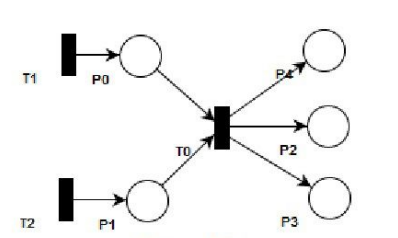
|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра теоретической и прикладной информатики | | |
| Расчётно-графическое задание | | |
| по дисциплине «Теория вычислительных процессов и структур» | | |
| **Сети Петри** | | |
|  | | |
|  | Бригада №6 | Ерощенко Артем, Кауфман Яна, Цыренова Сарюна |
| Группа ПМИ-33 |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Преподаватель | Дворецкая виктория константиновна |
|  |  |
| Новосибирск,2024 | | |

**1.** **Цель работы:**

**2. Задание:**

Для данной маркированной сети Петри, заданной графом, файл с вариантами заданий прикреплён.

****

Задание 1. Найти матрицы *D-*, *D+*, составную матрицу изменений *D*.

Задание 2. Построить дерево достижимости.

Задание 3. Провести анализ свойств сети Петри.

**3. Анализ задачи:**

**Входные данные: JSON файл сети Петри**

**Выходные данные:** матрицы D+, D-, D, дерево достижимости, матрица достижимости.

**Метод решения:** Программа считывает данные из **JSON файл сети Петри. Функция print\_destination\_tree реализует построение дерева достижимости для сети Петри на основе заданных параметров, а print\_destination\_tree методом перебора строит матрицу достижимости. В последствии, в консоль выводятся** матрицы D+, D-, D, дерево достижимости и матрицу достижимости.

**4. Код программы:**

import json # To read json file of Petri network

import numpy as np # To manipulate matrixes

def get\_destination\_matrix(network: dict, transitions: list, points: list):

dest = np.zeros((points\_len, points\_len), dtype=np.int32)

# Fill destination matrix

for trans in transitions:

for inp\_ind in [points.index(i[0]) for i in network[trans]["inp"]]:

for out\_ind in [points.index(i[0]) for i in network[trans]["out"]]:

dest[inp\_ind, out\_ind] = 1

return dest

def is\_vital(network: dict, transitions: list):

for trans in transitions:

if not len([points.index(i[0]) for i in network[trans]["inp"]]):

return False

return True

def print\_destination\_tree(start\_label: np.ndarray, D: np.ndarray, network: dict, transitions: list, points: list, steps: int, cur\_step = 0, cur\_trans = ''):

if cur\_step >= steps:

return

if cur\_step > 0:

print('|' \* (cur\_step - 1) + '∟', end='')

print(cur\_trans, start\_label, sep='')

points\_to = set([l for l\_ind, l in enumerate(points) if start\_label[l\_ind] != 0])

for trans\_ind, trans in enumerate(transitions):

new\_label = start\_label + D[trans\_ind]

if all(new\_label >= 0) and len(network[trans]["inp"]) > 0 and set([i[0] for i in network[trans]["inp"]]).issubset(points\_to):

print\_destination\_tree(new\_label, D, network, transitions, points, steps, cur\_step + 1, trans)

# Reading Petri network

with open('.\RGR\graph.json', 'r') as f:

petri = json.load(f)

# Some Global variables

points = petri["points"]

transitions = petri["transitions"]

points\_len, transitions\_len = len(points), len(transitions)

tree\_steps\_c = 5

start\_tree\_label = [1, 1, 0, 0, 0]

# Task 1: Change matrix

# Fill part's matrixes

D\_minus, D\_plus = np.zeros((transitions\_len, points\_len), dtype=np.int32), np.zeros((transitions\_len, points\_len), dtype=np.int32)

for trans\_ind, trans in enumerate(transitions):

# For input matrix

for point in petri["network"][trans]["inp"]:

point\_ind = points.index(point[0])

D\_minus[trans\_ind, point\_ind] = 1

# For output matrix

for point in petri["network"][trans]["out"]:

point\_ind = points.index(point[0])

D\_plus[trans\_ind, point\_ind] = 1

# Print Change matrix

D = D\_plus - D\_minus

print('D+:\n', D\_plus, sep='')

print('D-:\n', D\_minus, sep='')

print('D:\n', D, sep='')

# Task 2: Destination tree

# Print destianation tree for every label

start\_tree\_label = np.array(start\_tree\_label, dtype=np.int32)

print(f"Destination tree of steps: {tree\_steps\_c}:")

print\_destination\_tree(start\_tree\_label, D, petri["network"], transitions, points, tree\_steps\_c)

# Task 3: analysis

# Destination matrix

dest = get\_destination\_matrix(petri["network"], transitions, points)

print("Destination matrix:\n", dest, sep='')

# Vitality

print("Vitality: ", is\_vital(petri["network"], transitions), sep='')

**5.Задание №1**

D+ = D- = D =

**6.Задание №2**

Дерево достижимости: [1 1 0 0 0]

∟T0[0 0 1 1 1]

**7.Задание №3**

1. Сеть k-ограничена, так как ни одна из позиций не накапливает бесконечное количество меток. Ни в одной позиции сети при любой последовательности срабатываний переходов количество фишек не превышает некоторого K.

2. Сеть безопасна, так как может быть ограничена 1 меткой. Если в позициях p0, p1 будет по 1 метке, то позиции p4, p2, p3 будут содержать по 1 метке.

3. Если предположить, что весовой коэффициент каждой позиции равен 1, то сеть не сохраняема. Если p0 и p1 будут обладать весовыми коэффициентами 1/2, а p2, p3, p4 – 1/3 , то сеть будет сохраняема.

4. Позиции p2, p3, p4 достижимы из позиций p0, p1.

Матрица достижимости:

5. Сеть не живая, так как существуют два перехода t1, t2 для которых не существует входных дуг, и они не выполняются.

1

0