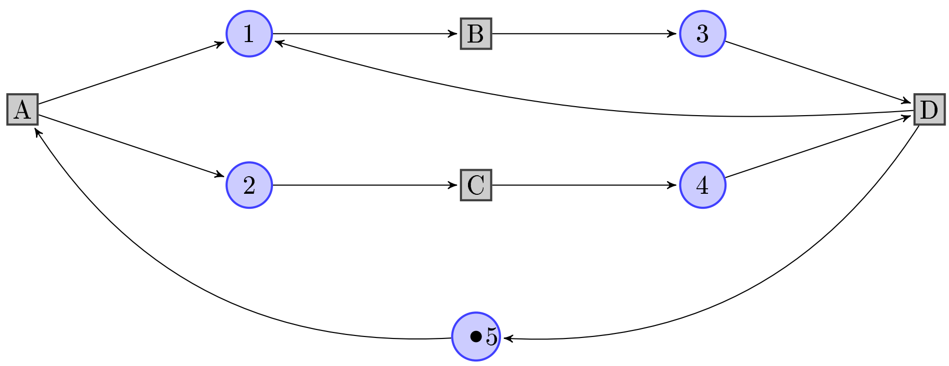
Лабораторна робота №4 з Математичного моделювання

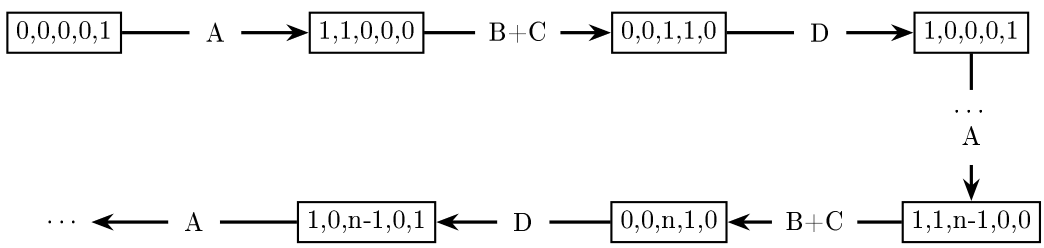
Шкаліков Олег, ФІ-81

10 листопада 2020 р.

Завдання 1

Побудувати граф досяжних розміток мережі Петрі.





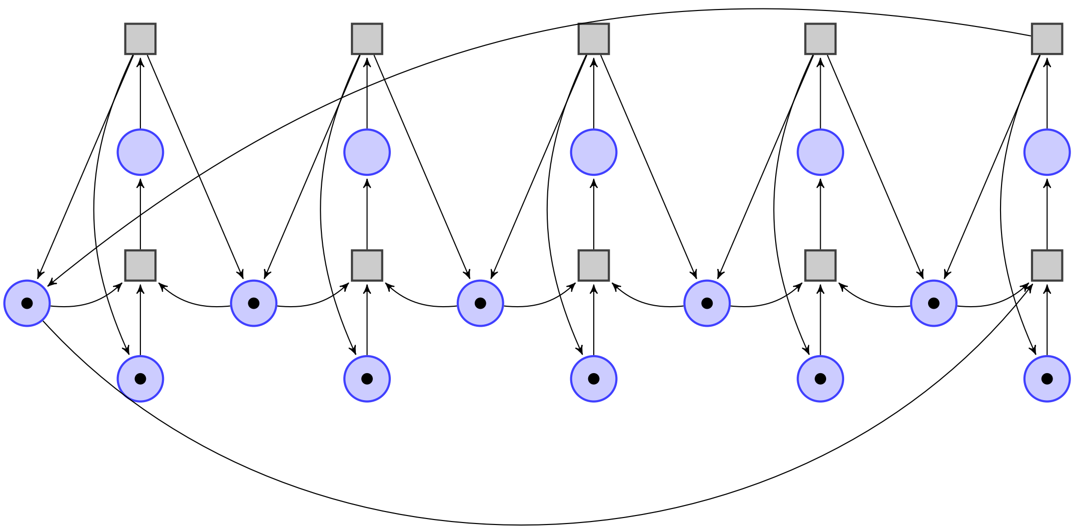
Де n - номер "ітерації"у циклі, який є у графі, що відповідає мережі Петрі(кількість разів, коли перехід A був виконаним).

Як ми можемо бачити мережі Петрі не відповідає мова, яка допускає скінченне слово(у графі досяжних розміток нескінченність вершин та немає термінальних вершин).

Завдання 2

Для виконання завдання варіанту 4 було обрано метод поступового ускладнення моделей з 1, 2 та 3 варіантів. Тому розглянемо процес створення мережі Петрі покроково. У доданках до цього протоколу наведено програмну реалізацію описаних мереж Петрі на мові Python за допомогою яких ми перевіряли корректність наведених моделей.

Варіант 1



Тут і надалі будемо нумерувати рівні позицій(кулі) та переходів(квадрати) знизу догори (окремо позиції та переходи). Тобто, вершини рівню 1 - це найнижчі 5 позицій у графі вище. Позиції рівню 1 - це найнижчі 5 переходів у графі вище.

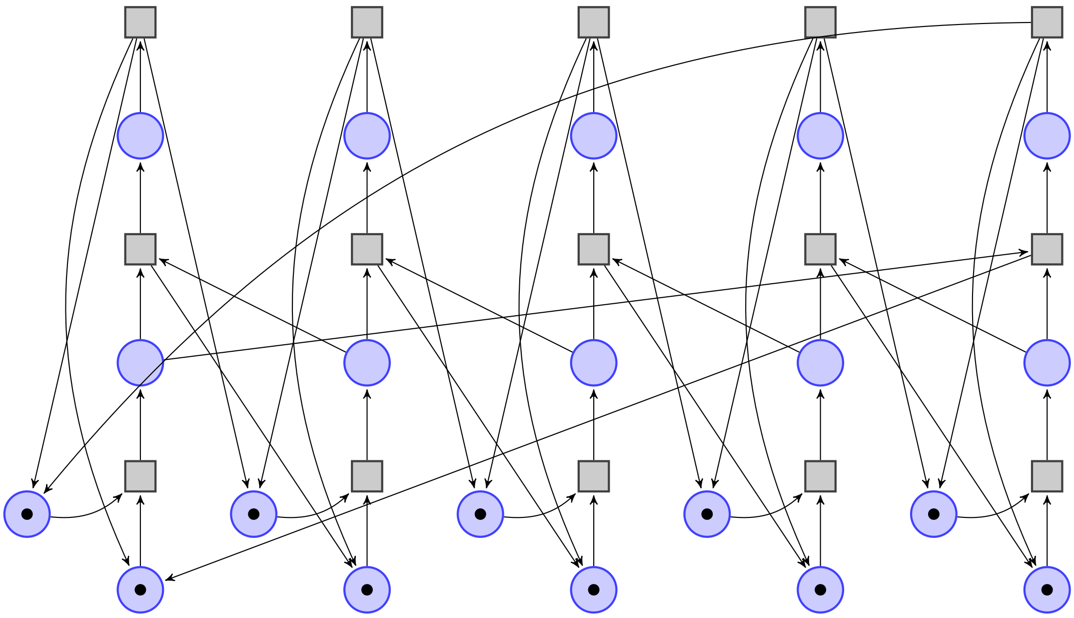
Опишемо значення переходів та позицій цього варіатну мережі Петрі. Позиції:

1. Філософ гуляє
2. Виделка лежить на своєму місці
3. Філософ обідає

Переходи:

1. Філософ бере 2 виделки
2. Філософ повертається до прогулянки та кладе виделки на місце

Варіант 2



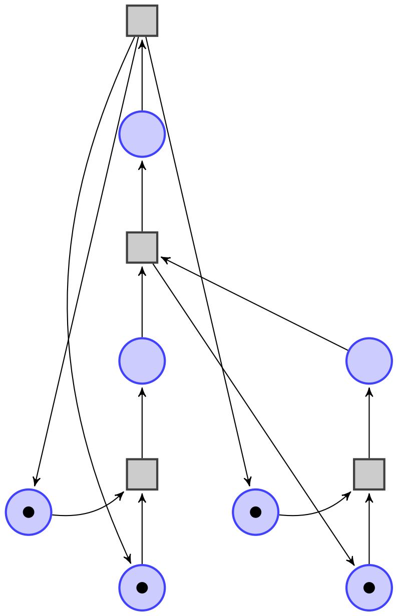
Опишемо значення переходів та позицій цього варіанту мережі Петрі. Позиції:

1. Філософ гуляє
2. Виделка лежить на своєму місці
3. Філософ взяв ліву виделку
4. Філософ обідає

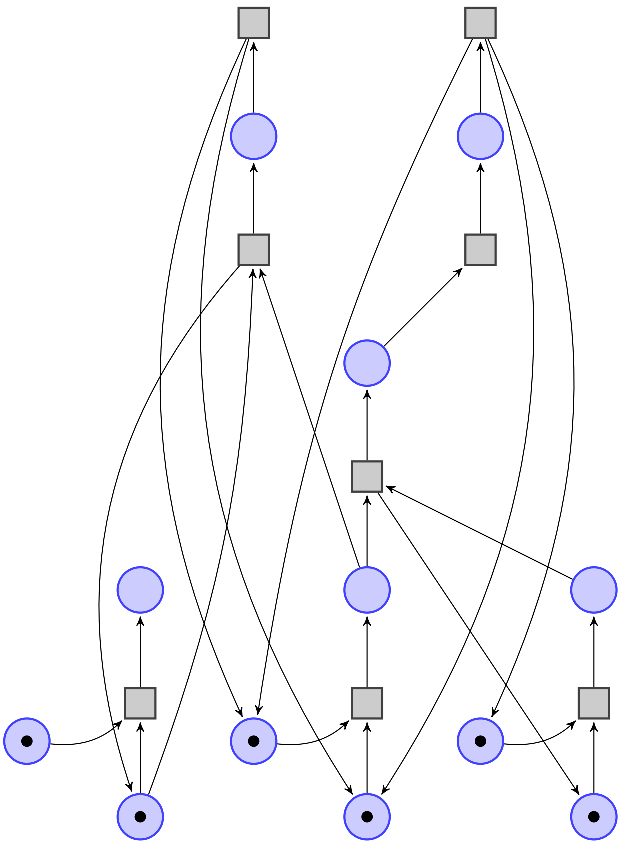
Переходи:

1. Філософ бере ліву виделки
2. Філософ бере праву виделку(її передав йому правий сусід)
3. Філософ повертається до прогулянки та кладе виделки на місце

Як ми можемо бачити, граф мережі Петрі має багато дуг, що значно ускладнює задачу його зображення на малюнку. Проте помітимо, що граф нашої задачі можна розділити на 5 блоків, що відповідають кожному з філософів. Тому для цього варіанту та надалі будемо малювати малюнки лише для одного блоку.



Варіант 3



Опишемо значення переходів та позицій цього варіанту мережі Петрі. Позиції:

1. Філософ гуляє
2. Виделка лежить на своєму місці
3. Філософ взяв ліву виделку
4. Філософ взяв обидві виделки
5. (а) (права гілка) Філософ обідає
   (б) (ліва гілка) Філософ приєднався до тих, кто обідає(сидить за столом)

Переходи:

1. Філософ бере ліву виделки
2. Філософ бере праву виделку(її передав йому правий сусід)
3. (а) (права гілка) Філософ готується до обіду
   (б) (ліва гілка) Філософ запрошується до столу
4. Філософ повертається до прогулянки та кладе виделки(у) на місце

Простими словами механізм запрошення можна описати так: запрошується той, хто ні отримав праву виделку від сусіда, ні віддав свою виделку лівому сусіду.

Варіант 4

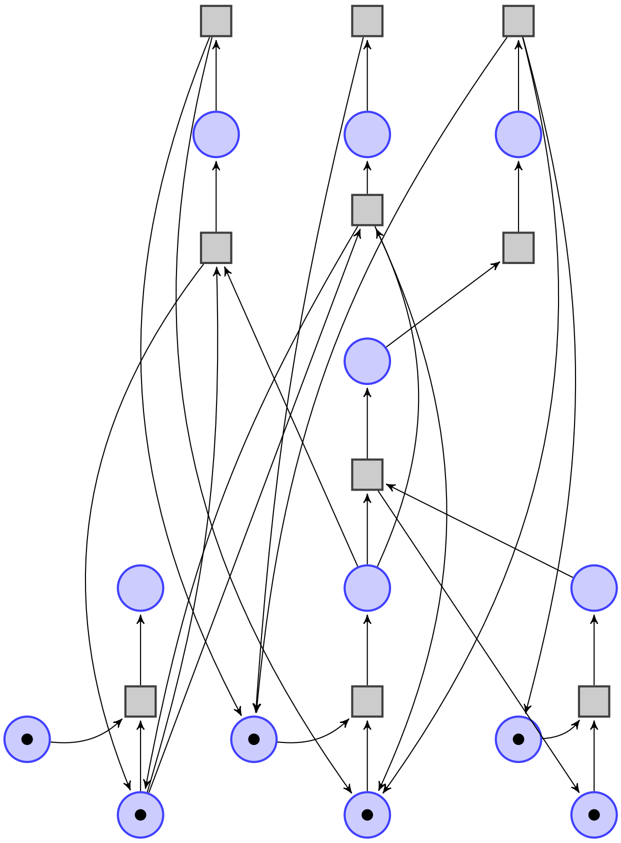
Опишемо значення переходів та позицій цього варіанту мережі Петрі. Позиції:

1. Філософ гуляє
2. Виделка лежить на своєму місці
3. Філософ взяв ліву виделку
4. Філософ взяв обидві виделки
5. (а) (права гілка) Філософ обідає
   (б) (середня гілка) Філософ не прийняв запрошення
   (в) (ліва гілка) Філософ прийняв запрошення

Переходи:

1. Філософ бере ліву виделки
2. Філософ бере праву виделку(її передав йому правий сусід)

3. (а) (права гілка) Філософ готується до обіду
   (б) (середня гілка) Філософ не погоджується на запрошення
   (в) (ліва гілка) Філософ погоджується на запрошення
4. Філософ повертається до прогулянки(ліва та права гілка). Виделка повертається на місце - середня гілка.



Додатки

Далі наведено програмний код імплементованих мереж Петрі.

```
import numpy as np
from typing import Iterable, List


class PetriNet:

    def __init__(self, n_transition: int, n_position: int
    ↪ , markup: List) -> None:
        assert(n_transition > 0)
        assert(n_position == len(markup))

        self.I = np.zeros((n_position, n_transition),
            ↪ dtype=np.int)
        self.O = np.zeros((n_transition, n_position),
            ↪ dtype=np.int)
        self.markup = np.array(markup)

    def set_input(self, trans_idx: int, pos_idx: int,
    ↪ edge_count: int) -> None:
        assert(pos_idx < self.I.shape[0])
        assert(pos_idx >= 0)
        assert(trans_idx < self.I.shape[1])
        assert(trans_idx >= 0)
        assert(edge_count >= 0)

        self.I[pos_idx, trans_idx] = edge_count

    def set_output(self, trans_idx: int, pos_idx: int,
    ↪ edge_count: int) -> None:
        assert(pos_idx < self.O.shape[1])
        assert(pos_idx >= 0)
        assert(trans_idx < self.O.shape[0])
        assert(trans_idx >= 0)
        assert(edge_count >= 0)

        self.O[trans_idx, pos_idx] = edge_count

    def _is_possible(self, trans_idx: int) -> bool:
        return np.all((self.markup - self.I[:, trans_idx
            ↪ ]) >= 0)

    def __iter__(self):
        return self

    def __next__(self) -> Iterable:
        trans_idx = np.arange(self.O.shape[0])
```

```
        np.random.shuffle(trans_idx)
        markup_diff = np.zeros_like(self.markup)

        for i in trans_idx:
            if(self._is_possible(i)):
                self.markup -= self.I[:, i]
                markup_diff += self.O[i, :]

        self.markup += markup_diff
        return self.markup

from PetriNet import PetriNet
import numpy as np


def variant1() -> PetriNet:
    init_markup = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
        ↪ 0, 0]
    net = PetriNet(10, 15, init_markup)

    for i in range(5):
        net.set_input(i, i, 1)
        net.set_input(i, i + 5, 1)
        if i != 0:
            net.set_input(i, i + 4, 1)
        else:
            net.set_input(0, 9, 1)
        net.set_output(i, i+10, 1)

        net.set_input(i+5, i+10, 1)
        net.set_output(i+5, i+5, 1)
        net.set_output(i+5, i, 1)
        if i != 0:
            net.set_output(i+5, i+4, 1)
        else:
            net.set_output(5, 9, 1)

    return net


def variant2() -> PetriNet:
    init_markup = [1, 1, 1, 1, 1,  # walking position
                   1, 1, 1, 1, 1,  # forks position
                   0, 0, 0, 0, 0,  # philosoph takes left
                       ↪ fork
                   0, 0, 0, 0, 0]  # eating
    net = PetriNet(15, 20, init_markup)

    for i in range(5):
        net.set_input(i, i, 1)
```

```
        net.set_input(i, i + 5, 1)
        net.set_output(i, i+10, 1)

        net.set_input(i+5, i+10, 1)
        if i != 4:
            net.set_input(i+5, i+11, 1)
        else:
            net.set_input(9, 10, 1)
        net.set_output(i+5, i+15, 1)
        if i != 4:
            net.set_output(i+5, i+1, 1)
        else:
            net.set_output(9, 0, 1)

        net.set_input(i+10, i+15, 1)
        net.set_output(i+10, i, 1)
        net.set_output(i+10, i+5, 1)
        if i != 4:
            net.set_output(i+10, i+6, 1)
        else:
            net.set_output(14, 5, 1)

    return net


def variant3() -> PetriNet:
    init_markup = [1, 1, 1, 1, 1,  # walking position
                   1, 1, 1, 1, 1,  # forks position
                   0, 0, 0, 0, 0,  # philosoph takes left
                       ↪ fork
                   0, 0, 0, 0, 0,  # philosoph take right
                       ↪ fork
                   0, 0, 0, 0, 0,  # sitting
                   0, 0, 0, 0, 0]  # eating
    net = PetriNet(30, 30, init_markup)

    for i in range(5):
        # can take left fork?
        net.set_input(i, i, 1)
        net.set_input(i, i + 5, 1)
        net.set_output(i, i+10, 1)

        # will take away from the right neighbor?
        net.set_input(i+5, i+10, 1)
        if i != 4:
            net.set_input(i+5, i+11, 1)
        else:
            net.set_input(9, 10, 1)
        net.set_output(i+5, i+15, 1)
        if i != 4:
```

```
            net.set_output(i+5, i+1, 1)
        else:
            net.set_output(9, 0, 1)

        # Select sitting
        net.set_input(i+10, i+10, 1)
        net.set_output(i+10, i+20, 1)
        if i != 0:
            net.set_input(i+10, i-1, 1)
            net.set_output(i+10, i-1, 1)
        else:
            net.set_input(10, 4, 1)
            net.set_output(10, 4, 1)

        # Start eating
        net.set_input(i+15, i+15, 1)
        net.set_output(i+15, i+25, 1)

        # End of sitting
        net.set_input(i+20, i+20, 1)
        net.set_output(i+20, i, 1)
        net.set_output(i+20, i+5, 1)

        # End of eating
        net.set_input(i+25, i+25, 1)
        net.set_output(i+25, i, 1)
        net.set_output(i+25, i+5, 1)
        if i != 4:
            net.set_output(i+25, i+6, 1)
        else:
            net.set_output(29, 5, 1)

    return net


def variant4() -> PetriNet:
    init_markup = [1, 1, 1, 1, 1,  # walking position
                   1, 1, 1, 1, 1,  # forks position
                   0, 0, 0, 0, 0,  # philosoph takes left
                       ↪ fork
                   0, 0, 0, 0, 0,  # philosoph take right
                       ↪ fork
                   0, 0, 0, 0, 0,  # sitting
                   0, 0, 0, 0, 0,  # eating
                   0, 0, 0, 0, 0]  # declined the
                       ↪ invitation(service positions)
    net = PetriNet(40, 35, init_markup)

    for i in range(5):
        # can take left fork?
```

```python
net.set_input(i, i, 1)
net.set_input(i, i + 5, 1)
net.set_output(i, i+10, 1)

# will take away from the right neighbor?
net.set_input(i+5, i+10, 1)
if i != 4:
    net.set_input(i+5, i+11, 1)
else:
    net.set_input(9, 10, 1)
net.set_output(i+5, i+15, 1)
if i != 4:
    net.set_output(i+5, i+1, 1)
else:
    net.set_output(9, 0, 1)

# Will decline the invitation?
net.set_input(i+10, i+10, 1)
net.set_output(i+10, i+30, 1)
net.set_output(i+10, i, 1)
if i != 0:
    net.set_input(i+10, i-1, 1)
    net.set_output(i+10, i-1, 1)
else:
    net.set_input(10, 4, 1)
    net.set_output(10, 4, 1)

# Will accept invitation?
net.set_input(i+15, i+10, 1)
net.set_output(i+15, i+20, 1)
if i != 0:
    net.set_input(i+15, i-1, 1)
    net.set_output(i+15, i-1, 1)
else:
    net.set_input(15, 4, 1)
    net.set_output(15, 4, 1)

# Start eating
net.set_input(i+20, i+15, 1)
net.set_output(i+20, i+25, 1)

# End of sitting
net.set_input(i+25, i+20, 1)
net.set_output(i+25, i, 1)
net.set_output(i+25, i+5, 1)

# End of eating
net.set_input(i+30, i+25, 1)
net.set_output(i+30, i, 1)
net.set_output(i+30, i+5, 1)
```

```
        if i != 4:
            net.set_output(i+30, i+6, 1)
        else:
            net.set_output(34, 5, 1)

        # Service transition(for declined invitations)
        net.set_input(i+35, i+30, 1)
        net.set_output(i+35, i+5, 1)

    return net


if __name__ == "__main__":
    net = variant4()
    N = 10

    for i in range(4*N-1):
        result = next(net)
        if (i+2) % 4 == 0:
            walking = np.argwhere(result[:5] > 0)[:, 0]
            siting = np.argwhere(result[20:25] > 0)[:, 0]
            eating = np.argwhere(result[25:30] > 0)[:, 0]

            print(" ".join([f"{k}: {str(v + 1)}" for k, v
                ↪ in
                                zip(['walking', 'siting', '
                                    ↪ eating'],
                                    [walking, siting, eating
                                        ↪ ])]))

if __name__ == "__main__2":
    net = variant2()
    N = 10

    for i in range(3*N-1):
        result = next(net)
        if (i+2) % 3 == 0:
            walking = np.argwhere(result[:5] + result
                ↪ [10:15] > 0)[:, 0]
            eating = np.argwhere(result[15:20] > 0)[:, 0]

            print(" ".join([f"{k}: {str(v + 1)}" for k, v
                ↪ in
                                zip(['walking', 'eating'],
                                    [walking, eating])]))


if __name__ == "__main__1":
    net = variant1()
    N = 10
```

```
for i in range(2*N-1):
    result = next(net)
    if i % 2 == 0:
        walking = np.argwhere(result[:5] > 0)[:, 0]
        eating = np.argwhere(result[10:15] > 0)[:, 0]

        print(" ".join([f"{k}: {str(v + 1)}" for k, v
            ↪ in
                        zip(['walking', 'eating'],
                            [walking, eating])]))
```