Laboratorium XI - Teoria współbieżności

Weronika Szybińska, 09.01.2023

Treść zadania:

Dane są:

- Alfabet A, w którym każda litera oznacza akcję.
- Relacja niezależności I, oznaczająca które akcje są niezależne (przemienne, tzn. można je wykonać w dowolnej kolejności i nie zmienia to wyniku końcowego).
- Słowo w oznaczające przykładowe wykonanie sekwencji akcji.

Zadanie:

Napisz program w dowolnym języku, który:

- Wyznacza relację niezależności I
  - wyznaczona została relacja zależności D, gdyż relacja niezależności I jest argumentem w zadaniu
- Wyznacza ślad [w] względem relacji I
- Wyznacza postać normalną Foaty FNF([w]) śladu [w]
- Wyznacza graf zależności dla słowa w
- Wyznacza postać normalną Foaty na podstawie grafu

Rozwiązanie:

Do rozwiązania podanych zadań wykorzystano język programowania Python oraz bibliotekę Graphiz.

W zadaniu 1 utworzona została funkcja zwracająca relacje zależności D (zmienione w stosunku do oryginalnej treści zadania gdyż relacja niezależności jest podawana jako argument z zadaniu). W funkcji tworzona jest lista zawierająca wszystkie możliwe połączenia między akcjami w alfabecie prócz tych znajdujących się w podanej liście zwierającej relację niezależności. Poniżej przedstawiona została implementacja funkcji.

```
def relation_D(A, I):
    return [(i,j) for i in A for j in A if (i,j) not in I]
```

Do wykonania zadania 2 utworzona została funkcja zwracająca ślad [w] względem podanej relacji niezależności. Jako argumenty funkcja dostaję zadane słowo w, listę relacji niezależności I oraz pustą listę do której dopisywane będą kolejne słowa. Początkowo dla każdych akcji znajdujących się obok siebie w podanym słowie sprawdza możliwość przestawienia ich.

Jeżeli akcje są niezależne i jest taka możliwość przestawia je i za pomocą rekursji odpala funkcje dla nowo powstałego słowa. Poniżej przedstawiona została implementacja opisanej funkcji.

```
def trace(word, I, traces):
    if word in traces:
        return
    traces.append(word)
    for i in range(len(word) - 1):
        if (word[i], word[i+1]) in I:
            letter_list = list(word)
            letter_list[i], letter_list[i+1] = letter_list[i+1],letter_list[i]
            trace(''.join(letter_list), I, traces)
```

W zadaniu 3 utworzone zostały 2 funkcję pomocnicze i jedna główna zawracająca postać normalną Foaty FNF([w]) śladu [w]. Funkcja create_stacks tworzy stosy zgodnie z algorytmem z zaproponowanej pracy (Handbook of Formal Languages). Funkcja stacks_remove usuwa dla każdej akcji kolejne wartości z jej stosu (jeżeli wartością tą jest * to jest ona usuwana). W funkcji foata_form najpierw tworzony jest stos za pomocą create_stacks. Następnie póki wszystkie stosy nie będą puste pobiera wartości ze stosów za pomocą funkcji stack_remove, które dodawane są do listy result_letters. Ze zwróconych wartości znajdujących się w liście tworzone są kolejne części postaci normalnej Foaty. Poniżej przedstawiona została implementacja powyższych funkcji.

```
def create_stacks(A, I, w):
    dependent_relations = relation_D(A,I)
    stacks = {char: [] for char in A}
    for l in w:
        stacks[l].append(l)
        for rel in dependent_relations:
            if rel[0] == l and rel[1] != l:
                stacks[rel[1]].append('*')

    return stacks

def stack_remove(stacks):
    letters = [stacks.get(letter).pop() for letter in stacks if len(stacks.get(letter)) > 0]
    result = set(letters)
    if "*" in result:
        result.remove("*")
    return sorted(list(result))

def foata_form(A, I, w):
    stacks = create_stacks(A, I, w)
    max_size = max([len(value) for value in stacks.values()])
    result_letters = []

    for i in range(max_size):
        letters = stack_remove(stacks)
        result_letters. append(letters)

    foata = ''
    for inx in range(len(result_letters) - 1, -1, -1):
        if len(result_letters[inx]) > 0:
            foata += '(' + ''.join([i[0] for i in result_letters[inx]]) + ')'
    return foata
```

Do wykonania zadania 4 zaimplementowane zostały funkcje tworzące graf zależności dla słowa w. Na początku funkcją create_graph dodaje do grafu wierzchołki (każdy wierzchołek to akcja w śladzie [w]). Następnie do listy krawędzi dodawane są wszystkie możliwe krawędzie (tzn. jeżeli akcja x poprzedzająca akcje z jest od niej zależna krawędź (x,y) jest dodawana do listy). Następnie za pomocą funkcji remove_unecessery usuwane są nadmiarowe krawędzie. Funkcja remove_unecessery jest funkcją rekurencyjną. Jako dane wejściowe dostaje początkową krawędź z której szukamy drogi do kolejnych osiągalnych wierzchołków. Jeżeli osiągniemy wierzchołek połączony krawędzią z wierzchołkiem od którego zaczynaliśmy krawędź ta jest usuwana (jest nadmiarowa, bo do wierzchołka można dotrzeć nie używając jej). Po usunięciu wszystkich niepotrzebnych krawędzi tworzony jest graf. Poniżej przedstawiona jest implementacja opisanych funkcji.

```
def remove_unecessery(node, edges, copy_edges, length, inx=0):
    if (node[0], copy_edges[inx][1]) in edges and length > 0:
        edges.remove((node[0], copy_edges[inx][1]))

    for inx2 in range(inx+1, len(copy_edges)):
        if copy_edges[inx2][0] == copy_edges[inx][1]:
            length += 1
            remove_unecessery(node, edges, copy_edges, length, inx2)

def create_graph(word,I,A,id):
    graph = Digraph()
    nodes = list(word)
    nodes_t = []

    for n in range(len(nodes)):
        graph.node(str(n)+nodes[n],nodes[n])
        nodes_t.append(str(n)+nodes[n])

    D = relation_D(A,I)
    edges_t = []

    for n1 in range(len(nodes_t)):
        for n2 in range(n1+1,len(nodes_t)):
            if (nodes_t[n1][1],nodes_t[n2][1]) in D:
                edges_t.append((nodes_t[n1],nodes_t[n2]))

    for i in edges_t:
        remove_unecessery(i,edges_t, edges_t.copy(), 0, edges_t.index(i))
    for e1,e2 in edges_t:
        graph.edge(e1,e2)

    graph.render('graph'+str(id)+'.gv', view=True)
    return nodes_t,edges_t
```

Do wykonania ostatniego zadania utworzona została funkcja wyznaczająca postać normalną Foaty z utworzonego wcześniej grafu. Funkcja jako argument dostaje listę krawędzi i zadane słowo w. Funkcja idzie po kolejnych akcjach śladu [w], jeżeli w liście tymczasowej nie ma akcji która posiada z obecnie sprawdzana akcją krawędzi w grafie to dodaje ten wierzchołek do listy tymczasowej. W momencie gdy akcja ma krawędź z którąś z akcji w liście, tymczasowa lista jest zerowana a jej zawartość dodawana jest do listy pośredniej. Lista pośrednia jest przekształcana w listą końcowa usuwając niepotrzebne znaki i puste listy. Lista końcowa zawiera listy, które są częściami postaci normlanej. Na koniec z listy tej tworzony jest zwracany string. Poniżej przedstawiona została implementacja.

```python
def foata_from_graph(edges, w):
    list = []
    temp = []
    for inx in range (len(w)-1):
        temp.append(str(inx)+w[inx])
        for val in temp:
            if (val,str(inx+1)+w[inx+1]) in edges:
                list.append(temp)
                temp = []

    if len(temp) == 0:
        list.append([w[len(w)-1]])
    else:
        temp.append(w[len(w)-1])
        list.append(temp)

    result = []
    for val in list:
        if len(val) == 0:
            continue
        temp = []
        for i in val:
            if len(i) == 2:
                temp.append(i[1])
            else:
                temp.append(i)
        result.append(temp)

    foata = ''
    for val in result:
        foata += '(' + ''.join([i[0] for i in val]) + ')'
    return foata
```

Wyniki:

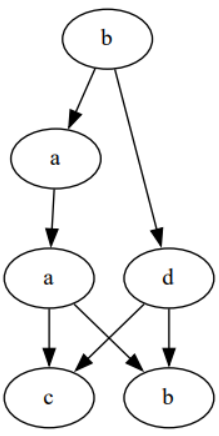
Dla podanych danych wejściowych:

```
print("Example 1")
A = ['a', 'b', 'c', 'd']
I = [('a', 'd'), ('d', 'a'), ('b', 'c'), ('c', 'b')]
w = "baadcb"
print("Relacja zależności D: ")
print(relation_D(A,I))
print("Ślad [w] względem relacji I:")
traces = []
trace(w,I,traces)
print(traces)
print("Postać normalna Foaty śladu [w]:")
print(foata_form(A,I,w))
print("Postać normalna Foaty śladu [w] na podstawie grafu:")
nodes, edges = create_graph(w, I, A, 1)
print(foata_from_graph(edges, w))


print("Example 2")
A = ['a', 'b', 'c', 'd', 'e', 'f']
I = [("a", "d"), ("d", "a"), ("b", "e"), ("e", "b"), ("c", "d"), ("d", "c"), ("c", "f"), ("f", "c")]
w = "acdcfbbe"
print("Relacja zależności D: ")
print(relation_D(A, I))
print("Ślad [w] względem relacji I:")
traces = []
trace(w, I, traces)
print(traces)
print("Postać normalna Foaty śladu [w]:")
print(foata_form(A, I, w))
print("Postać normalna Foaty śladu [w] na podstawie grafu:")
nodes, edges = create_graph(w, I, A, 1)
print(foata_from_graph(edges, w))
```

Otrzymano:

```
Example 1
Relacja zależności D:
[('a', 'a'), ('a', 'b'), ('a', 'c'), ('b', 'a'), ('b', 'b'), ('b', 'd'), ('c', 'a'), ('c', 'c'), ('c', 'd'), ('d', 'b'), ('d', 'c'), ('d', 'd')]
Ślad [w] względem relacji I:
['baadcb', 'badacb', 'bdaacb', 'bdaabc', 'badabc', 'baadbc']
Postać normalna Foaty śladu [w]:
(b)(a)(ad)(bc)
Postać normalna Foaty śladu [w] na podstawie grafu:
(b)(a)(ad)(cb)
```



Dla Example 2 (za duże wyniki do przedstawienia na zdjęciu):

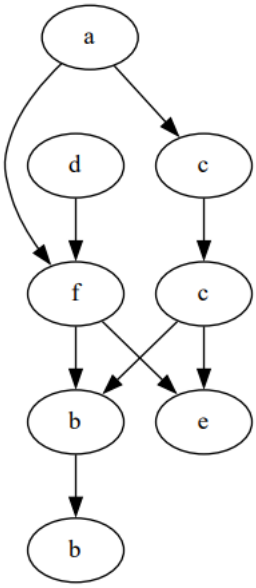
Relacja zależności D:

[('a', 'a'), ('a', 'b'), ('a', 'c'), ('a', 'e'), ('a', 'f'), ('b', 'a'), ('b', 'b'), ('b', 'c'), ('b', 'd'), ('b', 'f'), ('c', 'a'), ('c', 'b'), ('c', 'c'), ('c', 'e'), ('d', 'b'), ('d', 'd'), ('d', 'e'), ('d', 'f'), ('e', 'a'), ('e', 'c'), ('e', 'd'), ('e', 'e'), ('e', 'f'), ('f', 'a'), ('f', 'b'), ('f', 'd'), ('f', 'e'), ('f', 'f')]

Ślad [w] względem relacji I:

['acdcfbbe', 'adccfbbe', 'daccfbbe', 'dacfcbbe', 'adcfcbbe', 'acdfcbbe', 'acdfcbeb', 'adcfcbeb', 'dacfcbeb', 'dafccbeb', 'adfccbeb', 'adfccebb', 'dafccebb', 'dacfcebb', 'adcfcebb', 'acdfcebb', 'acdcfebb', 'adccfebb', 'daccfebb', 'daccfbeb', 'adccfbeb', 'acdcfbeb', 'accdfbeb', 'accdfebb', 'accdfbbe', 'adfccbbe', 'dafccbbe']

```
Postać normalna Foaty śladu [w]:
(a)(cd)(cf)(b)(be)
Postać normalna Foaty śladu [w] na podstawie grafu:
(a)(cd)(cf)(b)(be)
```



Bibliografia:

http://www.graphviz.org/

https://home.agh.edu.pl/~funika/tw/lab-trace2/

https://www-users.mat.umk.pl//~edoch/materialy.html