \le i < n$, the state q_{i+1} is in $\Delta(q_i, a_i)$, and finally $q_n \in F$. Equivalently, one can define an extended transition relation or a function

$$\Delta^* : \mathcal{P}(Q) \times \Sigma^* \;\longrightarrow\; \mathcal{P}(Q)$$

and say that α is accepted if

$$\Delta^*(Q_{start}, \alpha) \;\cap\; F \;\neq\; \emptyset.$$

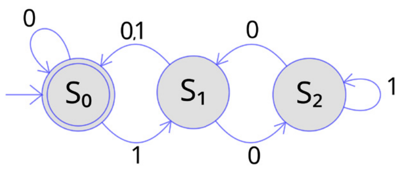


Figure 3: Example of a non-deterministic finite automaton.

3.3.2 Language of an NFA

The *language recognized* by the NFA $\mathcal{N}$ is

$$L(\mathcal{N}) \;=\; \{\alpha \in \Sigma^* \mid \mathcal{N} \text{ accepts } \alpha\}.$$

3.4 Comparison of Deterministic and Non-deterministic Automata

Even though DFAs and NFAs are defined differently, they recognize the same class of languages. Below, we highlight some core differences and the well-known result that connects them:

3.4.1 Unique Computation Path (DFA)

A *deterministic* finite automaton has, for each state and input symbol, exactly one possible next state. When processing an input string, the DFA follows a *single, unique computation path* from the start state to a (potentially) accepting state.

3.4.2 Parallel Computation (NFA)

In contrast, a *non-deterministic* finite automaton may have multiple possible next states for a given state-symbol pair. We can view the NFA as exploring *all possible* computation paths in parallel. If *any* of these paths leads to an accepting state, the input is considered to be accepted. This branching structure is often advantageous for making the design of an automata more simple.

3.4.3 Interpreting a DFA as an NFA

Every DFA can be trivially interpreted as an NFA. The transition function will produce a *singleton set* of next states. Thus, DFAs may be interpreted as special cases of NFAs with a single possible next state at every step.

3.4.4 Rabin–Scott Theorem and the Powerset Construction

An important theorem in formal language theory, established by Rabin and Scott [7], states that for every NFA there exists an equivalent DFA that recognizes the same language. This *equivalence* is typically demonstrated via a *powerset (or subset) construction.* Intuitively, the states of the new DFA to correspond to *subsets* of the original NFA's states, thereby capturing all possible non-deterministic moves in a deterministic fashion.

3.4.5 Why NFAs Are Often Used First

In many theoretical applications, it is sometimes *easier* to design an NFA, due to its flexibility in allowing multiple transitions. Once an NFA is specified, one can systematically convert it to a DFA if needed.

4 Representing Non-Deterministic Finite Automata Using Generalized Nets

As mentioned before, Generalized Nets (GNs) provide a flexible framework for simulating parallel processes [1, 3], making them suitable for modeling how words are accepted by automata. In this section, we demonstrate how a Non-Deterministic Finite Automaton (NFA) can be represented as a Generalized Net. Since every Deterministic Finite Automaton (DFA) can be easily interpreted as an NFA (by viewing each transition as producing a singleton set of next states), the same construction applies to DFAs with minimal modification. Figure 5 shows an induced sub-automaton that we will use as an example later.

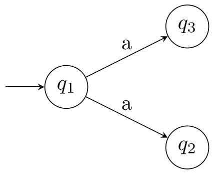


Figure 4: Induced sub-automaton with two arcs labeled 'a' from the same state.

We construct the Generalized Net (GN) to capture all possible parallel computation paths of the given Non-deterministic Finite Automaton (NFA). Each *transition* in the GN corresponds to a single *state* in the automaton, and each *place* in the GN represents an *arc* (or transition edge) of the automaton. We also introduce additional transitions and places as explained below.

Transitions

- **One Transition per Automaton State.** For every state q_i in the NFA, we define a corresponding GN transition Z_i. Conceptually, this transition handles all movements from state q_i to any of its successors, as determined by the automaton's transition function.
- **Final Transition.** Besides the transitions that map each state to its successors, we introduce a special *final transition* Z_{final}. This transition permits a *token* to *exit* the GN once the automaton has accepted the input word. The final transition has multiple input places (each connected from transitions corresponding to final states in the automaton) and a single output place that leads out of the net, signaling acceptance.

Places

- **Arcs as Places.** Each *arc* (transition edge) of the original NFA becomes a corresponding *place* in the GN.
- **Input/Output Places in the GN.** An arc from state q_i to state q_j in the NFA is represented as an *output* place for the GN transition Z_i and an *input* place for the GN transition Z_j. This ensures that a token moving through the net reflects a valid transition in the NFA.
- **Places Leading to the Final Transition.** For every final (accepting) state in the NFA, we add an additional output place from the corresponding GN transition. This output place serves as the input place for the *final transition* Z_{final}.
- **Initial Place.** The GN contains a dedicated initial place (p_{start}) where the initial token first appears. This place is the input place for the transition that corresponds to the NFA's start state.
- **Final Place.** The GN contains a dedicated final place, which is the single output place of the final transition Z_{final}. Tokens reaching this place represent successful acceptance by the automaton.

Tokens and Their Characteristics

Within a GN, the active entities traveling between places are called *tokens*. In this construction, each token corresponds to a distinct potential computation path in the NFA.

- **Word Remainder as a Characteristic.** Each token carries, as its *characteristic*, the remaining (unprocessed) portion of the input word – it is called `"word"`. Initially, the token's `"word"` characteristic is the full input word $w = a_0 a_1 \dots a_{n-1}$.
- **Characteristic Functions of Places.** Each output place for a transition in the GN has a characteristic function which updates the token's remaining input word. When a token moves through an output place corresponding to a non-initial state, the first symbol of the word remainder is removed (i.e., one symbol is considered "read"). However, for the output place of the transition corresponding to the initial (start) state, no symbol is removed when a token first arrives from the GN's initial place.
- **Parallel Paths.** If the NFA branches into multiple successor states from the same configuration, the GN splits the token to multiple tokens (each carrying an identical current word remainder). These tokens proceed in parallel along different transitions.
- **Empty String.** When a token's characteristic becomes the empty string, it indicates that the entire input word has been processed. If the token is on a place leading from (or belonging to) a final state, it may proceed to the *final transition* Z_{final} and exit the GN, signifying acceptance of the word.

Illustration of Token Movement

Consider the example in Figure 4, where an NFA state has two arcs labeled 'a' going to different states (e.g., q_2 and q_3). In the GN, the token *splits* into two tokens, each with the same remaining input. Suppose the original word is $a\alpha$, with $\alpha \in \Sigma^*$. After the split, both tokens carry the same remainder $a\alpha$.

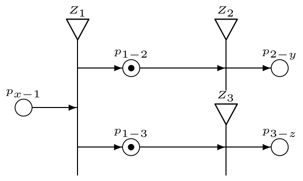


Figure 5: GN corresponding to an NFA from Figure 4 with two active cores BEFORE a step.

For place names, we use the convention $p_{\text{origin-destination}}$. In the subsequent move, the tokens advance to p_{2-y} and p_{3-z}. The characteristic functions of these places remove the first letter of the word; thus both tokens will carry the updated remainder α:

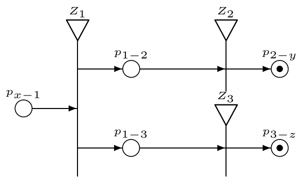


Figure 6: GN corresponding to an NFA from figure 4 with two active cores AFTER a step.

Index Matrices

In this section, we describe how to construct the index matrices of the generalized net (GN) that simulates a given Non-Deterministic Finite Automaton (NFA). Recall that an NFA is defined as a 5-tuple $(Q, \Sigma, \delta, q_{start}, F)$.

Key Predicate Definition

We introduce the following predicate used within the index matrices:

$$S(x,y) \iff \begin{array}{l} \text{The characteristic "word" of the token is } aw \quad (a \in \Sigma,\ w \in \Sigma^*), \\ \text{and } q_y \in \delta(q_x, a). \end{array}$$

Here, q_x corresponds to the NFA state associated with the input place p_x in the GN transition, and q_y is the state associated with the output place p_y. The predicate $S(x, y)$ is thus true precisely when the token at place p_x can move on input a to state q_y in the NFA, with the characteristic word updated from aw to w.

Transition Z_i for a Non-Final State q_i. For each state $q_i \in Q$, let *Input*(q_i) be the set of NFA states that have a direct transition (arc) into q_i, and let *Output*(q_i) be the set of states to which there is a transition (arc) from q_i. In the GN, these will correspond respectively to the input and output places of the transition Z_i. Formally, the index matrix of Z_i has rows indexed by the places corresponding to *Input*(q_i) and columns indexed by the places corresponding to *Output*(q_i).

Hence, the index matrix for the transition Z_i is:

$$\begin{array}{r|c} & p_{i-y} \quad \text{where } q_y \in \mathit{Output}(q_i) \\ \hline p_{x-i} \quad \text{where } q_x \in \mathit{Input}(q_i) & S(x,i) \end{array}$$

Initial State Transition with Additional Place. For the transition that corresponds to the start state, we introduce an additional place p_{start} from which the simulation begins. Any token in p_{start} can move (unconditionally) into one of the places that correspond to *Output*(q_{start}). Thus, in the index matrix for the start-state transition, there is an extra row for p_{start}, with all entries set to True:

$$\begin{array}{c|cccc} & \dots & & & \dots \\ \hline \vdots & \ddots & & & \vdots \\ p_{\text{start}} & \text{True} & \text{True} & \dots & \text{True} \end{array}$$

Additional Output Place for Final States. For each transition Z_i that corresponds to a final state $q_i \in F$, we introduce an additional output place $p_{i,\text{final}}$ that serves as an input to the ultimate "accepting" transition Z_{final}. The movement from the final-state transition into $p_{i,\text{final}}$ is always permitted (i.e., the predicate is True). Consequently, for each such Z_i, the index matrix has an extra column corresponding to $p_{i,\text{final}}$, where each entry (in that column) is set to True:

$$\begin{array}{c|ccc} & \dots & \dots & p_{i,\text{final}} \\ \hline \vdots & \ddots & \ddots & \text{True} \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \text{True} \end{array}$$

Final Transition (Z_{final}). Lastly, we define the global final transition Z_{final}. This transition takes tokens from the places $p_{t,\text{final}}$, where $q_t \in F$, and moves them to a single "accepting" output place P_{final}. In order for the NFA simulation to accept the input word, the characteristic word of the token must be empty. We therefore use the predicate:

$$E \iff \text{the characteristic "word" of the token is empty.}$$

Thus, the index matrix for Z_{final} becomes:

$$\begin{array}{c|c} & P_{\text{final}} \\ \hline p_{t,\text{final}} \ (q_t \in F) & E \end{array}$$

If E is satisfied, the token moves into P_{final}, indicating acceptance of the original input word by the simulated NFA.

Theorem 1. *Every non-deterministic finite automaton (NFA) can be represented by a corresponding Generalized Net.*

Proof. We have demonstrated a procedure for constructing a generalized net (GN) that simulates a given nondeterministic finite automaton (NFA). The procedure begins by creating a *transition for each state* of the NFA and introducing places corresponding to each incident arc of that state. By applying **Theorem (Completeness of GN Transitions)**, which states that every generalized net is the union of its constituent transitions, we establish that the union of these per-state transitions yields a fully formed GN [1, 3].

Specifically, we combine the initial special place, the set of constructed state transitions, and a final transition whose output place corresponds to an accepting configuration. In this way, any token starting in the initial place and ending in the final place set corresponds precisely to the acceptance of the input word in the original NFA. Hence, the constructed GN accurately reflects the operation of the NFA, thereby confirming the correctness of our construction.

□

5 Formal Flow and exapmle

5.1 Flow in the Generilized net

1. **Initialization.** A single token (core) enters the GN with its characteristic set to the full input word α. This token is placed in the input place of the GN that corresponds to the *start state* of the NFA.

2. **State Transitions.** At each GN transition Z_q, the predicate logic checks whether the token's next input symbol matches a valid transition from q to another state q'. If so, the token moves to the corresponding *place* (which represents the arc $q \to q'$). The characteristic string for that token is updated, removing the symbol just consumed.

3. **Branching.** If there are multiple valid transitions from q (reflecting the non-deterministic nature of the NFA), multiple tokens can emerge from Z_q, each with the same characteristic but leading to different subsequent transitions in the GN.

4. **Acceptance.** If a token has an empty string in its characteristic and is in (or can move into) a place corresponding to a final NFA state, it routes to the *final transition* Z_{final}. From there, it *exits* the GN, indicating successful acceptance of the input word.

By following these steps, the GN effectively simulates all possible computation paths of the NFA in parallel. Hence, the word is accepted if and only if at least one token reaches and passes through the final transition. This modeling approach highlights the versatility of Generalized Nets in simulating not only purely *deterministic* mechanisms (DFAs) but also systems requiring *parallel exploration* of states (NFAs).

5.2 Detailed example for an NFA represented as a Generilzed Net

Below is a detailed example of a GN for an NFA:

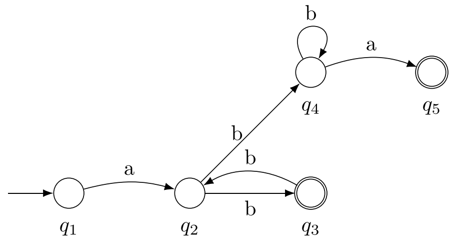


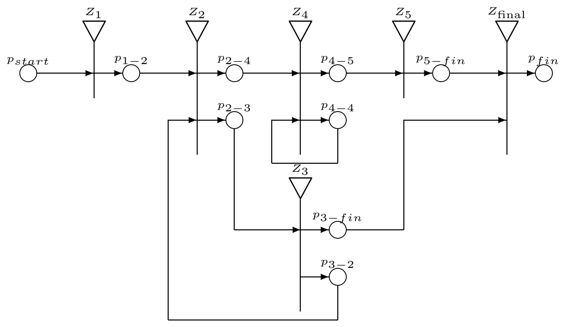
Figure 7: Example NFA with two final states, q_3 and q_5.

The language of this automaton can be represented by the regular expression

$$a\,(bb)^*\Big(b + b^+\,a\Big).$$

It is non-deterministic because from state q_2 there are two arcs labeled ‘b‘.

In the next figure, the generalized net that simulates this automaton is shown:



As described, we have one transition for each state $(Z1, Z2, Z3, Z4, Z5)$ and one additional final transition (Z_{final}).

6 Simulating Multiple NFAs with a Single Generalized Net

In this section, we demonstrate that a finite collection of NFAs can be represented by a single Generalized Net. This representation facilitates the simultaneous verification of input strings across multiple NFAs through a single Generalized Net.

Theorem 2. *Every finite set of NFAs can be represented by a single Generalized Net.*

Proof. We showed that any single NFA can be simulated by a corresponding Generalized Net. Moreover, according to a theorem, if E_1 and E_2 are Generalized Nets, then their union $E_1 \cup E_2$ is also a Generalized Net [1]. Consequently, by taking the union of the Generalized Nets corresponding to each NFA in the finite set, we obtain a single Generalized Net that collectively represents all of the NFAs in question. $\square$

7 Simulating a single NFA with OnlineGn

In this example, we introduced an NFA and constructed a corresponding GN to recognize the language containing the word `abbba`. We now demonstrate how to run this GN in the OnlineGn tool. In the simulation, we use a slightly different notation (since lower indices are not supported). For instance, q_{i-j} is written as `qi-j`.

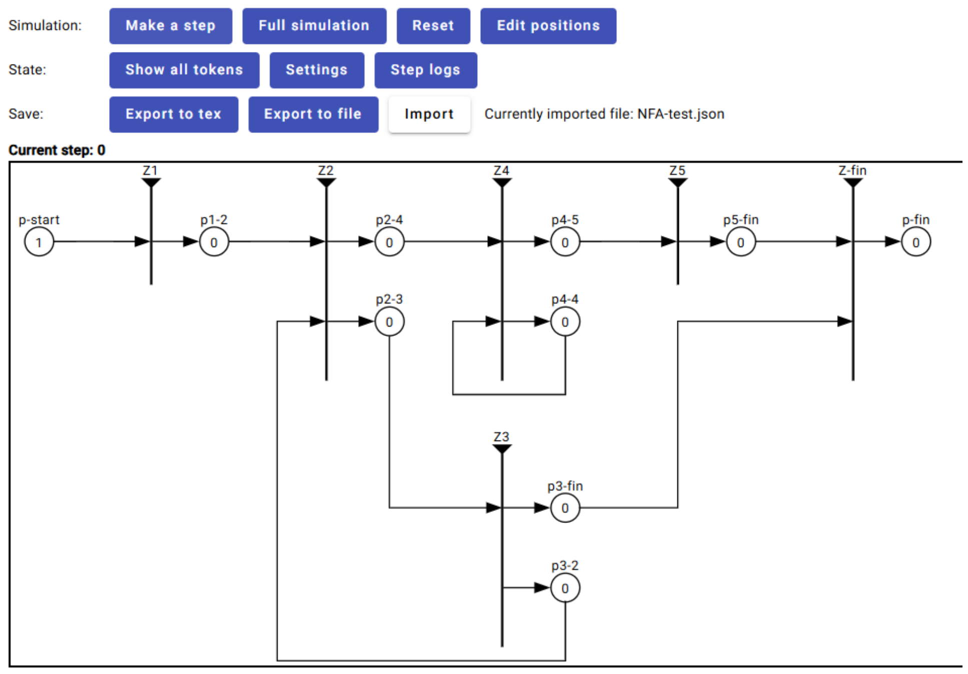
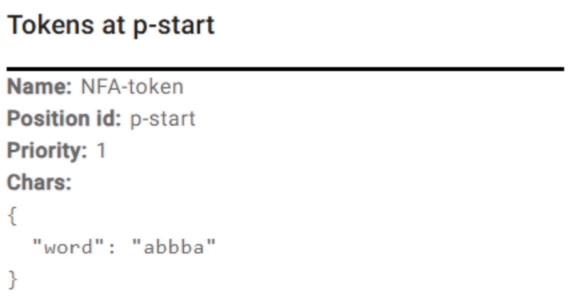


Figure 8: Initial GN setup with one token in the start place (`p-start`).

As shown in Figure 8, the GN begins with a single token located in the start place, denoted by `p-start`. The characteristic of this token is the full word `abbba`, indicating that the entire input is yet to be processed.



Tokens at p-start

Name: NFA-token
Position id: p-start
Priority: 1
Chars:

```
{
  "word": "abbba"
}
```

Figure 9: Token at `p-start` with characteristic `abbba`.

In the first simulation step, the token transitions from `p-start` to the place `p1-2` (see Figures 10 and 11). Note that the token's characteristic remains unchanged after this initial move because the processing of the input symbols (i.e., removing characters) has not yet begun.

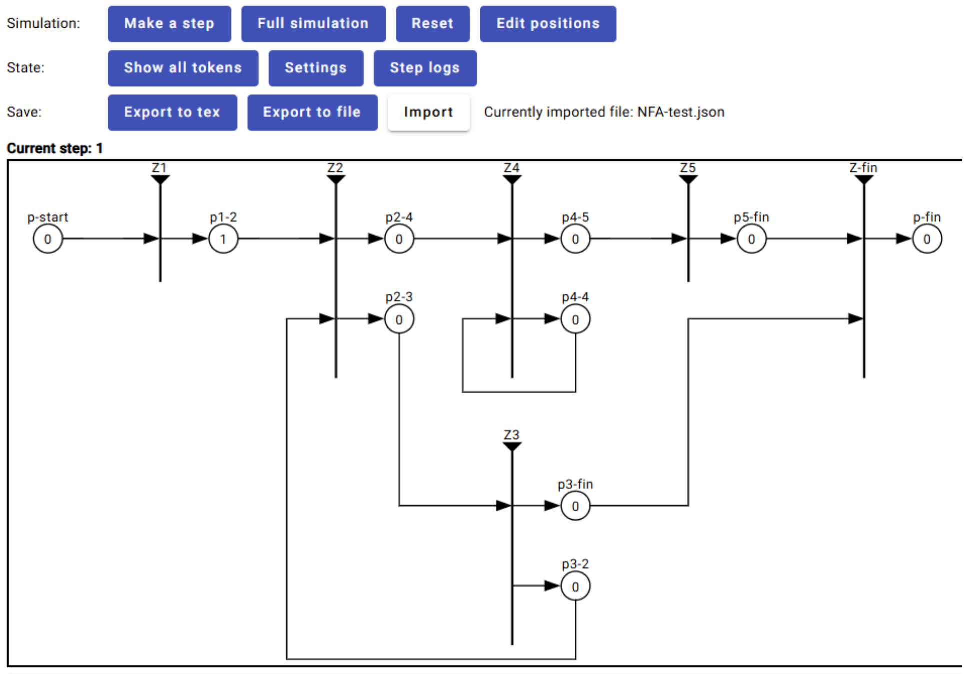
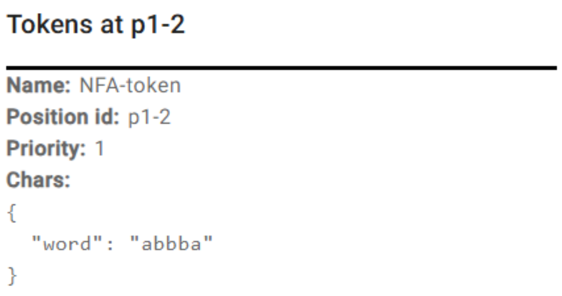


Figure 10: After the first step, the token has moved to `p1-2`.



Tokens at p1-2

Name: NFA-token
Position id: p1-2
Priority: 1
Chars:

```
{
  "word": "abbba"
}
```

Figure 11: Zoomed view of the token at `p1-2`, still carrying `abbba`.

Subsequent steps in the GN correspond to consuming each character in the word `abbba` and following the corresponding transitions. Eventually, once all characters have been consumed, the token is routed to the final place, `p-fin`, as shown in Figure 12.

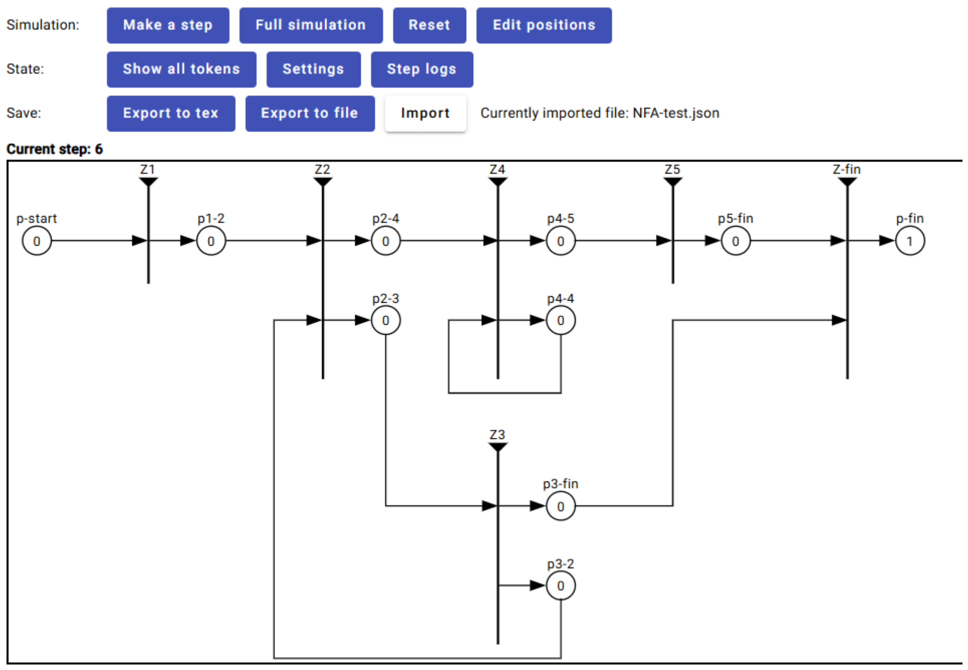
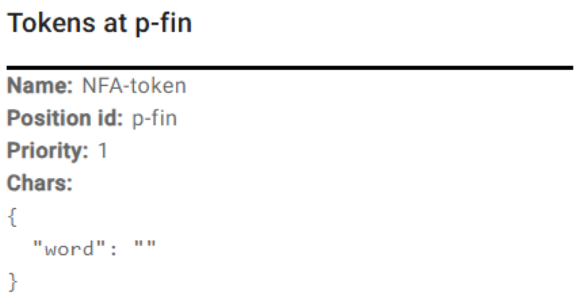


Figure 12: The token has reached the final place `p-fin`.

By the time the token arrives at `p-fin`, its characteristic is the empty string, indicating that the entire word `abbba` has been recognized (or "accepted") by the GN/NFA. This final state signifies successful completion of the simulation.



Tokens at p-fin

Name: NFA-token
Position id: p-fin
Priority: 1
Chars:

```
{
  "word": ""
}
```

Figure 13: The token's characteristic becomes the empty string upon acceptance.

This series of steps demonstrates how the OnlineGn tool can be used to visualize and track the movement of tokens and the progression of their characteristics (the input symbols) through a generalized net that models an NFA.

8 Conclusion

In this paper, we presented how Generalized Nets can be employed to simulate both deterministic and non-deterministic finite automata. By associating each automaton state with a transition in the GN and representing arcs as places, we effectively capture all parallel computation paths that an NFA might follow. The token-based architecture of Generalized Nets allows for easy modeling of branching and the simultaneous exploration of multiple states. This approach showcases the flexibility and robustness of GNs in formal language theory and automata-based computations, paving the way for further applications in areas where parallel processing and concurrency play a significant role.

References

[1] Atanassov, K. *Generalized Nets.* World Scientific, Singapore, 1991.

[2] Alexieva, J.; Choy, E.; Koycheva, E. Review and bibliography on generalized nets theory and applications. In *A Survey of Generalized Nets*; Choy, E., Krawczak, M., Shannon, A., Szmidt, E., Eds.; Raffles KvB Monograph No. 10; Raffles Publishing House: Sydney, Australia, 2007; pp. 207–301.

[3] Atanassov, K. *On Generalized Nets Theory.* ”Prof. Marin Drinov” Academic Publishing House, Sofia, 2007.

[4] Hopcroft, J. E.; Motwani, R.; Ullman, J. D. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 3rd edition, 2006.

[5] Kozen, D. C. *Automata and Computability.* Springer, 1997.

[6] Papadimitriou, C. H. *Computational Complexity.* Addison-Wesley, 1994.

[7] Rabin, M. O.; Scott, D. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.