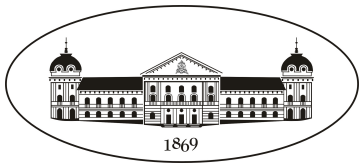


Българска академия на науките

Институт по биофизика и биомедицинско инженерство

Софтуерен продукт за реализация на обобщеномрежови модели и негови приложения

АВТОРЕФЕРАТ НА ДИСЕРТАЦИОНЕН ТРУД

за присъждане на образователна и научна степен „Доктор“

по професионално направление

4.6 „Информатика и компютърни науки“

докторска програма „Информатика“

Докторант: Ангел Иванов Димитриев

Научни ръководители:

акад. дмн дтн Красимир Атанасов

доц. д-р Нора Ангелова

София, 2026 г.

Автореферат на дисертационен труд

Автор: Ангел Иванов Димитриев

Заглавие: Софтуерен продукт за реализация на обобщеномрежови модели и негови приложения

Дисертационният труд е обсъден и допуснат до защита на заседание на Научно звено към ИБФБМИ – БАН на _____ г.

Публичната защита ще се проведе на _____ г. от _____ ч.
в _____.

Дисертационният труд съдържа:
182 страници, 85 фигури, 25 фрагмента от код и 19 формули.

Библиографията включва: 104 заглавия.

Структурата на дисертацията включва: Увод, 4 глави, Заключение, Авторска справка за приносите, Списък на публикациите по темата и Библиография.

- В **Увода** е обоснована актуалността на темата, дефинирани са обектът и предметът на изследване.
- **Глава 1** представя теоретичните основи на обобщените мрежи (ОМ), анализ на съществуващия софтуер за симулация и формулира целта и задачите на дисертационния труд.
- **Глава 2** е посветена на нововъведени алгоритми за автоматично изчертаване на ОМ и преобразуване на графични изображения в *TeX* формат.
- **Глава 3** описва архитектурата, дизайна и имплементацията на разработения софтуерен симулатор *OnlineGN*.
- **Глава 4** разглежда създаването на ОМ модели за процеси, за които до момента не са прилагани ОМ или са използвани други подходи, както и приложенията на симулатора, като всяка от посочените мрежи е симулирана и верифицирана чрез него.
- **Заключението** обобщава постигнатите резултати и очертава насоки за бъдеща работа.

Съдържание

Увод

Теорията на обобщените мрежи (ОМ) предоставя мощен математически апарат за моделиране и симулация на сложни процеси, характеризиращи се с паралелизъм и логическа обвързаност. От своето създаване апаратът на ОМ се утвърждава като гъвкаво средство в области като индустриалното производство, компютърните системи и изкуствения интелект. С нарастването на сложността на моделите обаче необходимостта от достъпни и високоефективни софтуерни решения за тяхното описание и изследване става все по-осезаема.

Въпреки съществуващите разработки, до момента се наблюдава липса на инструменти, които да позволяват работа в онлайн среда, автоматизирано генериране на графични представяния и лесна интеграция в научни публикации. Основната цел на настоящия труд е създаването на софтуерната система ***OnlineGN*** – уеб базиран симулатор, който обединява теоретичните основи на ОМ със съвременните уеб технологии. В рамките на дисертацията се предлагат нови алгоритми за автоматична визуализация и експорт на модели, които превръщат ***OnlineGN*** в комплексна среда за моделиране. Чрез поредица от симулации на класически задачи и реални производствени сценарии се доказват приложимостта и ефективността на разработения инструмент.

Глава 1. Обобщени мрежи: Теоретични основи и софтуер за симулации

1.1. Бележки върху теорията на обобщените мрежи

1.1.1. За понятието обобщена мрежа

Обобщените мрежи (ОМ) са представени за пръв път през 1982 г. в доклад на международна конференция, като текстът е публикуван през 1984 г. [1]. Те се появяват като естествено продължение на усилията за моделиране на паралелни процеси, започнали с изследванията на *C. A. Petri* от 1962 г., довели до формулирането на мрежите на Петри (МП) [2].

В класическия си вид МП са двуделен ориентиран граф с два типа върхове: *преходи* (дискретни събития) и *позиции* (условия), между които се движат *ядра* (*tokens*). Насочването към паралелизма води до множество разширения на МП (над двайсет до 1982 г.), които добавят: (i) времеви аспекти; (ii) оцветяване на ядрата за различимост; (iii) охраняващи условия (предикати) върху групи ядра за синхронизирани преминавания; и др.

На този фон ОМ предлагат по-богата рамка за спецификация на едновременност, синхронизация и условни зависимости, запазвайки интуитивната графова семантика на МП, но разширявайки изразителната мощ за моделиране на сложни паралелни процеси. Подробни разглеждания на ОМ са дадени в [7, 8, 9, 10], а малка част от техните приложения са разгледани в [11, 12, 13, 14].

1.1.2. Неформално описание на означенията в обобщените мрежи

В дисертационния труд са обобщени основните означения и правила в ОМ. Позиции и преходи се свързват с *ребра* (стрелки), които определят възможните направления на движение. В някои позиции се намират *ядра* (точки), аналогично на класическите мрежи на Петри [17]. Входящото ядро носи *начални характеристики*, а при преминаване през мрежата може да получи допълнителни характеристики. Тъй като първоначалните характеристики могат да останат валидни, за ядрото се натрупва *история*. От гледна точка на реализацията, това множество от характеристики може да се възприема като асоциативен списък „име $\rightarrow$ стойност“. Позицията по избор може да има *характеристична функция*, която задава нови стойности на характеристиките на ядрата, влизащи в нея.

Свързването в ОМ е опростено по отношение на степените: всяка позиция е свързана с поне едно ребро и допуска най-много едно входно и едно изходно ребро. Без входно ребро позицията е *входна*, а без изходно – *изходна*. Всеки преход притежава поне една входна и една изходна позиция ($m, n \geq 1$); възможно е при даден преход една и съща позиция да играе и двете роли [8]. Когато ядро достигне входна позиция на преход, то става *потенциално активно*; ако са налице условията за прехвърляне към изходните позиции, преходът е *активен*.

Условията към преходите не се описват с единичен предикат, а чрез *индексирана*

матрица (ИМ) – матрица от предикати, която едновременно специфицира кога е възможно движение по различните направления и определя капацитетите на съответните ребра. Така ИМ осигурява по-фин контрол спрямо стандартните мрежи на Петри [8].

Времето в дефиницията на ОМ е *дискретно*: разглеждат се последователни стъпки, номерирани с целочислена променлива, която нараства с единица. Допустимо е паралелно да се фиксира *абсолютна времева скала* с начален момент и хоризонт на функциониране [8]. При множество свързани процеси може да се използват отделни скали за всеки или обща за синхронизация.

ОМ са достатъчно изразителни, за да опишат както разширенията на мрежите на Петри, така и модели с изчислителна мощ от класа на машината на Тюринг (в частност и крайни автомати) [8]. Известните към момента разширения на ОМ са *консервативни*: за всяко от тях съществува стандартна ОМ, която еквивалентно описва функционирането и резултата му [8, 9].

1.1.3. Формална дефиниция на обобщените мрежи

В този раздел на дисертационния труд се разглежда строго математическо дефиниране на компонентите на ОМ. Формалният апарат се базира на теорията на множествата и апарата на ИМ, като дефинира структурата, времевите параметри и динамиката на ядрата. Формално всеки преход се описва чрез наредената седморка

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

където r е ИМ, която задава условията на прехода, а M е ИМ, която задава капацитетите на ребрата на прехода; преходът се активира в момента t_1, при условие че вярностната стойност на булевия израз е „истина“. ОМ се задава с наредената четворка

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

като се определят преходите и позициите, приоритетите, капацитетите, функциите за време и памет и характеристичните функции. Компонентите на ОМ са класифицирани според статичната структура, динамичната природа, времевата същност и паметта на ОМ, и е въведено понятието *празна ОМ*.

1.1.4. Алгоритми за движение на ядрата в преход и в обобщена мрежа

В този раздел на дисертационния труд се представя разширена версия на най-общия от познатите досега алгоритми за функциониране на преход в ОМ [8, 7, 9, 10].

1.2. Текущ софтуер за симулация на обобщени мрежи

1.2.1. Основни недостатъци на текущите симулатори

В този раздел на дисертационния труд се разглеждат основни недостатъци на текущите симулатори, които засягат едновременно архитектурата, потребителското изживяване и поддръжката. Налице са **фрагментираност и непълна интеграция**,

проблеми с преносимост и зависимости, както и **езиков слой и съвместимост**. Тези и други причини мотивират разработването на *OnlineGN* като уеб базирана, интегрирана и платформено независима среда.

1.2.2. Анализ на съществуващите софтуерни решения

Разглеждат се съществуващи софтуерни решения за моделиране и симулация на ОМ, като се описват архитектурата, компонентите и начините на взаимодействие в ***GN Lite*** (вкл. *GNTicker*, *TickerServer/GNTP*, *GN IDE*, *GNVis*, *XGN2SVG*, *GNProfy*, *GNDraw*), както и подходът на ***GoGN*** за разработване на модели чрез библиотека на *C#*. Акцентът е върху използваните технологии, степента на интеграция между инструментите и практическите аспекти при работа с формата *XGN*.

1.2.3. Основни недостатъци на текущите симулатори

Обобщени са основни недостатъци на текущите симулатори, които засягат едновременно архитектурата, потребителското изживяване и поддръжката.

- **Фрагментираност и непълна интеграция.**
  Ранните версии на *GNTicker* са конзолни, а интерактивността се компенсира чрез *TickerServer*/*GNTP* и отделни помощни инструменти извън основния цикъл моделиране–симулация.
- **Проблеми с преносимост и зависимости.**
  Част от инструментите са насочени към *Windows/.NET* и при *Unix*-подобни ОС често разчитат на *Mono*, което увеличава усилията за инсталация и поддръжка.
- **Езиков слой и съвместимост.**
  GNTCFL увеличава кривата на обучение; *GNJS* цели да я снижи, но прототипи не осигуряват пълна съвместимост с *GNTP*, което ограничава интеграцията.
- **Мащабируемост и производителност.**
  Алгоритмите са коректни, но не винаги оптимални за много големи или силно свързани мрежи, което води до по-дълго време за изпълнение и по-висока ресурсоемкост.
- **Ограничения на инструментариума и екосистемата.**
  XGN2SVG дава статичен изход; *GNVis* е отделен *.NET* модул; *GNProfy* е фокусиран върху диаграми на последователности и по-стари *XMI* версии; *GoGN* изисква програмиране и компилация вместо декларативно моделиране.

1.3. Цел и задачи на дисертационния труд

1.3.1. Формулиране на целта

Целта на настоящия дисертационен труд е да се разработи уеб базиран онлайн симулатор за ОМ – *OnlineGN*, който предоставя интерактивна среда за създаване,

визуализация, редактиране, споделяне и симулация на модели, и да се валидира приложимостта му чрез разработване, симулация и верификация на реални ОМ модели.

1.3.2. Задачи за постигане на целта

За постигане на поставената цел са формулирани следните задачи:

Задача 1. Разработване и имплементация на алгоритъм за автоматично изчертаване на ОМ по тяхно описание в структуриран текстов формат.

Задача 2. Разработване и имплементация на алгоритъм за преобразуване на *SVG* изображение на ОМ в *TeX* формат, с цел лесно включване на визуализации в научни публикации.

Задача 3. Формулиране на изисквания към уеб базирано решение.

Задача 4. Проектиране на архитектурата на системата *OnlineGN* и дефиниране на основните модули (съхранение на мрежи, визуализация, редакция, симулация, импорт/експорт, *API*).

Задача 5. Реализация на уеб базиран графичен интерфейс за визуализация, редакция и персонализация на генерираното изображение (позиции, преходи, връзки и визуални параметри).

Задача 6. Реализация на удобен интерфейс за редакция на предикати в индексираната матрица и характеристични функции на позициите.

Задача 7. Реализация на механизъм за онлайн съхранение и споделяне на мрежи чрез уникална хипервръзка (линк), базиран на централизирано съхранение в база данни.

Задача 8. Създаване и формализиране на нови ОМ модели на реални процеси, които до момента не са описвани чрез ОМ.

Задача 9. Симулация и верификация на разработените модели в *OnlineGN* чрез сценарии и експерименти.

Реализирането на поставените задачи ще доведе до създаването на функционален уеб базиран симулатор, придружен от нови алгоритми за автоматизирано изчертаване и преобразуване към *TeX*. Практическата приложимост на разработката ще бъде потвърдена чрез валидиращи примери и формализиране на нови ОМ модели на реални процеси.

В обобщение, дисертационният труд адресира необходимостта от автоматизация и ефективно сътрудничество при работа с апарата на ОМ. Проектът *OnlineGN* предлага цялостно, гъвкаво и лесно за употреба решение, съобразено със съвременните софтуерни стандарти и нуждите на научноизследователската общност.

Глава 2. Нововъведени алгоритми за обобщени мрежи

2.1. Алгоритъм за автоматично изчертаване на обобщени мрежи

Главата представя решението на **Задача 1** чрез разработване на алгоритъм за автоматично изчертаване на ОМ, подробно описан в [39].

2.1.1. Въведение

При работа с ОМ съществено затруднение е необходимостта от значителен ръчен труд при чертането им. Традиционно координатите на елементите се задават прецизно, което е времеемко и крие риск от човешка грешка. Същият проблем се наблюдава и при ОМ симулаторите, където ръчното въвеждане на координати забавя моделирането и ограничава бързото експериментиране.

В отговор се предлага алгоритъм за автоматизирано чертане, който генерира ясни визуални представяния на ОМ само по тяхното абстрактно описание и елиминира нуждата от ръчно задаване на координати. Така се улеснява създаването и манипулирането на ОМ и се насочва вниманието към анализа, а не към техническите детайли по изобразяването.

2.1.2. Текущи практики при визуализирането на обобщени мрежи

Визуализирането на ОМ изисква прецизно позициониране на позиции, преходи и ребра с цел яснота и структурна коректност. Традиционно това се прави интерактивно чрез ръчно задаване на координати, като се следват утвърдени конвенции за подредба и четимост.

Насоченост на стрелките.
Обичайно стрелките са ориентирани отляво надясно, като изходите на преходите се изобразяват вдясно. При обратни връзки те се добавят така, че да не се нарушава прегледността.

Позициониране на изходни позиции.
Изходните позиции се разполагат непосредствено вдясно на съответните преходи, което улеснява проследяването на потока и логиката на мрежата.

Позиции с двойна роля (вход и изход).
Когато позиция е едновременно вход и изход, тя се позиционира в долната част на прехода, с цел по-подредена диаграма и минимизиране на пресичанията.

Допълнителни практики.
За по-добра четимост се прилагат равномерни отстояния, ясни надписи, при нужда

цветово разграничаване, минимизиране на пресичанията и последователна ориентация на мрежата.

Въпреки че тези практики подпомагат ясната визуализация, ръчното задаване на координати остава трудоемко и податливо на грешки, което мотивира автоматизирани решения. В следващите части се представя алгоритъм за автоматично чертане на ОМ, съобразен с наложилите се конвенции, който намалява ръчния труд и риска от грешки.

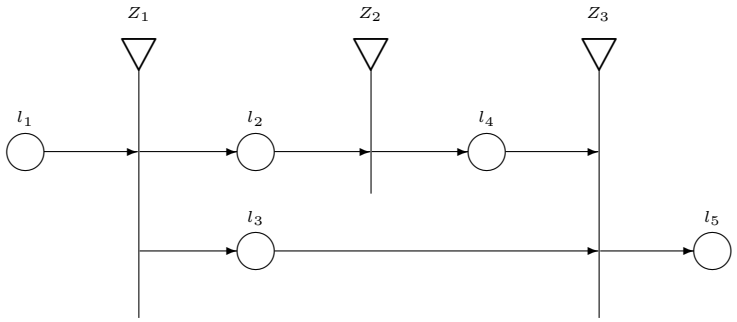
2.1.3. Изчертаване на графи

Визуализацията на графи цели ясно представяне на структурата чрез подходящо разполагане на върхове и ребра с минимални пресичания [31]. За тази цел често се използва *Graphviz* – инструмент за автоматично генериране на графови изображения от *DOT* описание, с различни алгоритми за подреждане (напр. *dot*, *neato*) [33].

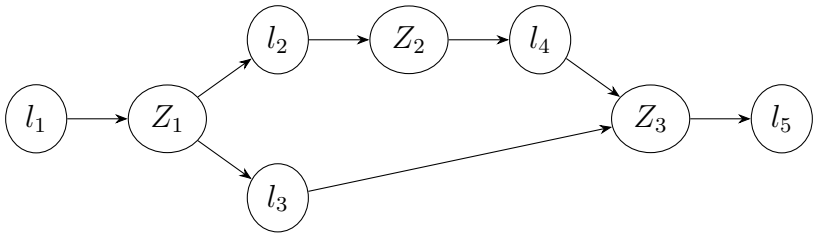
2.1.4. Моделиране на обобщена мрежа като ориентиран граф

За нуждите на визуализацията и структурния анализ ОМ може да се представи като ориентиран граф, при който **позициите** и **преходите** се разглеждат като **върхове**, а **ребрата** описват ориентирани връзки между тях. Чертае се ребро $l \to Z$, когато позиция l е вход за преход Z, и ребро $Z \to l$, когато l е изход от Z. Така посоката на ребрата отразява потока на ядрата в мрежата.

Пример.
Разглежда се ОМ с позиции l_1, l_2, l_3, l_4, l_5 и преходи Z_1, Z_2, Z_3, като връзките са: $l_1 \to Z_1$, $Z_1 \to l_2, l_3$, $l_2 \to Z_2$, $Z_2 \to l_4$, $l_3, l_4 \to Z_3$, $Z_3 \to l_5$. ОМ е показана на **Фигура 5**, а съответстващият ориентиран граф – на **Фигура 6**.



Фигура 5 *– Обобщена мрежа с позиции l_1, l_2, l_3, l_4, l_5 и преходи Z_1, Z_2, Z_3.*



Фигура 6 *– Ориентиран граф, съответстващ на обобщената мрежа. Върховете представляват позиции и преходи, а ориентираните ребра – потока между тях.*

2.1.5. Визуализация на обобщени мрежи с *Graphviz*

За коректна и четима визуализация на ОМ с *Graphviz* се задават параметри и ограничения, които запазват стандартната подредба на елементите.

Задаване на глобални параметри.
Използват се *rankdir=LR* за ориентация отляво надясно и *splines=ortho* за ортогонални ребра, което поддържа решетъчния вид на ОМ и улеснява проследяването на потока.

Дефиниране на върховете за преходи.
Преходите се моделират като правоъгълни върхове (*shape=rect*) с малка фиксирана ширина, а височината се задава според броя изходни позиции, за да се отрази връзката им с множество изходи.

Групиране на изходните позиции с *subgraph*.
Преходът и изходните му позиции се групират в *subgraph* (напр. *cluster*), като се използва *rank=same* за подравняване на изходите и запазване на близостта им до прехода при автоматичното подреждане.

Управление на входните ребра чрез „невидими“ върхове.
За да се получи единна входна точка и входовете да „влизат“ отляво, пред прехода се добавят помощни „невидими“ върхове, към които се свързват входните позиции, а след това те се свързват към самия преход, като относителното им разположение се фиксира чрез групиране.

2.1.6. Алгоритъм за генериране на *Graphviz* низове от обобщени мрежи

В този раздел се описва алгоритмичната логика за генериране на *Graphviz* низ от ОМ (*GN*). Алгоритъмът е създаден за автоматизиране на визуализацията на ОМ и гарантира, че полученият граф следва предварително зададените параметри на подредба. Основната функция, *generateGraphvizString*, използва няколко спомагателни функции,

които отговарят за картографирането на преходи и позиции, генерирането на невидими върхове и създаването на необходимите ребра между елементите в ОМ.

Структурата на псевдокода, обобщаваща основните стъпки и логиката на алгоритъма, е дефинирана по следния начин:

Генериране на невидими върхове.
Функцията *genInvisNodes* генерира невидими върхове, използвани за подравняване на преходите при графичното подреждане на графа.

Алгоритъм 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
```

Свързване на невидимите върхове с преходите.
Функцията *genEdgeBetweenInvisNodesAndTransition* генерира ребра от невидимите върхове към преходите, като осигурява коректно подравняване и свързване.

Алгоритъм 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
```

Генериране на преходи.
Функцията *genTransitionsString* генерира необходимата конфигурация за всеки преход в ОМ, включително невидимите върхове и връзките помежду им.

Алгоритъм 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
```

Генериране на изходни позиции и ребра.
Две отделни функции отговарят за генерирането на изходните позиции и съответните ребра. Така се гарантира, че ребрата между преходите и позициите се създават коректно.

Алгоритъм 4 genOutgoingPlacesFromTransition(genNet, result, transition)

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

Алгоритъм 5 genOutgoingEdgesFromTransition(genNet, result, transition)

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

Генериране на изходни ребра от позициите.
Накрая функцията *genOutputEdgesFromPlaces* генерира ребра, свързващи позициите към невидимите върхове, като така се осигурява коректното подравняване на преходите.

Алгоритъм 6 genOutputEdgesFromPlaces(genNet, result)

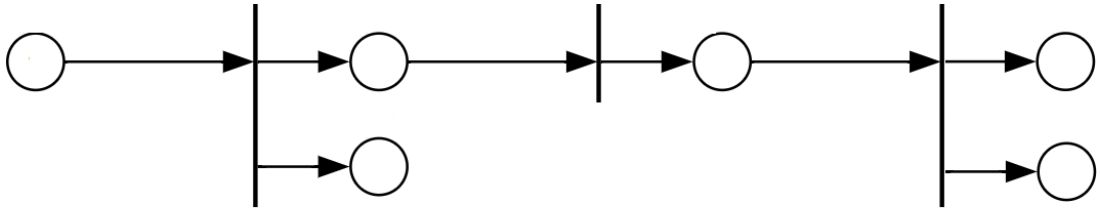
```
for each place in genNet do
   invisNodeIndex ← first invisible node index for the transition
   result ← result + place.name + " -> " + "invis_node_" +
"transitionName" + "_" + invisNodeIndex + "[arrowhead=none]"
end for
```

Главна функция за генериране на *Graphviz* низ.
Главната функция *genGraphvizString* обединява всички предходни функции, като се генерира пълният *Graphviz* низ, представляващ ОМ.

Алгоритъм 7 genGraphvizString(genNet)

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

Резултатът от прилагането на описания алгоритъм за генериране на *Graphviz* низ е показан на **Фигура 8**. Визуализацията отразява автоматично генерираната структура на ОМ преди последваща графична обработка.



Фигура 8 *– Пример за визуализиран граф (обобщена мрежа) с помощта на Graphviz.*

Следваща обработка на генерираното изображение.
В този параграф се описват техниките, които се прилагат към *SVG* (*Scalable Vector Graphics*) файла, генериран от алгоритъма. Първоначално се създава граф, където преходите са представени като много тънки правоъгълни върхове, а позициите – като кръгови върхове. За да се адаптира това визуално представяне към формата на ОМ (*GN*), при последваща обработка се извършват редица модификации.

Целта е полученото изображение да съответства по-добре на формата на ОМ. По-долу са описани конкретните промени:

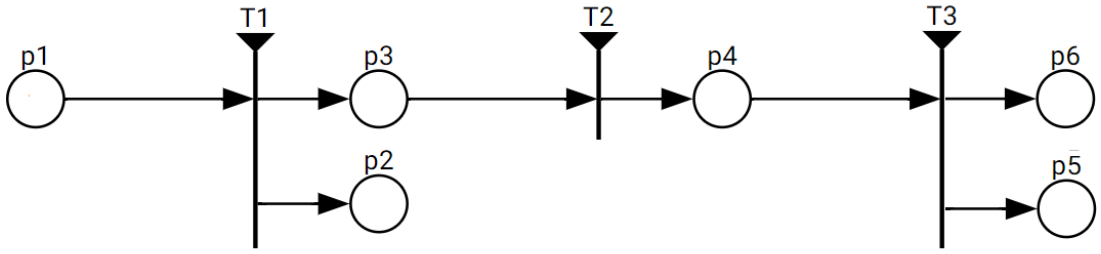
- **Преходи:** Всеки преход, първоначално дефиниран като тесен правоъгълник, подлежи на допълнителна обработка – в горната му част се добавя черен триъгълник, който служи за графичен идентификатор и подобрява неговата видимост в общата структура. Над триъгълника се визуализира етикет с наименованието на прехода, което гарантира неговата еднозначна идентификация в модела.

- **Позиции:** При позициите, представени с окръжности, се добавят имената им над всяка окръжност. Това е необходимо, тъй като инструментът за генериране на първоначалния *SVG* (*Graphviz*) не поддържа по подразбиране този начин на надписване. Затова ръчно се вмъкват нужните надписи, за да се осигури яснота в крайното визуално представяне.

- **Визуализиране на обобщената мрежа:** След прилагането на горните промени полученият *SVG* значително наподобява ОМ. Преходите са ясно маркирани и надписани, а позициите (места) имат четливо поставени наименования над своите окръжности. Този подход води до по-интуитивна и точна графична репрезентация на системата, която се моделира, улеснявайки нейното тълкуване и анализ.

  Крайният резултат след прилагане на описаните стъпки за последваща *SVG* обработка е показан на **Фигура 9**. Полученото изображение следва визуалните конвенции на ОМ и позволява интуитивно възприемане на структурата и потока.

- **Пример за резултатно изображение:**



Фигура 9 *– Пример за визуализирана обобщена мрежа след обработка.*

- **Псевдокод на стъпката по обработка:**

Алгоритъм 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
```

- **Псевдокод на основната функция за чертане:**

Алгоритъм 9 drawGenNet(genNet)

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

2.2. Алгоритъм за преобразуване на *SVG* изображение на обобщена мрежа в *TeX* формат

В този раздел се представя алгоритъм за автоматично преобразуване на *SVG* изображение на ОМ в *TeX* код, с което се решава **Задача 2**.

2.2.1. Въведение и цел на алгоритъма

SVG е удобен векторен формат за генериране и редакция, но в научни текстове най-често се използва *TeX*, където фигурите трябва да се вписват чисто и консистентно със стила на документа. Затова се въвежда метод за пренасяне на *SVG* изображение на ОМ в *TeX* без ръчно пречертаване.

Алгоритъмът извлича позиции, преходи и връзки от входния *SVG* и генерира еквивалентна схема чрез `\put`, `\circle`, `\line` и `\vector`. Полученият код се вмъква директно в документ и минимизира нуждата от ръчни корекции.

2.2.2. Описание на формата на входния *SVG* файл

Алгоритъмът работи със *SVG* с предварително зададена структура. Всеки елемент има атрибут `class`, който указва дали е позиция, преход или ребро. Компонентите на ОМ са представени като групи `<g>`:

- ***Places* (позиции)**: `class` по шаблон `"node place {id}"`. Вътре има `<ellipse>` (визуализация) и `<title>`, `<text>` (идентификатор).
- ***Transitions* (преходи)**: `class` по шаблон `"node transition {id}"`. Групата съдържа два `<polygon>` (вертикална черта и триъгълник), маркирани с `class "triangle"`, както и `<title>` и `<text>` за идентификатора.
- ***Edges* (ребра)**: `class` по шаблон `"edge {originId}__{destinationId}"`. Групата съдържа `<path>` (геометрия на реброто) и `<polygon>` (връх на стрелката).

2.2.3. Извличане на координатите от входния *SVG* файл

Разчитат се само координатите, необходими за построяване в *TeX*:

- ***Places* (позиции)**: `cx, cy` от `<ellipse>` и `x, y` на етикета от `<text>`.
- ***Transitions* (преходи)**: координати на **двата** `<polygon>` елемента и `x, y` на етикета от `<text>`.
- ***Edges* (ребра)**: от `<path>` се взима `d` и се парсва в последователност от точки `(x_k,y_k)` чрез командите `M` и `L`. Върхът на стрелката се разчита от `<polygon points="...">` като точки `(x_j,y_j)`.

Накрая се извлича и **глобалната трансформация** (напр. `translate` и `scale`), която се прилага върху всички координати.

2.2.4. Глобални трансформации и нормализация на координатите

Координатите в *SVG* често са подложени на обща трансформация, която трябва да се приложи, за да се получи коректен мащаб и разположение в *TeX*. Типичният формат е:

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

Извличат се:

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

За всяка точка (x, y) се прилага:

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

След това се използва общ множител за намаляване:

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

Нормализацията към `picture` е нужна, защото в *TeX* произходът е долу вляво, а в *SVG* оста y расте надолу, и защото е удобно координатите да са в рамките на платното. След изчисляване на *bounding box* $(X_{\min}, X_{\max}, Y_{\min}, Y_{\max})$, всяка точка се преобразува:

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

Размерите на платното се задават така, че да поберат цялата фигура:

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

2.2.5. Нормализация на размери и етикети

След нормализацията се уеднаквяват размерите и се коригират етикетите, тъй като мащабът и шрифтовете в *SVG/Graphviz* и *TeX* се различават.

Фиксиране на радиуса на позициите.
Радиусът на позициите се фиксира чрез постоянен диаметър на окръжността:

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

Корекция на етикетите на позициите.
Етикетът се измества по посоката от центъра към оригиналната си позиция:

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

където k е `placeLabelOffsetFactor`.

Корекция на етикетите на преходите.
Етикетът на преход се измества нагоре с константа:

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

2.2.6. Изрязване на ребрата до границите на елементите

За да не навлизат ребрата в позициите/преходите, крайните точки се изрязват до границата на съответния елемент. Реброто е $(p_0, p_1, \ldots, p_{n-1})$, като се коригират само p_0 (по посока към p_1) и p_{n-1} (по посока от p_{n-2}).

Изрязване на крайни точки на ребра при позиции.
При позиция (окръжност с радиус r_{TeX}) се използва:

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

Изрязване на крайните точки на реброто при преходи.
Преходът се приближава с правоъгълник $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$. Посоката се определя от съседната точка Q чрез:

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

и критерий:

$$\text{хоризонтален, ако } |\Delta x| \geq |\Delta y|, \qquad \text{вертикален, ако } |\Delta y| > |\Delta x|. \tag{10}$$

След това точката се поставя върху съответната страна:

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

Запазване на коректния край на стрелката.
След изрязването последният сегмент се генерира като `\vector`, така че върхът на стрелката да съвпадне с новата крайна точка.

2.2.7. Псевдокод на алгоритъма

В по-общ вид стъпките могат да се синтезират, както следва:

```
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
```

След изпълнението на тези стъпки се генерира *TeX* фрагмент, готов за директно вмъкване в научен документ съгласно приетата конвенция за изчертаване на ОМ.

Глава 3. *OnlineGN*: Симулатор за обобщеномрежови модели

В настоящата глава се представя разработеният в рамките на дисертационния труд нов симулатор за ОМ – *OnlineGN*. Разглеждат се изискванията към системата, архитектурните решения и използвания технологичен стек. Системата е самостоятелен софтуерен компонент, като е предвидено в бъдещи етапи да се разшири съвместимостта чрез работа с файлови формати, генерирани от други симулатори. Основните функционалности са демонстрирани във видео демонстрация [42]. Изходният код на приложението е публикуван в [41].

3.1. Изисквания и цели на системата

Разделът дефинира функционални и нефункционални изисквания към уеб базираното решение, които служат като критерии за проектиране и оценка на *OnlineGN*.

3.1.1. Функционални изисквания

OnlineGN визуализира ОМ в двумерно пространство (позиции, преходи и връзки) и извършва симулация чрез прилагане на алгоритъм Б (виж **1.1.4.**), като се спазва коректният ред на обработка на преходи/състояния/ядра и се поддържат промени на характеристики, разделяне и сливане на ядра. Реализирани са два режима на изпълнение: режим за проследяване (стъпка по стъпка) и непрекъснато изпълнение до крайно състояние (или критерий за спиране).

Системата позволява рестарт на симулацията, интерактивна промяна на координати на елементи и връзки (вкл. точки по връзка), както и редакция на характеристики на позиции и преходи с отражение върху симулацията. Осигурява се детайлна информация за ядрата във всяка стъпка и регистър на операциите в рамките на една времева стъпка, който описва реда на обработка според приоритети и капацитети.

Поддържа се зареждане/запис на мрежи в *JSON* формат, съхранение и извличане от база данни, генериране на хипервръзка за достъп до съхранена мрежа, генериране на представяне в *TeX* формат, добавяне на потребителски функции на *JavaScript* и динамична промяна на скоростта на симулацията. Реализирано е и автоматично изтриване на мрежи, които не са били достъпвани за предварително дефиниран период.

3.1.2. Нефункционални изисквания

Нефункционалните изисквания обхващат производителност, надеждност, сигурност, използваемост и мащабируемост. Зададени са прагове за време за отговор до 2 секунди при леко натоварване и време за възстановяване след отказ до 30 минути, като изборът е обоснован спрямо утвърдени практики и критерии за отзивчивост и наличност [43]. Допълнително, системата не събира лични данни и е съобразена с изискванията на *GDPR*.

3.1.3. Ограничения и предпоставки на средата

Системата е внедрена с безплатните облачни услуги на ***Render*** (за сървъра) и ***Atlas*** (за базата данни), поради което се съобразява с ограничените ресурси, предоставени от тези услуги.

3.2. Архитектура на системата

Разделът описва архитектурната организация на *OnlineGN* и взаимодействието между основните компоненти, съответстващи на изпълнението на **Задача 4**.

3.2.1. Обща архитектура и сценарии на взаимодействие

OnlineGN е реализирана като трислойна архитектура: слой за данни (БД), сървърен слой (*API*/услуги) и клиентски слой (потребителско приложение), като клиентът поема основната част от бизнес логиката, а сървърът предоставя спомагателни услуги. Базата данни съхранява мрежи и състояния и отговаря за консистентността на данните; сървърът е посредник между клиента и БД и изпълнява поддържащи задачи (напр. периодично изтриване на остарели записи); клиентът реализира *UI/UX*, визуализацията, симулацията и интерактивните модификации.

Описани са типични сценарии: зареждане на мрежа от БД и стартиране на симулация, съхранение на мрежа с връщане на хипервръзка, както и периодично изтриване на неизползвани мрежи от страна на сървъра.

3.2.2. Физическа/*Deployment* архитектура и облачна инфраструктура

Внедряването е реализирано чрез ***Render*** и ***MongoDB Atlas***. Инстанцията на сървъра се поддържа в инфраструктурата на *Render* и доставя клиентското приложение към браузъра, като комуникацията клиент–сървър е по *HTTPS*. Инстанцията на базата данни се поддържа в *Atlas*; сървърът комуникира с БД по *HTTPS* и се удостоверява с потребителско име и парола [51, 52].

3.2.3. Технологичен стек и външни зависимости

Базата данни е *MongoDB* (версия 8.0), като данните се съхраняват като документи с *JSON*-подобна структура и физическо представяне *BSON*.

Сървърната част е реализирана с *Node.js* и *Express.js* за *REST API*, с унифицирана маршрутизация, валидация и централизирана обработка на грешки. Използваната *Node.js* версия е 22.x (*LTS*), избрана заради предвидим жизнен цикъл и дългосрочни актуализации по сигурност; подходът поддържа ниска цена на миграция към следваща *LTS* версия. За сървъра се използват външни библиотеки като `body-parser`, `cors`, `cron`, `express`, `is-svg` и `mongoose`, а за разработване – `nodemon`.

Клиентското приложение е разработено с *Angular* (версия 16) върху *TypeScript* и *SCSS*, като се следват добри практики за поддръжка и бъдещо преминаване към след-

ваща *LTS* версия [53, 54]. Освен основните пакети на *Angular*, са използвани *Angular Material* и *CDK* за *UI* компоненти, `ngx-toastr` за известия, както и `d3` и `d3-graphviz` за интерактивна визуализация, анимации и рендиране на *Graphviz DOT* в браузъра [55].

3.3. Дизайн и модули на приложението

В контекста на **Задача 4** се представя модулната организация на *OnlineGN*. Описват се основните функционални компоненти (съхранение, визуализация, редакция, симулация, импорт/експорт и *API*) и взаимодействието между тях. Модулният дизайн цели ясна разделимост на отговорностите (*separation of concerns*), улеснена поддръжка и мащабируемост.

3.3.1. Модул за обработка на данните (*GenNet Persistence*)

GenNet Persistence реализира устойчивостта на данните за мрежите и осигурява основни операции за работа със записи. Включени са подмодули за:

- ***Loading*** – зареждане на мрежа по уникален идентификатор, като при достъп се актуализира информацията за последно използване;
- ***Saving*** – записване/обновяване на данните за мрежа, така че да бъдат налични за последващо прочитане;
- ***Expiration*** – автоматично изтриване на мрежи, които не са били достъпвани в последните 90 дни.

3.3.2. *JSON* формат за описание на обобщени мрежи

В дисертационния труд се описва разработен *JSON* формат за унифицирано, машинночетимо представяне на ОМ, пригодено за директна интерпретация от симулатора. Форматът на данните е дефиниран от интерфейса *GenNetExportModel*. Един документ описва структурата на мрежата, началната конфигурация на ядрата, параметри на изпълнение, потребителски дефинирани функции и предварително генерирана визуализация. Форматът е организиран в четири основни секции: `genNetRaw`, `settings`, `code` и `svg`.

Коренова структура.
Кореновият обект съдържа:

- ***`genNetRaw`*** – декларативно описание на ОМ;
- ***`settings`*** – настройки на симулационната среда;
- ***`code`*** – потребителски *JavaScript* код;
- ***`svg`*** – предварително генерирана визуализация на мрежата.

Така се осигурява ясно разграничение между структурни данни и поведенческа спецификация.

Секция `genNetRaw`.
Секцията `genNetRaw` съдържа основните данни за мрежата и включва колекциите `places`, `transitions` и `tokens`.

- ***`places`*** – масив от обекти от тип *Place*, чрез които се задават идентификатор, име, функция за трансформиране на характеристиките, функция за сливане, приоритет и капацитет на позицията;
- ***`transitions`*** – масив от обекти от тип *Transition*, чрез които се задават идентификатор, име, функция за разделяне, приоритет и връзките между входните и изходните позиции чрез обекти от тип *TransitionItem*;
- ***`tokens`*** – масив от обекти от тип *Token*, дефиниращ началната конфигурация на симулацията чрез идентификатор, име, текуща позиция, приоритет и начални характеристики.

В дисертационния труд са дадени примерни *JSON* представяния на обекти от тип *Place*, *Transition* и *Token*.

Секция *`settings`*.
Секцията `settings` съдържа параметри, влияещи на изпълнението, без да променят структурния модел. В разглежданата реализация е включен обектът `simulationSettings`, чрез който се задава продължителността на една симулационна стъпка посредством `stepDurationMS`.

Секция *`code`*.
Секцията `code` съдържа низ с потребителски *JavaScript* код, използван при симулацията. За да бъде кодът достъпен в контекста на симулатора, на най-външно ниво се дефинира обектът ***`this.globalObj`***, който служи като пространство от имена за потребителските дефиниции.

В ***`this.globalObj`*** се съдържат:

- ***`tokens`*** – референция към текущата колекция от ядра;
- ***`functions`*** – функции, извиквани чрез имената си от полетата в ***`genNetRaw`***;
- ***`globals`*** – допълнителни стойности, използвани като спомагателен контекст при изпълнение на симулацията.

Така форматът позволява декларативно задаване на мрежата, а конкретното поведение и условията при симулацията се дефинират централизирано и се свързват чрез имената на функциите. В дисертационния труд е представен и примерен *JSON* модел на ОМ с две позиции, един преход и едно ядро.

Секция `svg`.
Секцията `svg` съдържа *SVG* низ за визуализация на мрежата. Ако такъв не е предоставен, програмата генерира *SVG* визуализация чрез алгоритъма за автоматично изчертаване, описан в **2.1.**.

3.3.3. Модул за графична визуализация (*SVG/Graphviz*)

Разгледан е модулът за визуализация на ОМ в клиентското приложение: зареждане на готов *SVG* или генериране на *SVG* от *Graphviz* низ, рисуване/прерисуване на елементи, анимации при промени, обработка на потребителски събития и запазване на текущото състояние като *SVG*.

3.3.4. Модул за симулация на движението на ядрата

Описан е модулът *Simulation*, който реализира изпълнението на симулацията по правилата на ОМ и синхронизира логическите промени с визуализацията: единичен ход (*Single move*), пълна симулация (*Full simulation*) и рестартиране (*Reset*) чрез подмодул за актуализация на визуализацията (*Update visualization*).

3.3.5. *API* и скриптов интерфейс

Сървърът предоставя две крайни точки за достъп от клиент.

- `GET /api/genNets/:id` – извлича мрежа по идентификатор и връща `status code 200` при успех или `status code 404`, ако мрежата не бъде намерена
- при намерена мрежа се връща обект с полета `id`, `content` и `lastFetchedAt`
- `POST /api/genNets` – записва подадена мрежа в базата от данни
- в заявката се подава поле `content`, а при успешно записване се връща `status code 201` и `id`
- при неуспешно записване се връща `status code 400`

Тук в дисертационния труд са дадени примерни заявки и *JSON* отговори от *API*.

3.3.6. Модул за импортиране и експортиране на мрежи

Модулът за импортиране и експортиране осигурява зареждане и запазване на данните на мрежата в различни формати, съобразно интерфейса, описан в **т. 3.3.5.** Поддържа се експортиране в *JSON* файл, *TeX* формат и *JSON* формат в база от данни, както и импортиране от файл и от съхранени *JSON* данни в базата. Функционално модулът се разделя на два подмодула: *Import* (зареждане) и *Export* (запис).

Подмодул *Import*.

Import включва:

- ***JSON*** – прочитане и валидация на входни данни в *JSON* формат спрямо интерфейса от **т. 3.3.5.**, като при успешна валидация се създават необходимите обекти за визуализация; използва се и при зареждане от сървър, тъй като данните пристигат в *JSON*;
- ***File*** – обработка на файлове, предоставени от потребителя, като при валиден *JSON* съдържанието се предава към *JSON* подмодула.

Подмодул *Export*.

Export включва:

- ***TeX*** – преобразуване на текущата визуализация на мрежата в *TeX* формат за интеграция в научни документи;
- ***JSON*** – генериране на *JSON* представяне на текущо заредената мрежа (вкл. данни за ядра и визуализация) според формата от **т. 3.3.5.**; използва се и при съхранение в базата от данни;
- ***File*** – създаване на файл за сваляне от потребителя, като данните се генерират от *JSON* подмодула и се записват във файл.

3.3.7. Модул за визуализация на динамични процеси и анимация

За визуализация на движението на ядрата в симулатора се използва библиотеката *d3*, като всяко ядро се представя чрез *SVG* група (графичен символ и текст). Преместването се реализира по траектории, зададени чрез *SVG path* елементи, като анимацията се изпълнява чрез интерполация: изчислява се дължината на пътя (`getTotalLength()`), дефинира се преход с `attrTween` и параметър $t \in [0, 1]$, а текущата позиция се получава с `getPointAtLength(t * l)` и се прилага чрез трансформация *translate*. След приключване на прехода елементът се премахва от *DOM*, което освобождава ресурси и отразява преминаването на ядрото по реброто.

3.4. Алгоритми и имплементационни детайли

Описват се основните алгоритми и ключови имплементационни решения, използвани при реализирането на *OnlineGN*.

3.4.1. Разработен алгоритъм 1: автоматично изчертаване на ОМ

Алгоритъм 1 е реализиран програмно в системата *OnlineGN* като отделна имплементация, която автоматизира изчертаването на ОМ на база на структурните данни на мрежата. Реализацията генерира координати и графично представяне, което се използва директно от модула за визуализация.

3.4.2. Разработен алгоритъм 2: преобразуване на *SVG* към *TeX*

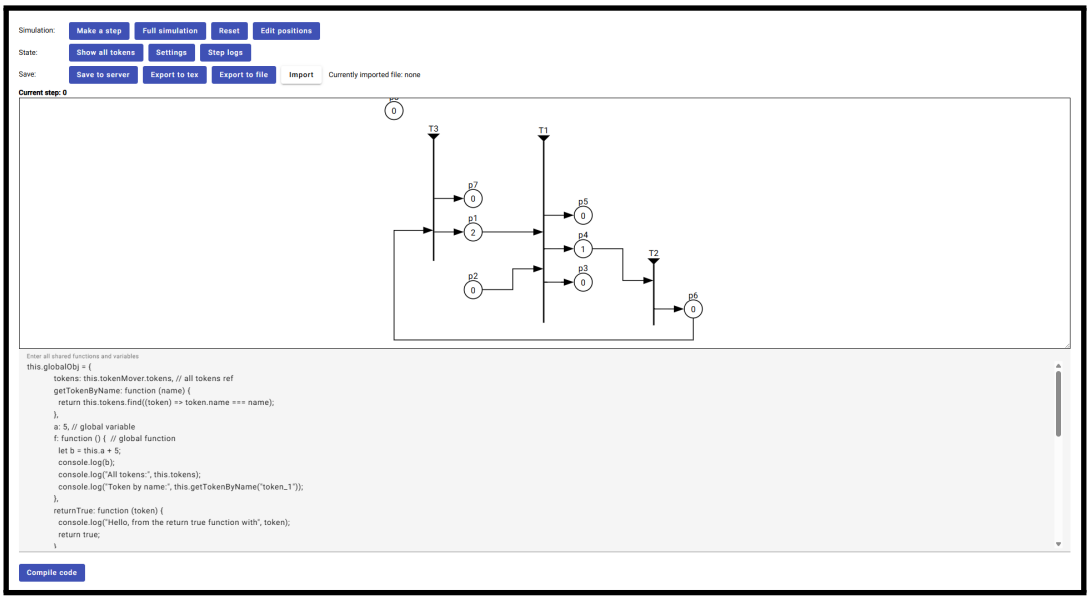
Алгоритъм 2 е имплементиран в *OnlineGN* за автоматично преобразуване на текущата *SVG* визуализация в *TeX* представяне. Реализацията трансформира основните графични примитиви и текстови елементи, като осигурява експортиране в *TeX* формат за включване в научни документи.

3.5. Интерфейс и потребителско изживяване

Главата представя реализацията на **Задача 5** и **Задача 6**, като разглежда дизайна на потребителския интерфейс и основните принципи на потребителското изживяване в *OnlineGN*. Акцентът е върху навигацията, визуалните компоненти и начина, по който интерфейсът подпомага интерактивната работа със симулатора, включително визуализация, персонализация и редакция на параметри на ОМ.

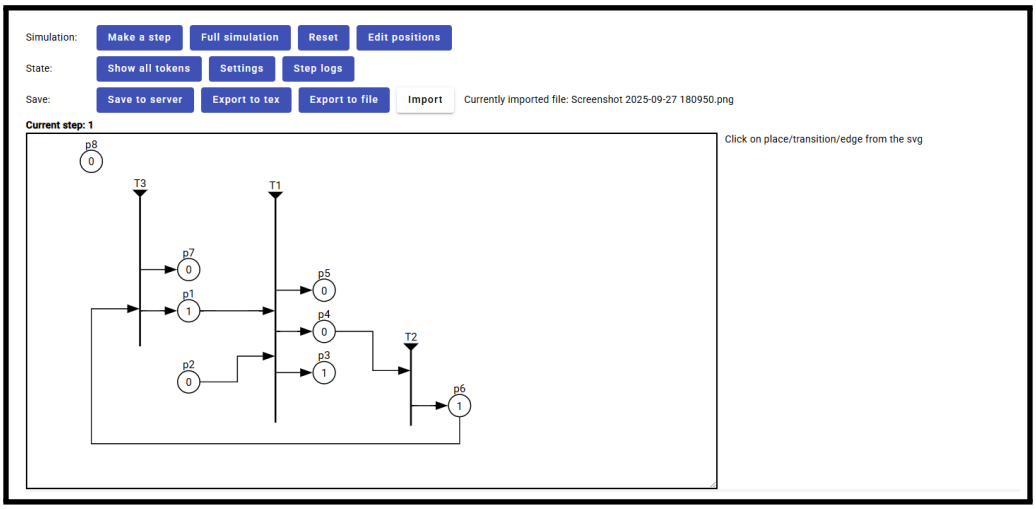
3.5.1. Навигация и визуални компоненти на уеб клиента

Клиентското приложение обединява основните функционалности в едно основно работно поле: визуализация на мрежата, управление на симулацията и достъп до операции за зареждане/запис и експортиране. Интерфейсът осигурява директен преглед на текущото състояние на мрежата и стъпката на симулацията, както и удобен достъп до основните действия чрез функционални бутони и диалози.



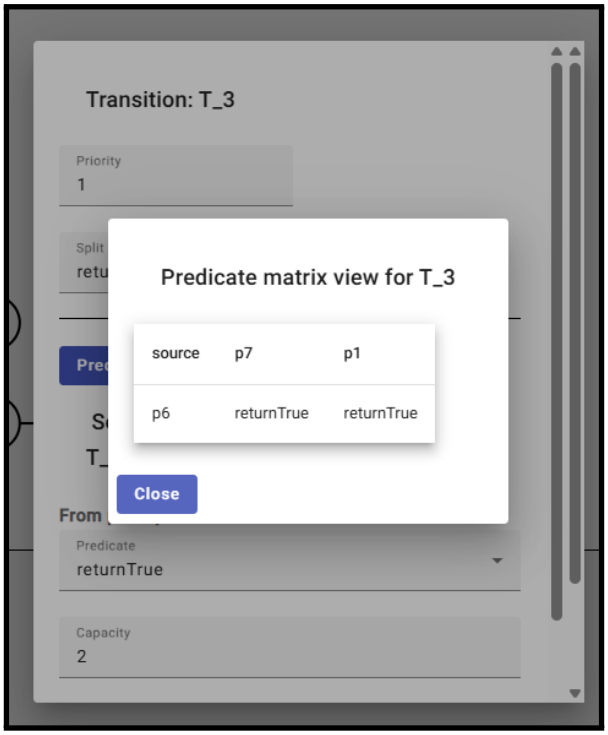
Фигура 16 *– Основно работно поле и контроли.*

За реализацията на **Задача 6** е въведен режим за редакция на елементите на мрежата, чрез който се извършва избор на позиция/преход/ребро и промяна на координати и параметри. Това позволява интерактивна корекция на визуализацията и настройване на мрежовите обекти директно в работното поле.



Фигура 22 *– Режим за промяна на координати („Edit positions“).*

Предикатите и правилата за преместване се визуализират чрез специализирани компоненти за редакция и преглед на преходите, включително представяне на предикатната матрица, която отразява условията за преминаване между входни и изходни позиции на преход.



Фигура 44 *– Предикатна матрица на конкретен преход.*

3.5.2. Помощна система, документация и локализация

Предвидена е помощна система, която допълва уеб клиента чрез вградена документация и контекстна помощ. Осигурени са и възможности за локализация на интерфейса, което улеснява използването на симулатора в образователен и изследователски контекст.

Глава 4. Приложения на обобщените мрежи и симулатора *OnlineGN*

В настоящата глава се разглеждат четири нови ОМ модела на процеси, които до този момент не са били описвани чрез апарата на ОМ. Всеки от разработените модели е симулиран и верифициран чрез програмната среда *OnlineGN*. Изложението и проведените симулации в тези четири раздела представляват решението на **Задача 8** и **Задача 9**.

4.1. Симулация на крайни автомати чрез обобщени мрежи

В дисертационния труд се разглежда как краен автомат може да бъде описан и симулиран с помощта на ОМ. Изгражда се връзката между състоянията и преходите на автомата и съответните елементи на ОМ, така че моделът да се превърне в работеща схема. Разгледан е и конкретен пример на краен автомат, за който се изгражда съответната ОМ и се реализира реална симулация с *OnlineGN*. При разработването и анализа се следва подходът, предложен в [56].

В [57] е разгледан начин за представяне на автомати чрез ОМ, но основният акцент е върху теоретичното доказателство за съществуването; тук се дава конструкция, директно приложима за симулация в софтуерни среди за ОМ, с конкретни структурни ограничения и оптимизации.

4.1.1. Основни понятия за крайни автомати

Крайни азбуки и езици.
Използват се стандартните понятия *крайна азбука* Σ, *дума* $\alpha \in \Sigma^*$ и *език* $L \subseteq \Sigma^*$, като празната дума се означава с ε.

Детерминиран краен автомат (ДКА).
ДКА е петорка $\mathcal{A} = (Q, \Sigma, \delta, q_{start}, F)$ [58, 59], където $\delta : Q \times \Sigma \to Q$ е тотална. Думата α се приема тогава и само тогава, когато $\delta^*(q_{start}, \alpha) \in F$.

Недетерминиран краен автомат (НКА).
НКА е петорка $\mathcal{N} = (Q, \Sigma, \Delta, q_{start}, F)$ [58, 60], където $\Delta : Q \times \Sigma \to \mathcal{P}(Q)$. Думата α се приема тогава и само тогава, когато $\Delta^*(\{q_{start}\}, \alpha) \cap F \neq \varnothing$.

Сравнение между ДКА и НКА.
ДКА следва единствена изчислителна пътека, а НКА допуска разклонени (паралелни) пътеки; двата модела разпознават един и същи клас езици (теорема на Рабин–Скот [61], *powerset construction*).

4.1.2. Представяне на недетерминирани крайни автомати чрез обобщени мрежи

ОМ предоставят гъвкава рамка за симулиране на паралелни процеси [8, 9], което ги прави подходящи за моделиране на начина, по който думите се приемат от НКА. ОМ се конструира така, че да обхваща всички възможни паралелни изчислителни пътеки: всяко *състояние* q_i се съпоставя с преход Z_i, а всяко *ребро* $q_i \to q_j$ се представя като *позиция* (изходна за Z_i и входна за Z_j). Въвеждат се специална начална позиция p_{start} и краен преход Z_{final}, който реализира напускане на мрежата при приемане.

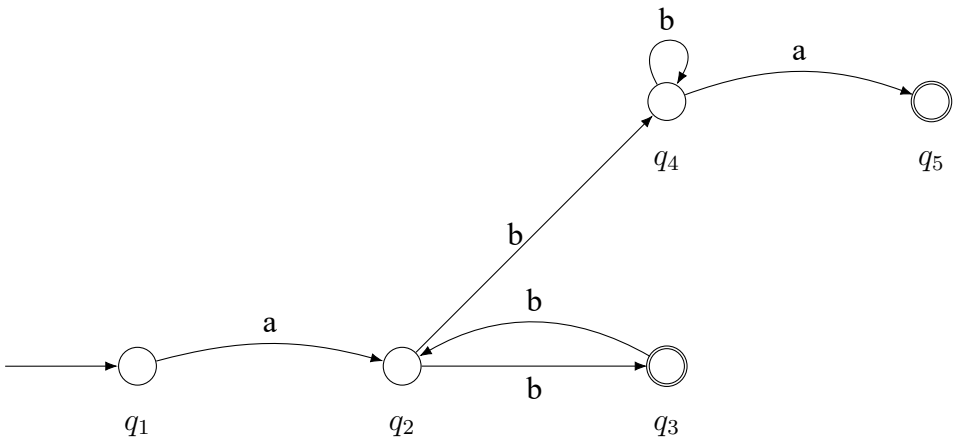
Ядра и характеристики.
Активните обекти са *ядра*, като всяко ядро съответства на възможна изчислителна пътека в НКА. Като характеристика се носи остатъкът от входната дума (`word`); при преминаване през позиция първият символ се премахва (с изключение на началното преместване от p_{start}). При разклоняване в НКА ядрото се дублира в няколко копия с еднакъв текущ остатък на думата.

Индексирани матрици и предикати.
Конструкцията се формализира чрез *индексирани матрици* и предикати от вида $S(x, y)$ (за съвместимост между следващия входен символ и допустим преход в НКА), както и предикат E (празна характеристична „дума“) като условие за преминаване през Z_{final}.

Твърдение. Всеки НКА може да бъде представен чрез съответна ОМ (вж. [8, 9]).

4.1.3. Подробен пример за недетерминиран краен автомат, представен като обобщена мрежа

Разглежда се примерен НКА с две крайни състояния (q_3 и q_5), който е недетерминиран поради две ребра с етикет *b* от едно и също състояние. На базата на този автомат се построява съответната ОМ с по един преход за всяко състояние и допълнителен краен преход Z_{final}.



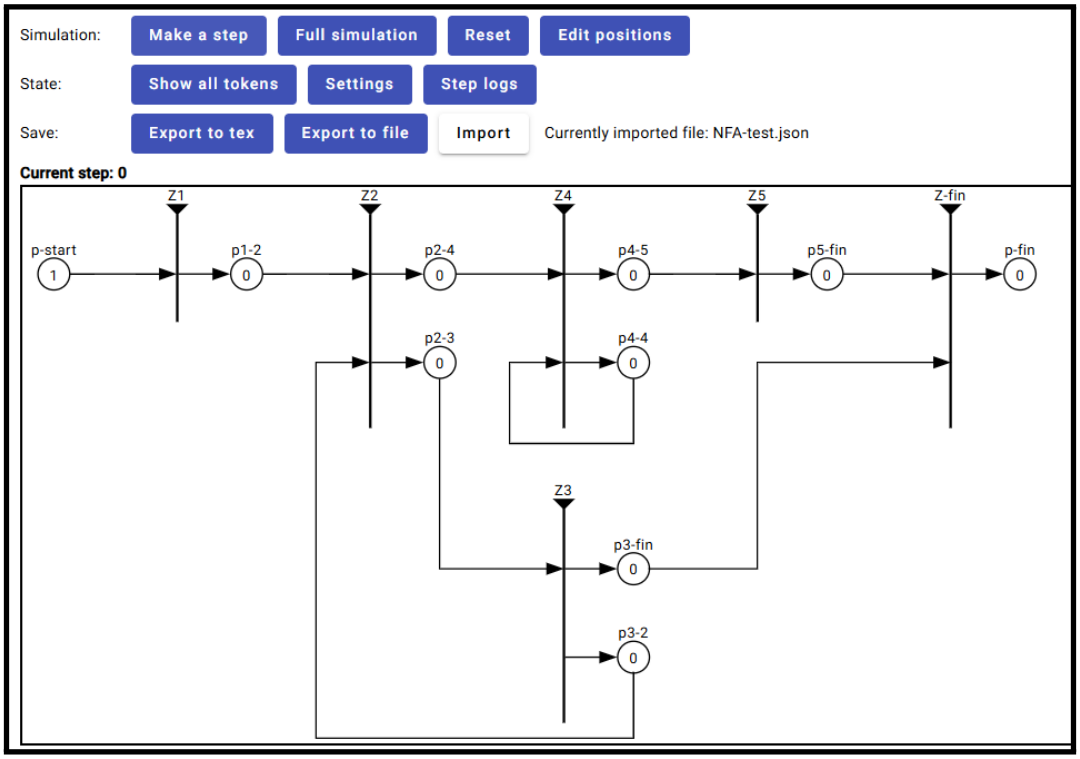
Фигура 52 *– Примерен НКА с две крайни състояния: q_3 и q_5.*

4.1.4. Симулация на множество недетерминирани крайни автомати чрез една обобщена мрежа

Показано е, че крайно множество от НКА може да бъде представено чрез една ОМ, чрез обединяване на ОМ, съответстващи на всеки НКА от множеството, като се използва свойството $E_1 \cup E_2$ да е ОМ при ОМ E_1 и E_2 [8]. Това позволява едновременно проверяване на входни низове през множество НКА чрез една единствена ОМ.

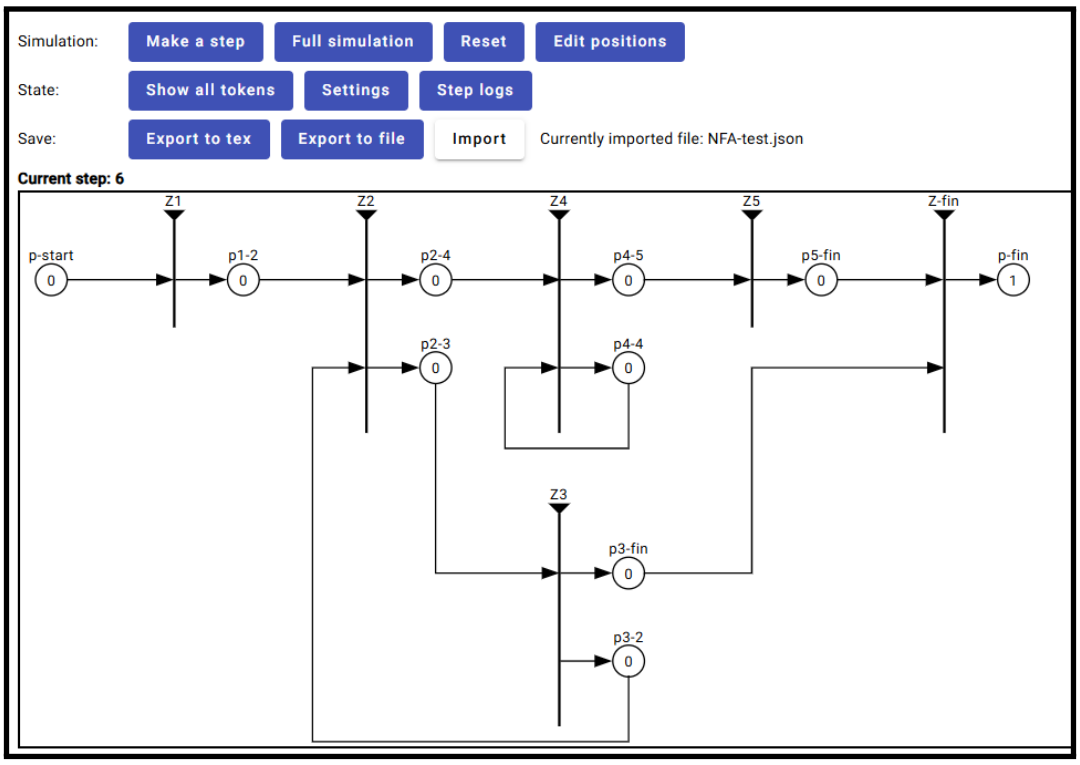
4.1.5. Симулация на един недетерминиран краен автомат с помощта на *OnlineGN*

Демонстрирана е реална симулация в *OnlineGN* на построената ОМ, която разпознава думата `abbba`. В симулацията се използва леко изменена нотация (тъй като долни индекси не се поддържат), напр. q_{i-j} се изписва като `qi-j`. ОМ започва с едно ядро в `p-start`, носещо характеристика `abbba`; при последователните стъпки се визуализира движението на ядрата и развитието на характеристиката (прочитане на символите). При приемане ядрото достига финалната позиция, а характеристиката става празен низ.

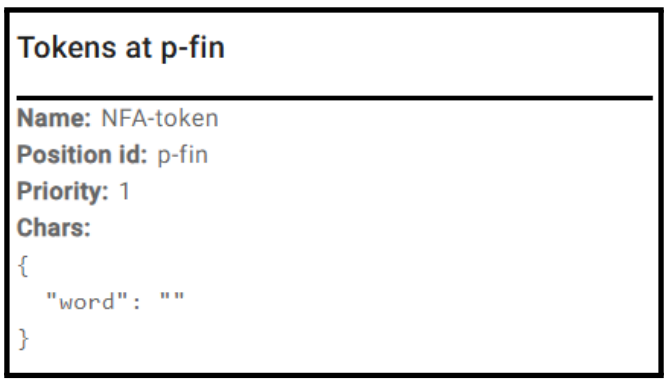


Фигура 54 *– Начална конфигурация на ОМ с едно ядро в началната позиция (`p-start`).*

Фигура 55 – *Ядро в* `p-start` *с характеристика* `abbba`.



Фигура 58 – *Ядрото е достигнало крайната позиция* `p-fin`.



Фигура 59 – *Характеристиката на ядрото става празен низ при приемане.*

4.1.6. Заключение

В дисертационния труд се разглежда как ОМ могат да бъдат използвани за симулация на ДКА и НКА чрез съпоставяне на състоянията с преходи и на ребрата с позиции, като се улавят паралелните изчислителни пътища. Архитектурата, базирана на ядра и характеристики, позволява проследима симулация, като *OnlineGN* се използва за демонстрация на изпълнението стъпка по стъпка и за ясно показване на приемането (финална позиция и празна характеристика).

4.2. Паралелизация на задачата за Ханойските кули чрез обобщени мрежи

Разглежда се *паралелна* версия на класическата задача за Ханойските кули, при която в една стъпка се допуска едновременно преместване на повече от един диск. Задачата е формализирана чрез апарата на ОМ, а симулационната реализация в *OnlineGN* демонстрира изпълнимостта на модела и ефекта от паралелизма при експоненциални комбинаторни процеси.

4.2.1. Описание на задачата за Ханойските кули

Класическата задача за Ханойските кули [63] се състои от три пръта и n пръстена с различни размери, подредени първоначално върху първия прът; по-голям пръстен не може да се поставя върху по-малък (**Фигура 60**). Целта е прехвърляне на всички пръстени от първия към третия прът при спазване на правилото. При преместване на точно един пръстен на стъпка са необходими минимум $2^n - 1$ стъпки [64, 65]. Настоящата глава описва процеса чрез ОМ.

4.2.2. Паралелизация на Ханойските кули: Концептуален преглед

Класическите ограничения са: **(i)** на всяка стъпка се премества само един пръстен и **(ii)** по-голям пръстен не се поставя върху по-малък. В паралелния вариант **(i)** се разхлабва: на една *паралелна стъпка* може *едновременно* да се повдигне пакет от най-много k пръстена (от върховете на произволна комбинация от пръти), след което да се *поставят* един по един по приоритет (първо най-големият, последен най-малкият). Цикълът повдигане–поставяне се брои за една паралелна стъпка.

4.2.3. Конфигурация на системата

- **Пръти.**
  Три пръта R_1 (източник), R_2 (помощен), R_3 (целеви), всеки със стек от пръстени със строго намаляващ размер отдолу нагоре.

- **Пръстени.**
  n пръстена, етикетирани от 1 (най-малък) до n (най-голям).

- **Въздушен буфер.**
  След повдигането избраните до k пръстена са „във въздуха“, като вътрешният ред на пръстените, идващи от един и същ прът, се запазва.

4.2.4. Дефиниция на паралелна стъпка

Всяка стъпка има две фази:

1. **Фаза на повдигане.** Избират се до k пръстена от върховете на прътите и се отстраняват *едновременно*.
2. **Фаза на поставяне.** Поставяне по *правило за приоритет*: (1) освобождаване в строго низходящ ред на размера; (2) поставяне върху произволен прът без нарушаване на локалния ред по размер.

Минималният брой паралелни стъпки се означава с $H(n, k)$.

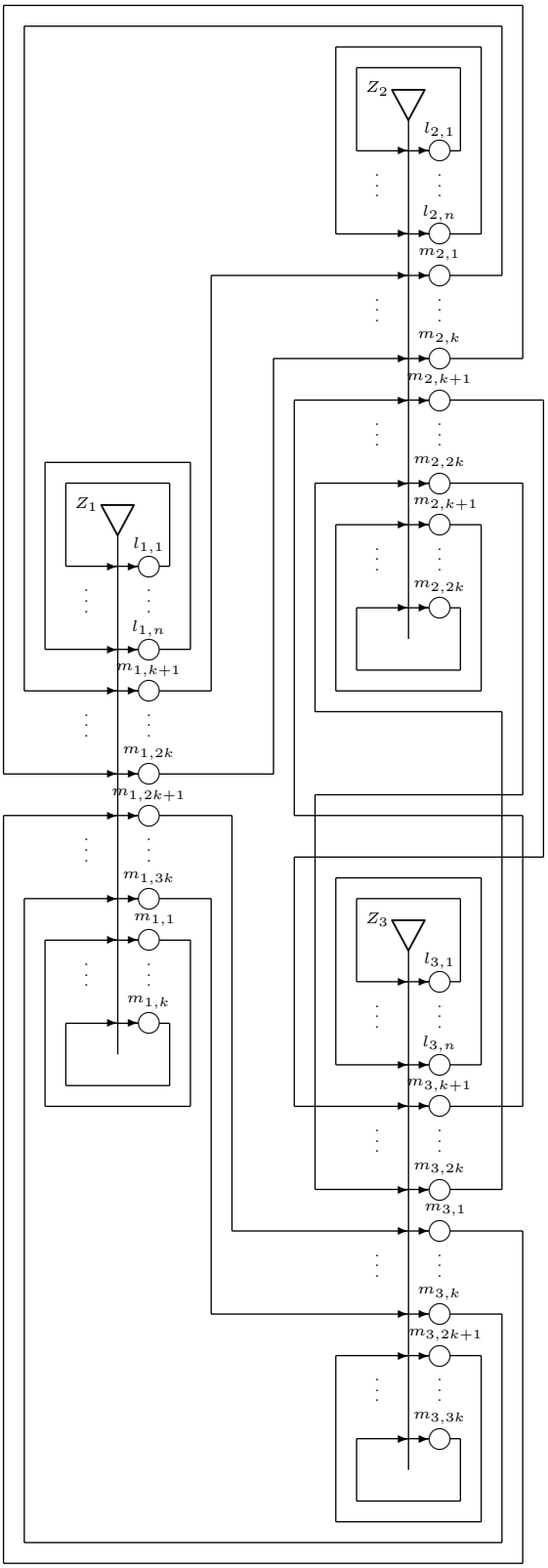
Теорема 1. $H(n, k) = 2^{\lceil \frac{n}{k} \rceil} - 1$.

Доказателството групира n пръстена в $M = \lceil n/k \rceil$ блока с размер най-много k, които се третират като „суперпръстени“, и свежда паралелния модел до класическата рекурсия върху M елемента (горна и долна граница съвпадат).

Наблюдение (Оптималност на блоковите премествания в паралелния вариант на Ханойските кули).
Оптимална стратегия винаги прехвърля точно k последователни пръстена от един прът към един друг прът; всеки блок се държи като „супердиск“, поради което задачата се свежда до класическите Ханойски кули върху $\lceil n/k \rceil$ супердискове.

4.2.5. Обобщеномрежови модел за паралелните Ханойски кули

Въвежда се ОМ модел за симулация на k-паралелното правило. ОМ съдържа n ρ-ядра („пръстени“) и k χ-ядра („ръце“). χ-ядрата позволяват в една паралелна стъпка да се повдигнат едновременно най-много k ρ-ядра и да се поставят по приоритетното правило. Мрежата е изградена така, че да реализира оптималната блокова стратегия и да е изпълнима в *OnlineGN*.



Фигура 61 *– ОМ за Ханойските кули с n ρ-ядра и k χ-ядра.*

Ядра и премествания.

Използват се конвенциите:

- ρ-ядро: пръстен; начална характеристика – размерът.
- χ-ядро: „ръка“; характеристика – индекс на „ръката“.

Разграничават се *ход от пъзела* и *мрежов ход*. Всеки ход от пъзела се реализира с **два мрежови хода**: (i) позициониране на ρ- и χ-ядра към изходните позиции на прехода на изходния път; (ii) прехвърляне към входно–изходните позиции на прехода на целевия път. Така ОМ симулира прехвърлянето на ρ-ядра между пръти чрез последователност от мрежови ходове.

Преходи и позиции.

ОМ съдържа **три прехода** Z_1, Z_2, Z_3 (по един за прът). За ρ-ядрата се използват n **входно–изходни позиции** $l_{i,j}$ ($i \in \{1,2,3\}, j = 1, \ldots, n$); ρ-ядро в $l_{i,j}$ означава „пръстен с размер j е върху прът i“. За χ-ядрата се използват **позиции за задържане** $m_{i,j}$ ($j \in [1, 3k]$), които кодират текущия/следващия прът за вземане и прехвърляне, в зависимост от четността на мрежовата стъпка. Общият вид на ОМ е показан на **Фигура 61**.

Индексирани матрици.

Предикатите върху индексираните матрици на ОМ са:

- $U_{x,y}$: „ρ-ядро се премества от x към y в тази стъпка на пъзела“;
- $W_{x,y,i,j}$: избор на χ-ядро за преместване на ρ-ядро и гарантиране на коректна позиция при блоковия режим;
- P_i: „ρ-ядро има характеристична тежест i“ (коректно позициониране след хода);
- S_x: разполагане на χ-ядрата в изходни позиции според предстоящото преместване.

Индексираните матрици за нечетни и четни мрежови ходове са дадени на **Фигури 63 и 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	$\cdots$	$U_{a,c}$

Фигура 63 *– Индексирана матрица на Z_X за нечетните мрежови ходове (начало на ход от задачата).*

	$l_{a,1}$	$\cdots$	$l_{a,n}$	$m_{a,(a-1)*k+1}$	$\cdots$	$m_{a,a*k}$	$m_{a,(b-1)*k+1}$	$\cdots$	$m_{a,b*k}$	$m_{a,(c-1)*k+1}$	$\cdots$	$m_{a,c*k}$
$l_{a,1}$	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

Фигура 64 *– Индексирана матрица на Z_X за четните мрежови ходове (в края на ход от задачата).*

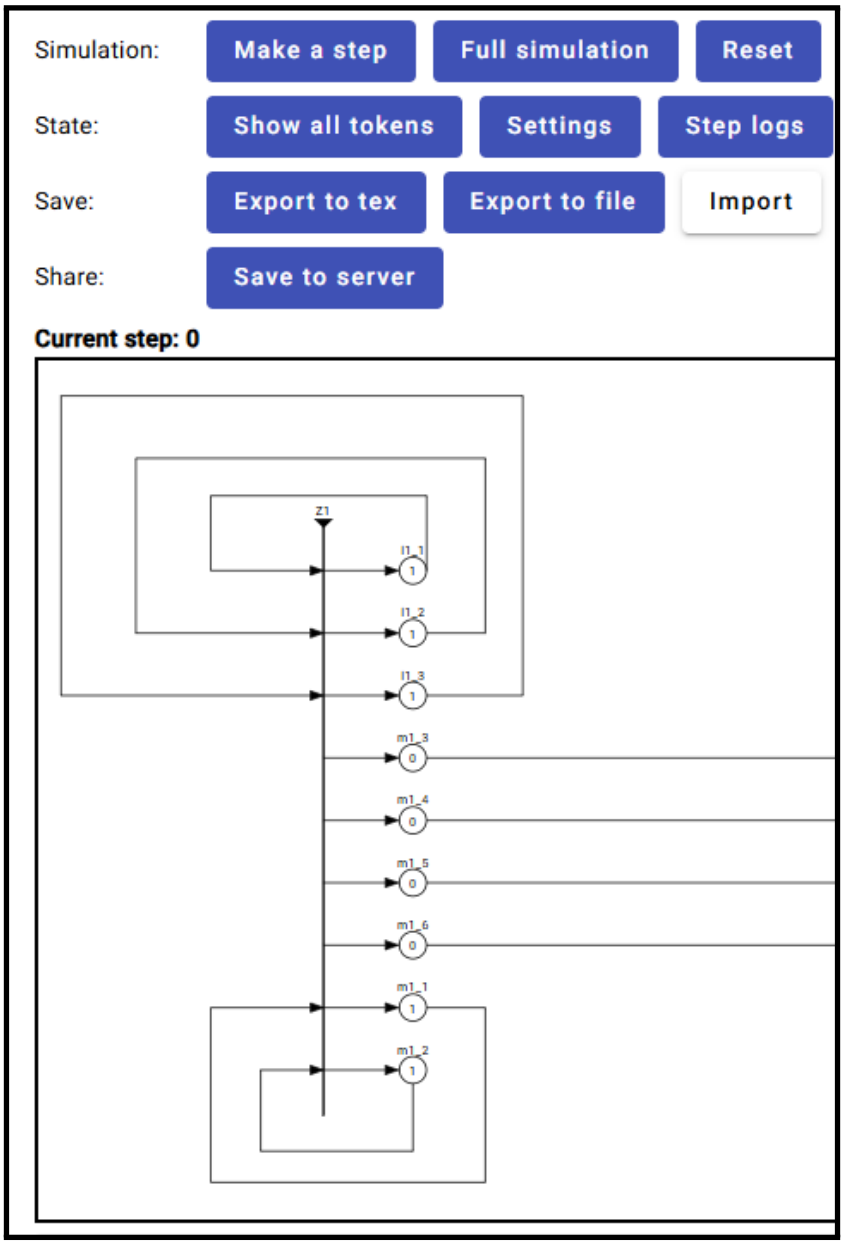
Теорема 2. *ОМ, описваща пъзела на Ханой с n ρ-ядра и k χ-ядра, прехвърля ρ-ядрата от първия към третия прът за $2^{\lceil \frac{n}{k} \rceil} - 1$ хода от пъзела.*

Доказателството е пряко приложение на **Теорема 1** при $N = n$, $K = k$, чрез групиране на ρ-ядрата в блокове с размер най-много k.

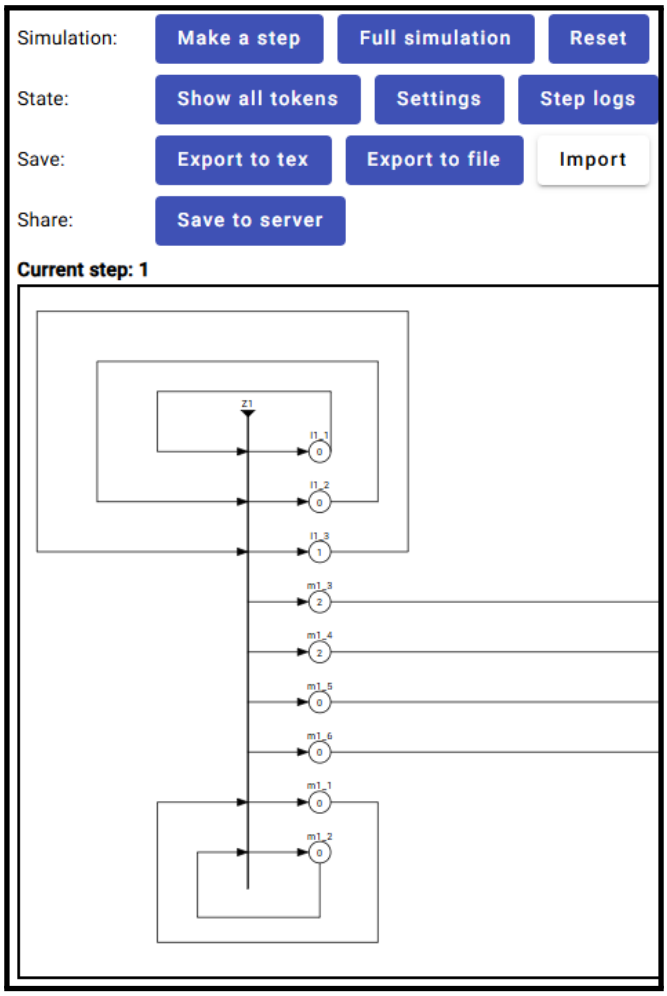
4.2.6. Симулации с *OnlineGN*

Разработеният ОМ модел е симулиран с *OnlineGN*. Поради големината на мрежата се използват екранни снимки, които показват активните сегменти. За пример $n = 3$,

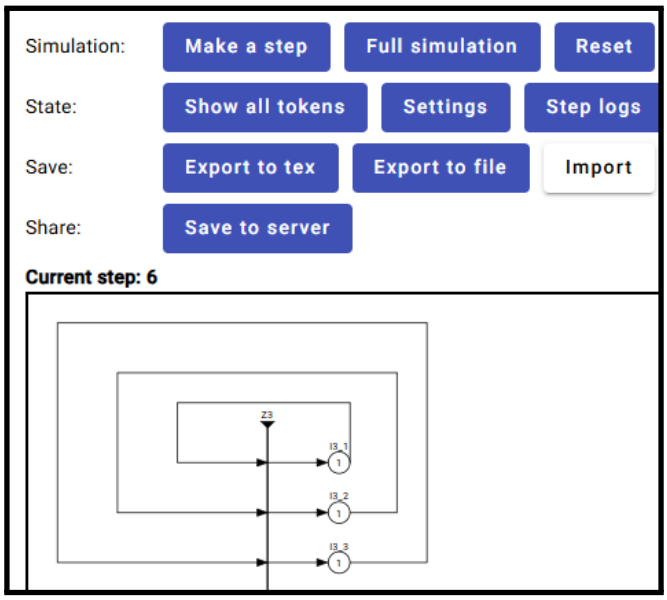
$k = 2$ началната конфигурация е със ρ- и χ-ядра при Z_1 (**Фигура 65**), след което се наблюдава междинно позициониране към m-позиции, насочени към Z_2 (**Фигура 66**), и крайна конфигурация с всички ρ-ядра в Z_3 (**Фигура 67**). За всички тествани двойки (n, k) отчетеният брой стъпки съвпада с $2^{\lceil n/k \rceil} - 1$.



Фигура 65 *– ОМ за Ханойските кули с 3 ρ-ядра и 2 χ-ядра в OnlineGN.*



Фигура 66 *– ρ- и χ-ядрата са позиционирани за преместване към другия преход (път).*



Фигура 67 *– Крайна конфигурация: всички ρ-ядра заемат изходните позиции на Z_3 (Интерпретация на Ханойските кули: трети път).*

4.2.7. Други проучвания за паралелно решаване на задачи

В този раздел на дисертационния труд са разгледани други паралелни решения на експоненциални задачи.

4.2.8. Изводи от резултатите

Разработената ОМ е директен, изпълним модел на k-паралелното правило на Ханойските кули. При изпълнение в ***OnlineGN*** симулацията генерира очаквания брой ходове $2^{\lceil \frac{n}{k} \rceil} - 1$ за всички тествани n и k, което потвърждава коректността и числовата достоверност на модела. Увеличаването на k съкращава критичния път, но печалбите намаляват поради цената на синхронизация (повдигане и поставяне при глобално ограничение за реда), т.е. процесът е *синхронизационно ограничен*.

4.3. Модел с обобщена мрежа на производството на газови и полимерни продукти в нефтопреработвателна рафинерия

В дисертационния труд се разглежда ОМ модел на производството на газово гориво, втечнен петролен газ (ВПГ, *LPG*), пропилен и полипропилен, както и неговата програмна симулация чрез *OnlineGN*. Изграждането на модела и детайлите по формализацията са представени в [78].

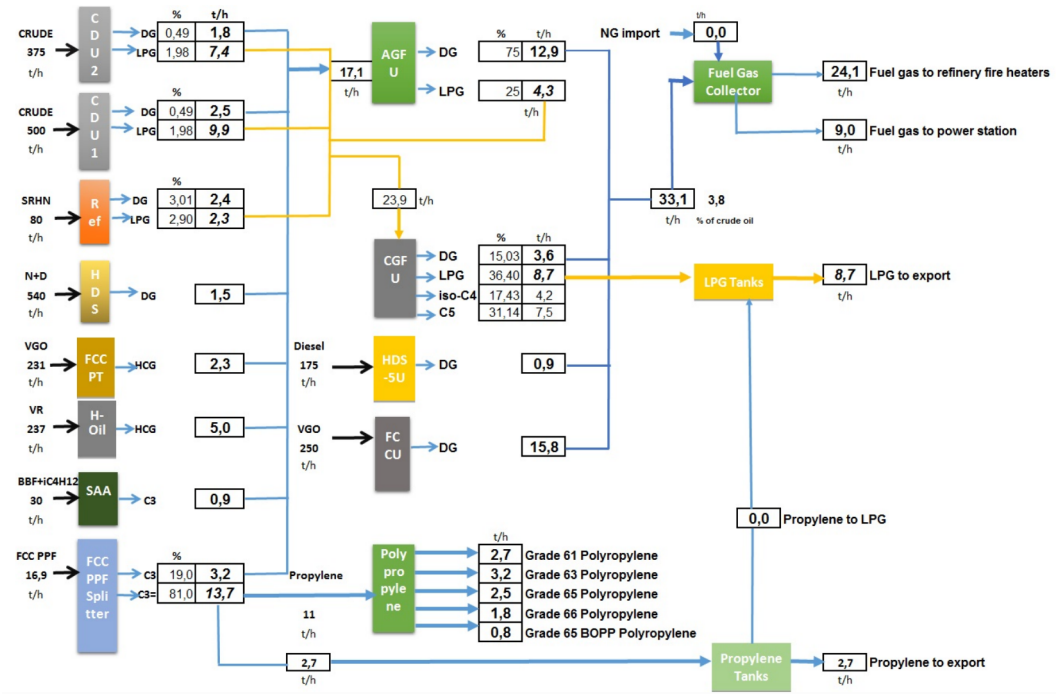
4.3.1. Увод в производството на продукти в нефтопреработвателна рафинерия

Моделирането на процесите на производство на рафинирани продукти в нефтопреработвателна рафинерия подпомага производственото планиране и анализа на ефективността. В литературата се срещат частични модели на отделни аспекти на проблема [79, 80, 81], докато при ОМ се задават причинно–следствените зависимости чрез предикатите на преходите и характеристиките на ядрата.

В предишни изследвания [82, 83] е показано, че производството на автомобилен бензин [82] и дизелово гориво [83] може да бъде моделирано чрез ОМ; настоящата глава допълва тази линия с производството на газово гориво, ВПГ, пропилен и полипропилен и верификация чрез *OnlineGN*.

4.3.2. Технологична схема за производство на газово гориво, втечнен петролен газ, пропилен и полипропилен в нефтопреработвателна рафинерия

Технологичната производствена верига за газово гориво, ВПГ, пропилен и полипропилен е представена на **Фигура 69** (в дисертационния труд се използва като основа за формализацията в ОМ).

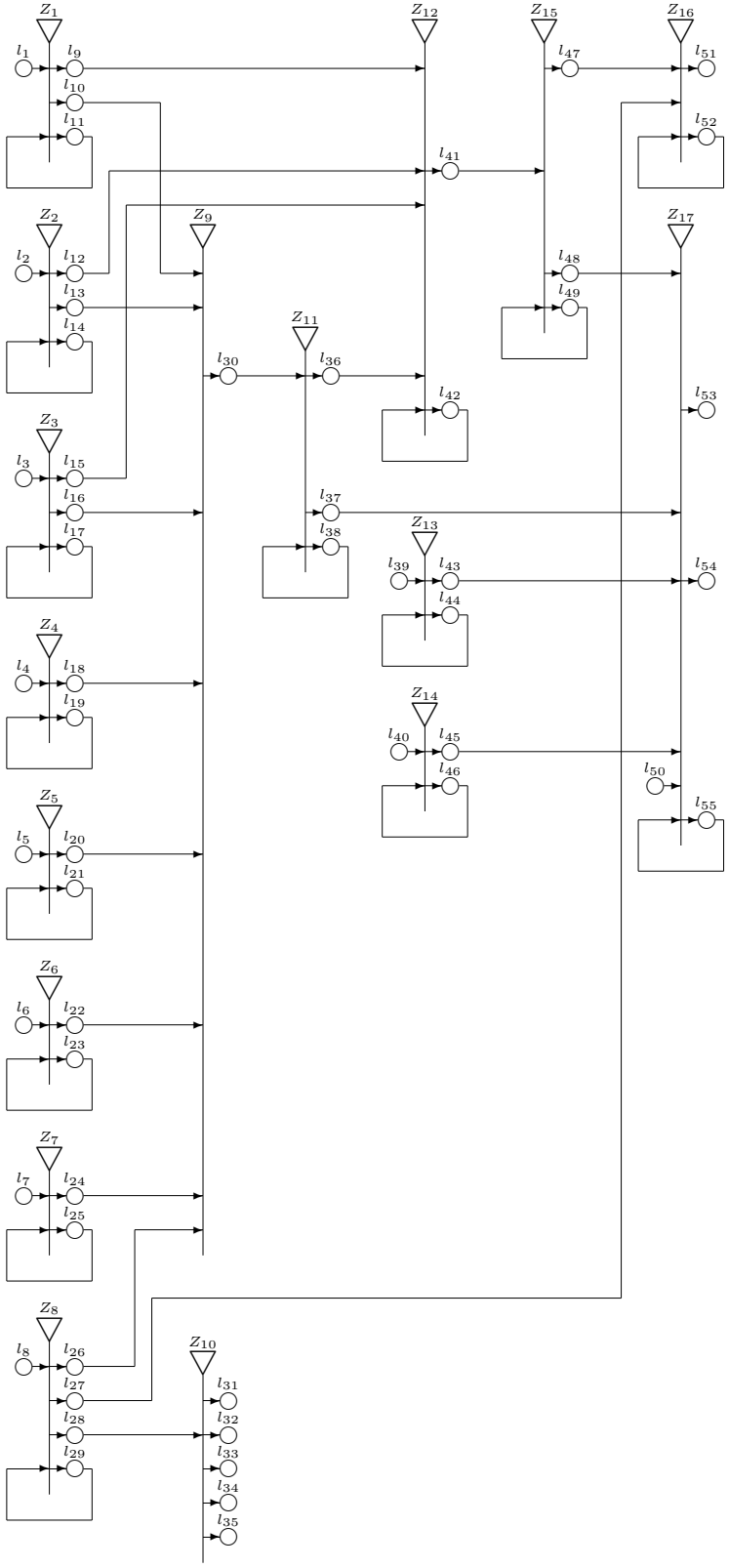


Фигура 69 *– Технологична схема за производство на газово гориво, ВПГ, пропилен и полипропилен в нефтопреработвателна рафинерия.*

Производството на петролни газови продукти зависи от произхода на суровия нефт и експлоатационните режими в процесните инсталации. В зависимост от пазарните изисквания течният пропилен може да се експортира като полимеризационен пропилен или да се използва като компонент на ВПГ; от инсталацията за полипропилен се произвеждат различни марки полипропиленови продукти.

4.3.3. Основни резултати: обобщеномрежов модел

В дисертационния труд се разглежда ОМ моделът (**Фигура 70**), който описва разглежданата технологична верига чрез позиции, преходи и ядра с характеристики. Моделът съдържа 17 прехода, 55 позиции и 47 типа ядра, като са включени: потоци суровини/междинни фракции (σ_i), процесни звена (ρ_i) и продукти (α, β, γ, δ, ε), съгласно [78].



Фигура 70 *– ОМ модел.*

Логиката на протичане се задава чрез индексирани матрици с предикати за всеки преход (напр. условия от типа „има заявка за продукт ...“), които определят премества-

нето на ядрата от входните към изходните позиции и промяната на характеристиките им при симулация.

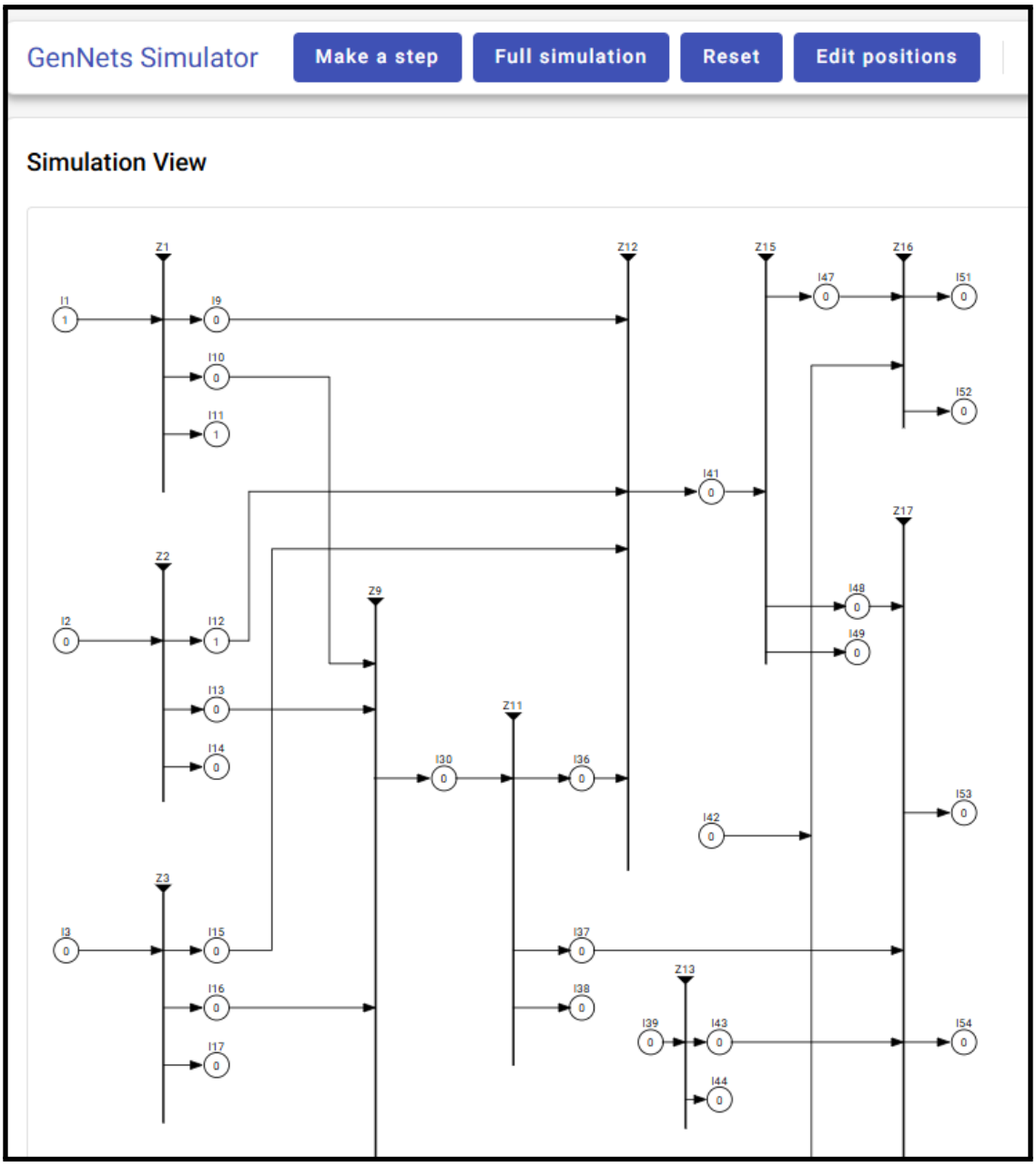
4.3.4. Бележки по резултатите и обобщеномрежов модел

Производството на газово гориво, ВПГ, пропилен и полипропилен представлява сложен паралелен процес с множество технологични инсталации и ограничения, който може да бъде представен чрез ОМ.

4.3.5. Симулация с *OnlineGN*

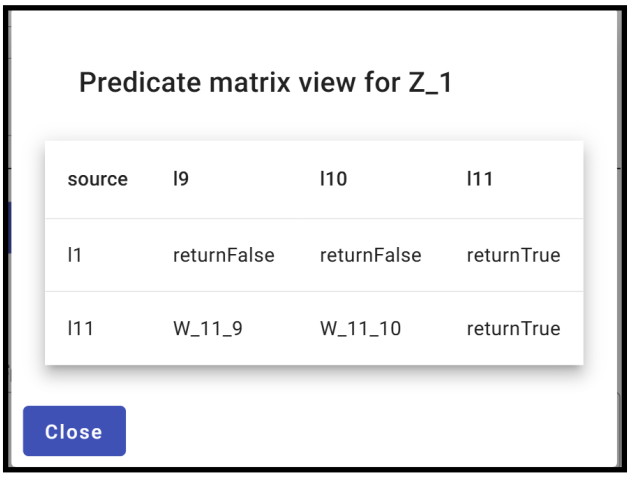
За да проверим на практика дали предложеният ОМ модел описва коректно процеса на производство на горивен газ, ВПГ, пропилен и полипропилен в нефтопреработвателна рафинерия, го имплементирахме и симулирахме в *OnlineGN*. Така получаваме не само визуализация на структурата (позиции, преходи и връзки), но и възможност да проследим как „се движат“ ядрата и как се променят техните характеристики при преминаване през отделните звена.

Поради големината на модела визуалното му представяне е разделено на три части: горна, средна и долна. Тук е изложена само горната в **Фигура 71**.



Фигура 71 *– Горна част на ОМ модела в OnlineGN.*

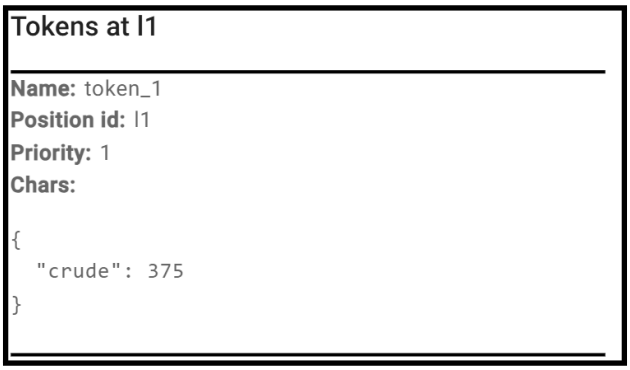
Следващата важна стъпка е логиката на преходите: за всеки преход са дефинирани индексирани матрици, в които са заложени предикатите от теоретичното описание. Именно тези предикати „решават“ дали дадено ядро може да премине от входните към изходните позиции на прехода. На **Фигура 74** е показан пример с индексирана матрица за преход Z_1, който в модела съответства на *CDU-2*.



Фигура 74 *– Индексирана матрица с предикатите за преход* Z_1 *(CDU-2).*

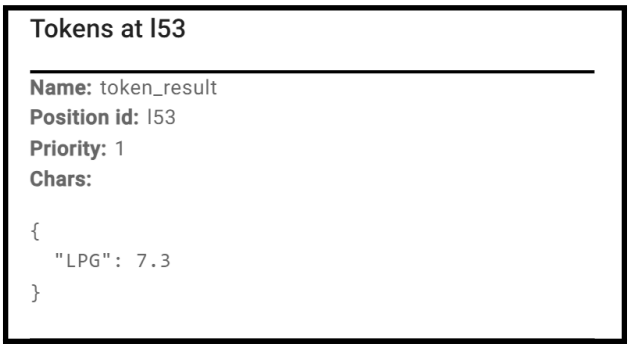
В този конкретен пример предикатите отразяват условия от типа „има заявка за продукт ВПГ от *CDU-2*“ и „има заявка за продукт горивен газ от *CDU-2*“. С други думи, преходът Z_1 пропуска ядра нататък само когато са налице съответните заявки/условия, описани в модела. Аналогично, за останалите преходи са въведени индексирани матрици, съответстващи на описаните предикати за конкретните процесни звена и потоци.

Началните характеристики на ядрата са настроени според сценария от статията. На **Фигура 75** е показано ядро във входната позиция l_1, което постъпва към Z_1 (*CDU-2*), заедно с неговата начална характеристика. Тази характеристика съдържа необходимите атрибути за симулацията (напр. вид заявка и количества), така че по време на изпълнение да може ясно да се проследи какво „носи“ ядрото и какво се променя по веригата.



Фигура 75 *– Ядро във входната позиция* l_1 *с начална характеристика (пример за вход към* Z_1*).*

След успешното изпълнение на симулацията наблюдаваните изходни характеристики съвпадат с очакваните теоретични резултати. Като конкретна илюстрация, на изходната позиция l_{53} (обозначена като *„LPG TANKS“*) се получава ядро, чиято характеристика съдържа количеството ВПГ, което моделът предвижда. Това е показано на **Фигура 76**.



Фигура 76 *– Ядро в изходната позиция l_{53} („LPG TANKS“) с характеристика след симулация.*

Обобщено, симулационните резултати потвърждават, че зададените предикати и начални характеристики работят коректно и че ОМ моделът описва адекватно разглеждания производствен процес. Това дава основание да приемем модела за валидиран спрямо теоретичния сценарий и използваем за последващи експерименти и анализи.

4.3.6. Изводи за модела и симулацията

Производството на газообразни продукти в нефтопреработвателна рафинерия (газово гориво, ВПГ и пропилен, включително като суровина за полипропилен) е сложен паралелен процес, при който коректното отразяване на причинно–следствените връзки е ключово; в ОМ това се задава естествено чрез предикатите на преходните условия. Настоящата глава представя ОМ модел и неговата програмна реализация и симулация чрез *OnlineGN*. Получаването на резултати, съответстващи на очакваното поведение на процеса, потвърждава коректността на модела и демонстрира приложимостта на ОМ при моделиране и симулация на подобни производствени вериги.

4.4. Модел с обобщена мрежа на производството на тежки нефтопродукти в нефтопреработвателна рафинерия

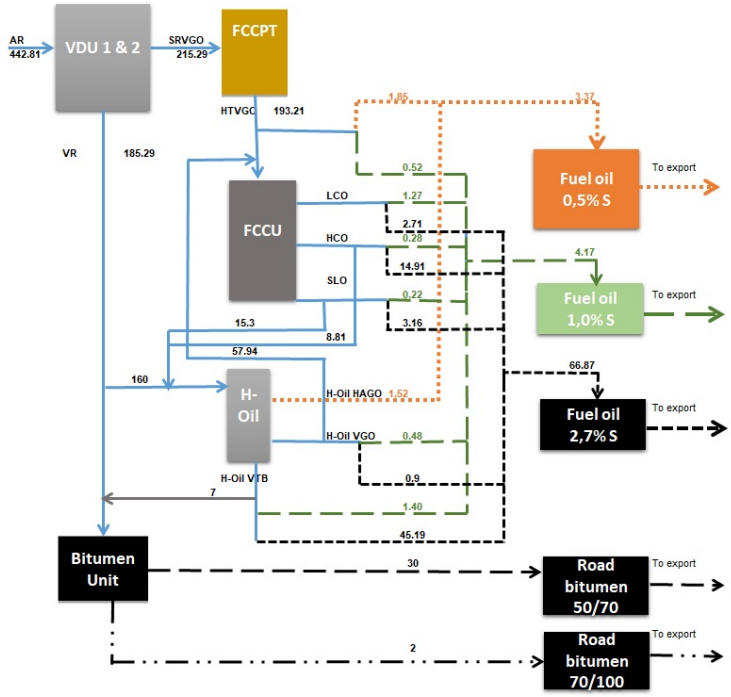
Тук се разглежда ОМ модел на производството на тежки нефтопродукти в нефтопреработвателна рафинерия и неговата симулация чрез *OnlineGN*. Изграждането на модела е описано подробно в [89].

4.4.1. Увод в обобщеномрежовите модели за нефтопродукти в нефтопреработвателна рафинерия

В дисертационния труд се разглежда, че паралелно протичащите производствени процеси в нефтопреработвателна рафинерия могат да бъдат моделирани и симулирани посредством ОМ [8]. В настоящата глава фокусът е върху преработката на тежките остатъчни фракции и производството на тежки нефтени продукти. Тежкият нефт представлява остатъчната нефтена фракция след атмосферна дестилация [91], с температура на кипене над 360 °C и относителна плътност над 0.933 ($API < 20$) [92]. Целта е процесът да се моделира с помощта на ОМ и моделът да бъде симулиран чрез *OnlineGN*.

4.4.2. Схема на преработка за производство на различни видове тежко гориво и битум за пътни настилки в нефтопреработвателна рафинерия, моделирана чрез обобщени мрежи

В дисертационния труд се разглежда производството на три вида гориво с различно съдържание на сяра (*Fuel oil* 0.5% *S*, *Fuel oil* 1.0% *S*, *Fuel oil* 2.5% *S*) и производство на пътен битум в *BU*, както е показано на **Фигура 77**. Спецификациите и изискванията за горивата са представени в [89]. Вакуумните фракции (*VR*, *SRVGO*) се получават във *VDU-1*/*VDU-2* от атмосферния остатък (*CDU*) [94].

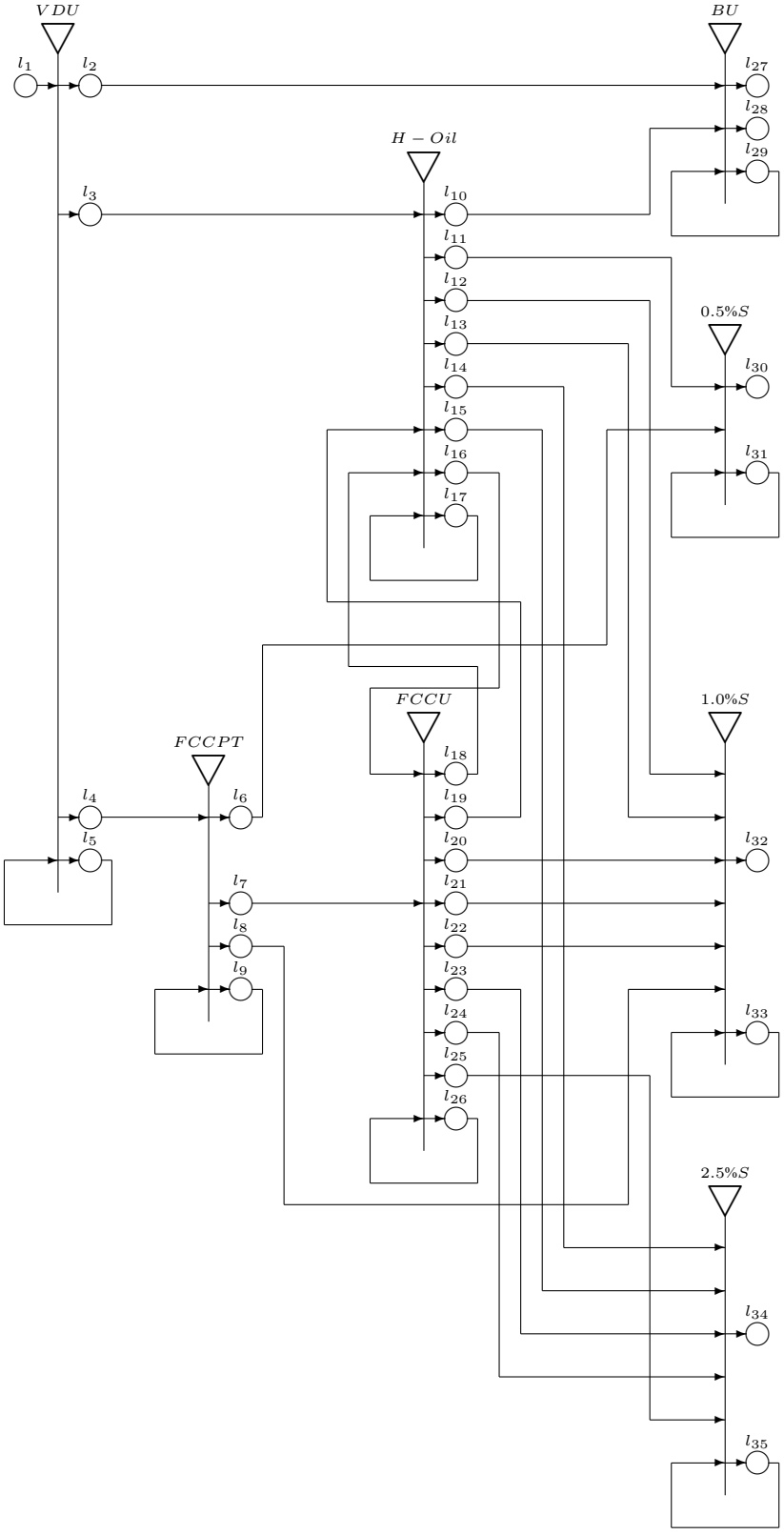


***Фигура 77** – Схема на преработка за производство на различни видове тежко гориво и битум за пътни настилки в нефтопреработвателна рафинерия, която ще бъде моделирана чрез обобщени мрежи (Числата в диаграмата съответстват на количеството потоци от тежък нефт в размер на t/h. Използвани са различни цветове за разграничаване на трите вида произведени горива: оранжев – за гориво с 0.5% S, зелен – за гориво с 1.0% S и черен – за гориво с 2.5% S).*

4.4.3. Резултати от моделирането на производството на тежки нефтени продукти в нефтопреработвателна рафинерия с помощта на обобщени мрежи

ОМ моделът (вж. **Фигура 78** в дисертационния труд) съдържа 8 прехода, 35 позиции и 8 типа ядра. Моделирани са технологичните звена *VDU*, *FCCPT*, *H-Oil*, *FCCU*, *BU*, както и изходите за 0.5% *S*, 1.0% *S* и 2.5% *S*. В началния момент ядра $\alpha_0, \beta_0, \gamma_0, \delta_0, \epsilon_0, \zeta_0, \eta_0, \theta_0$ са разположени съответно в позиции $l_1, l_9, l_{17}, l_{26}, l_{29}, l_{31}, l_{33}, l_{35}$ с характеристики, съответстващи на *AR*, *SRVGO*, *SRVR*, *FCC* суровина, битумна суровина и крайни продукти. Логиката на протичане се задава чрез индексирани матрици и предикати (напр. $W_{5,2}, W_{5,3}, W_{5,4}$ за заявки към *VDU*), като при изпълнение ядрата

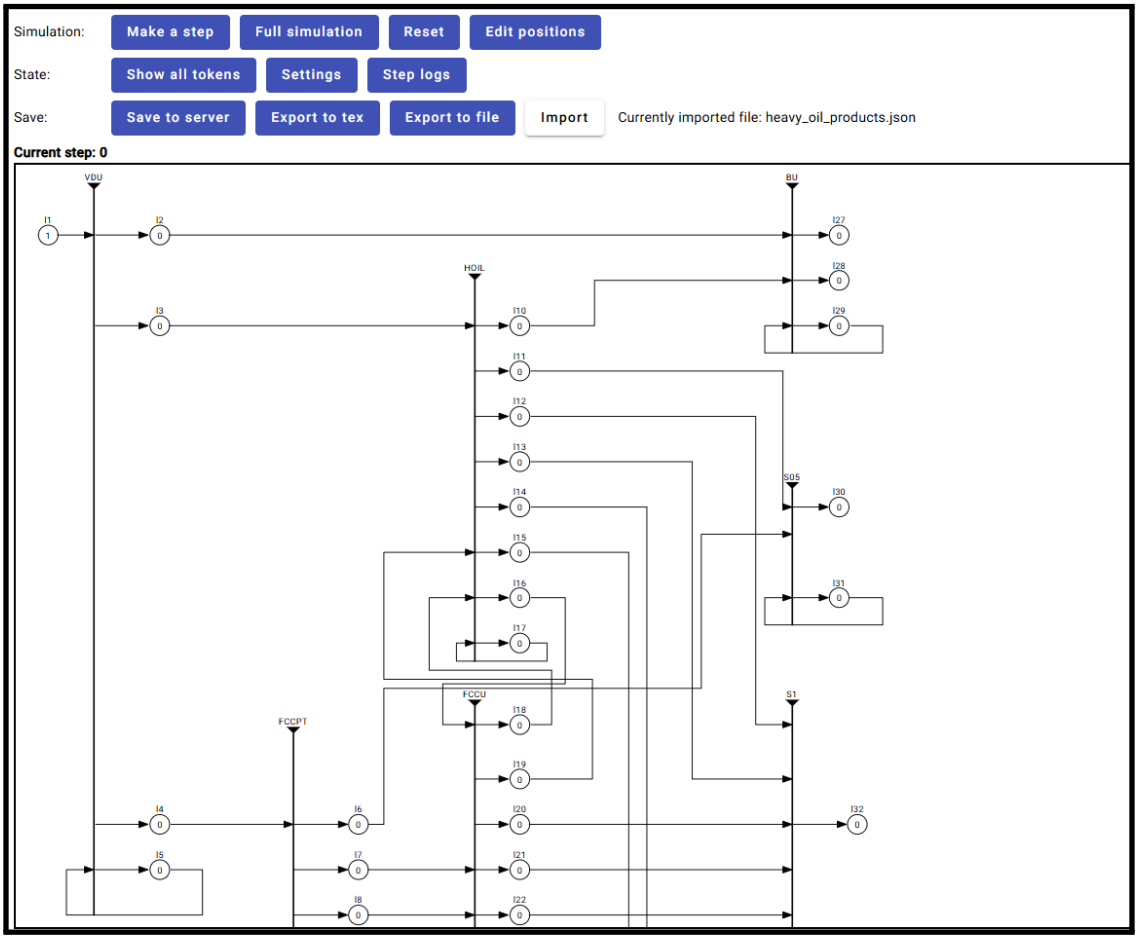
се обединяват и разделят в зависимост от истинностните стойности на предикатите и зададените количества $q_i \in [0, Q_i]$.



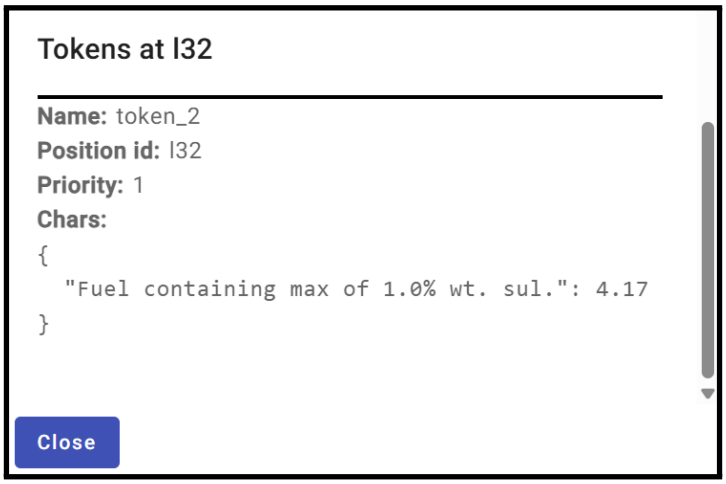
Фигура 78 *– Модел на ОМ за производството на тежки петролни продукти в рафинерията „ЛУКОЙЛ Нефтохим Бургас”.*

4.4.4. Симулиране на модела с *OnlineGN*

В дисертационния труд се разглежда реализация и изпълнение на ОМ модела в *OnlineGN* с цел верификация и извеждане на количествени резултати. **Фигура 79** показва схемата на ОМ, интегрирана в симулатора. За разглеждания сценарий началното ядро в позиция l_1 е атмосферен остатък (*AR*) с характеристична стойност 442.81, а заявките към следващите звена се задават чрез характеристичните функции (*code*) и индексираните матрици на преходите. Резултатите се извеждат от характеристиките на ядрата в изходните позиции (напр. l_{32} за 1.0 мас.% *S* и l_{34} за 2.5 мас.% *S*).



Фигура 79 *– Модел на обобщена мрежа (ОМ) на веригата за преработка на тежки нефтопродукти в рафинерията.*



Фигура 84 *– Полученото количество за мазут с максимум 1.0 мас.% сяра (позиция l_{32}).*

4.4.5. Резултати за модела

Както се вижда на **Фигура 77**, производството на трите класа тежко гориво (мазут) и класовете пътен битум представлява сложен паралелен процес, включващ *VDU*, *FCCPT*, *FCCU*, *H-Oil* и *BU*, в рамките на които се получават рафинирани тежки нефтени продукти и крайни продуктови потоци.

4.4.6. Изводи

Получените резултати показват, че процесите по производство на различни класове тежки нефтени продукти в рафинерия могат да бъдат успешно моделирани чрез ОМ, като паралелният характер и причинно–следствените зависимости се представят естествено. Проведената симулация чрез ***OnlineGN*** води до резултати, съответстващи на очакваното поведение на системата, което служи като верификация на коректността на модела и потвърждава приложимостта на ОМ при моделиране и симулация на производствени процеси в нефтопреработвателна рафинерия.

Заключение

В настоящия дисертационен труд бе постигната поставената основна цел – разработване на нов симулатор за ОМ, функциониращ в онлайн среда, който съществено улеснява и ускорява работата с ОМ. Реализираното решение подпомага ключови дейности като създаване на модели, последващо редактиране (включително промени по функции и визуализация) и ефективно споделяне на мрежи. При разработката са използвани съвременни и широко утвърдени технологии, съобразени с изискванията за надеждност, разширяемост и удобство при работа през уеб среда.

Наред със софтуерния резултат, в рамките на дисертацията са получени и нови приноси, свързани с теорията и приложението на ОМ. Разработени бяха нови алгоритми, насочени към подпомагане на моделирането и представянето на ОМ. Първо, предложен е алгоритъм за автоматично изчертаване на ОМ по нейното описание, който съкращава времето за получаване на коректна и прегледна визуализация и ограничава необходимостта от ръчни корекции. Второ, разработен е алгоритъм за експортиране на мрежата във формат *TeX*, така че след извършени редакции и визуални настройки в симулатора, моделът да може директно да бъде включван в научни публикации и други академични материали.

Допълнително бяха създадени нови модели на ОМ в различни приложни области, които демонстрират изразителната мощ на апарата и валидират възможностите на разработения симулатор. Сред тях са модели, свързани с представяне на крайни автомати, модел на математическа игра, както и модели за индустриални процеси, включително сценарии, свързани с производството на газови и нефтени продукти. Всички предложени модели са реализирани и изследвани чрез симулации с разработения в дисертацията симулатор, което осигурява практическа проверка на функционалността и приложимостта на системата.

Авторска справка

Справка за приносите в дисертационния труд

Научни приноси

В настоящия дисертационен труд са постигнати следните научни приноси:

1. **Алгоритми за автоматизация и визуализация на обобщени мрежи (ОМ):**
   - Разработен е алгоритъм за автоматизирано изчертаване на ОМ въз основа на тяхното абстрактно описание, елиминиращ необходимостта от ръчно задаване на координати.
   - Създаден е алгоритъм за преобразуване на векторни изображения (*SVG*) на ОМ в *TeX* формат чрез идентифициране на графични примитиви и трансформация на координати.
2. **Паралелно решаване на експоненциална задача (Ханойски кули):**
   - Формализиран е нов паралелен вариант на задачата за Ханойските кули, допускащ едновременно преместване на до k пръстена.
   - Доказана е теорема за минималния брой паралелни стъпки $H(n,k) = 2^{\lceil \frac{n}{k} \rceil} - 1$ и е реализиран конструктивен алгоритъм за решаването на задачата.
3. **Обобщеномрежово моделиране и симулация на паралелната експоненциална задача за Ханойски кули:**
   - Създаден е оригинален ОМ модел на паралелните Ханойски кули, използващ ρ-ядра за обектите и χ-ядра за ресурсите.
   - Доказана е приложимостта на формализма чрез симулации в среда *OnlineGN*, демонстриращи ефективно управление на конкурентни процеси и логически ограничения.

Научно–приложни приноси

В научно-приложно отношение са постигнати следните резултати:

1. **Проектиране, реализация и приложение на софтуерната система *OnlineGN*:**
   - Разработена е иновативна уеб базирана платформа за моделиране, симулация и визуализация на ОМ, предлагаща централизирана среда без необходимост от локална инсталация.
   - Разработен е удобен и интерактивен интерфейс за промяна на всеки компонент на ОМ в симулатора, както и за проследяване на детайли по изпълнението на мрежата.

- Внедрена е функционалност за експорт от *SVG* към *TeX*, което оптимизира подготовката на графични материали за академични публикации.

2. **Нов метод за представяне на крайни автомати чрез ОМ:**

   - Разработена е методика за описание на детерминирани и недетерминирани крайни автомати чрез ОМ, позволяваща улавяне на паралелни изчислителни пътища.

3. **Моделиране на съществуващи процеси в нефтопреработвателна рафинерия чрез ОМ:**

   - Разработени са ОМ модели на реални технологични процеси в нефтопреработвателна рафинерия, включително производството на газови, полимерни и тежки нефтени продукти.

4. **Верификация на новите ОМ модели чрез *OnlineGN*:**

   - Всички новосъздадени модели в дисертационния труд са верифицирани чрез симулации в *OnlineGN*, което верифицира едновременно коректността на моделите и възможностите на системата.

Публикации по дисертационния труд

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.

Библиография

[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] Атанасов, К. *Въведение в теорията на обобщените мрежи*. Бургас: Понтика Принт, 1992, pp. 7–10.

[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] Атанасов, К.; Сотирова, Е. *Обобщени мрежи*. София: Академично издателство „Проф. Марин Дринов", 2017. 172 с. ISBN 978-954-322-881-2.

[11] Орозова, Д. *Обобщеномрежови модели на интелигентни среди за обучение*. София: Академично издателство „Проф. Марин Дринов", 2011. 234 с. ISBN 978-954-322-481-4.

[12] Георгиев, П. Р. *Обобщено-мрежови модели на системи базирани на знания*. Дисертационен труд за придобиване на образователна и научна степен „доктор", СНС по ЕКТ, София, 1998. Научна специалност: 02.21.05 „Системи с изкуствен интелект". Научен ръководител: К. Атанасов.

[13] Стефанова-Павлова, М. М. *Моделиране на гъвкави автоматизирани производствени системи с обобщени мрежи*. Дисертационен труд за придобиване на образователна и научна степен „доктор", СНС по ЕКТ, София, 2001. Научна специалност: 01.01.12 „Информатика". Научен ръководител: К. Атанасов.

[14] Аладжов, Х. Ц. *Обобщено-мрежови модели на обучение и самоорганизация и приложението им за управление на двигателна активност*. Дисертационен труд за придобиване на образователна и научна степен „доктор", СНС по ЕКТ, София, 2002. Научна специалност: 02.21.05 „Системи с изкуствен интелект". Научен ръководител: К. Атанасов.

[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.

[33] 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] *Увод в OnlineGN - Симулатор за обобщени мрежи*. Video demonstration, published 12 Feb 2026. Available online: `https://youtu.be/sM_Cp-pt2TU?si=kl58ruSYlCRxdJir` (accessed 12 Feb 2026).

[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.