Laboratorium XI

Teoria śladów cz. II

Dominik Marek

27 grudnia 2024

AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

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:

1. Wyznacza relację zależności D
2. Wyznacza ślad $[w]$ względem relacji I
3. Wyznacza postać normalną Foaty $FNF([w])$ śladu $[w]$
4. Wyznacza graf zależności dla słowa w

5. Wyznacza postać normalną Foaty na podstawie grafu

Do zadania należy dostarczyć sprawozdanie, które będzie zawierać:

1. Opis programu z komentarzami
2. Wyniki działania dla przykładowych danych

Sprawozdanie i kod programu proszę przysłać na adres: funika@agh.edu.pl

Uwagi:

- Proszę wykorzystać algorytm ze str. 10 rozdziału Partial commutation and traces (pochodzi z Handbook of Formal Languages, Springer, 1997.
- Do rysowania grafu można wykorzystać Graphviz i format DOT. Wersja online: Webgraphviz.

Przykład

Dla danych:

(a) x := x + y

(b) y := y + 2z

(c) x := 3x + z

(d) z := y - z.

A = {a, b, c, d}

I = {(a, d), (d, a), (b, c), (c, b)}

w = baadcb

Wyniki:

1. *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)}*
2. FNF([w]) = (b)(ad)(a)(bc)
3. Graf w formacie dot:

```
digraph g{
 1 -> 2
 2 -> 3
 1 -> 4
 3 -> 5
 4 -> 5
 3 -> 6
 4 -> 6
 1[label=b]
 2[label=a]
```

3[label=a]

4[label=d]

5[label=b]

6[label=c]

}

Dane testowe 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*

2.Rozwiązania

1.

Poniższa funkcja tworzy zbiór zależnych par symboli w alfabecie, które nie należą do relacji niezależności. Wynik to zbiór wszystkich par z alfabetu, które są zależne od siebie.

```
from itertools import product


def dependent_relations(alphabet, independent_relaton):
    return {pair for pair in product(alphabet, alphabet) if pair not in
independent_relaton}
```

2.

Generuje wszystkie ślady (kolejności znaków) możliwe przez zamianę niezależnych symboli w słowie. Jeśli dwie sąsiadujące litery są niezależne, zamienia ich miejsca i rekurencyjnie analizuje wynikowe słowo.

```
def trace(word, independent_relation, traces):
    traces.append(word)

    for i in range(len(word)-1):
        w = list(word)
        w[i], w[i+1] = w[i+1], w[i]
        swapped_word = ''.join(w)
        if (word[i], word[i+1]) in independent_relation and swapped_word
not in traces:
            trace(swapped_word, independent_relation, traces)
```

3.

Funkcja *create_stacks* buduje stosy reprezentujące zależności między literami w słowie na podstawie alfabetu i relacji niezależności. Każdy stos zawiera indeksy liter oraz symbol *, jeśli występują zależności z innymi literami. Następnie funkcja *stack_pop* usuwa i zwraca zestaw elementów z góry stosów, ignorując symbol *.Zwrócone symbole są posortowane i reprezentują kolejne litery w porządku Foaty. Funkcja *get_foata_form* generuje formę Foaty słowa poprzez iteracyjne usuwanie liter ze stosów, grupując litery w niezależne klasy. Litery są zbierane w grupy tak, aby w każdym kroku były niezależne od siebie. Finalnie za pomocą funkcji *foata_form_string* przekształcamy listę grup liter (forma Foaty) w format tekstowy, gdzie każda grupa jest umieszczona w nawiasach.

```
def create_stacks(alphabet, independent_relation, word):

    stacks = {char: [] for char in alphabet}
    index = len(word)

    for char in word[::-1]:
        stacks[char].append((char, str(index)))
        index -= 1

        for letter in alphabet:
            if letter is not char and (char, letter) not in
independent_relation:
                stacks[letter].append('*')
    return stacks


def stack_pop(stacks):

    popped_letters = [stacks.get(letter).pop() for letter in stacks if
len(stacks.get(letter)) > 0]

    set_popped = set(popped_letters)

    if '*' in set_popped:
        set_popped.remove('*')
    return sorted(list(set_popped))

def get_foata_form(alphabet, independent_relation, word):

    stacks = create_stacks(alphabet, independent_relation, word)

    max_size = max([len(value) for value in stacks.values()])

    letter_mapping = []

    for i in range(max_size):
        letters = stack_pop(stacks)
        letter_mapping.append(letters)

    return letter_mapping


def foata_form_string(letter_mapping):

    foata = ''
```

```
    for foata_class in letter_mapping:
        if len(foata_class) > 0:
            foata += '('
            foata += ''.join([i[0] for i in foata_class])
            foata += ')'

    return foata
```

4.

Funkcja tworzy graf zależności, w którym każdy węzeł to litera z indeksem w słowie (np. a1 dla pierwszego wystąpienia litery a). Krawędzie grafu łączą litery, które są zależne zgodnie z podaną relacją niezależności. Dodatkowo funkcja usuwa redundantne krawędzie – jeśli istnieje pośrednia ścieżka między dwoma węzłami przez inne, to bezpośrednia krawędź jest usuwana. Wynikiem jest uproszczony graf zależności, który pomaga w określeniu struktury słowa w formie Foaty.

```
def determine_dependency_graph(word, independent_relation):

    dependency_graph = {char + str(idx + 1): [] for idx, char in
enumerate(word)}

    for i in range(len(word)):
        for j in range(i + 1, len(word)):
            if (word[i], word[j]) not in independent_relation:

                dependency_graph[word[i] + str(i + 1)].append(word[j] +
str(j + 1))


    def remove_redundant_edges(graph):
        nodes = list(graph.keys())


        for node1 in nodes:
            for node2 in graph[node1]:


                for inter in graph[node2]:

                    if inter in graph[node1]:

                        graph[node1].remove(inter)
        return graph


    dependency_graph = remove_redundant_edges(dependency_graph)

    return dependency_graph
```

5.

Na podstawie grafu zależności funkcja iteracyjnie buduje formę normalną Foaty.
W każdej iteracji wybierane są litery, które nie mają zależności od innych aktualnie nieprzetworzonych węzłów – tworzą one grupę w formie Foaty.
Litery te są usuwane z listy nieprzetworzonych, a proces powtarza się, aż wszystkie węzły zostaną przypisane do grup.

```
def foata_normal_form_from_graph(word, dependency_graph):

    unprocessed = set(w + str(i + 1) for i, w in enumerate(word))


    foata_form = []

    while unprocessed:

        independent_set = [
            char for char in unprocessed if all(dep not in unprocessed for
dep in dependency_graph.get(char, []))
        ]


        foata_form.append(''.join(sorted(independent_set)))

        unprocessed -= set(independent_set)


    return '(' + ')('.join(foata_form) + ')'
```

6.

Wywołanie wyżej przedstawionych funkcji dla podanych danych testowych.

```
def main() -> None:
    # Example I
    print(f'--------------EXAMPLE_I--------------')
    alphabet = {'a', 'b', 'c', 'd'}
    independent_relation = {('a', 'd'), ('d', 'a'), ('b', 'c'), ('c', 'b')}
    word = 'baadcb'

    dependent_relation = dependent_relations(alphabet,
independent_relation)

    foata = foata_form_string(
        get_foata_form(alphabet=alphabet,
independent_relation=independent_relation, word=word))

    graph = determine_dependency_graph(word, independent_relation)

    foata_from_graph = foata_normal_form_from_graph(word, graph)

    print(f'{dependent_relation=}')
    print(f'{foata=}')
    print(f'{graph=}')
    for node, edges in graph.items():
        for edge in edges:
            print(f"{node} -> {edge}")
```

```
    print(f'{foata_from_graph=}')
    # Example II
    print(f'--------------EXAMPLE_II--------------')
    alphabet = {'a', 'b', 'c', 'd', 'e', 'f'}
    independent_relation = {('a', 'd'), ('d', 'a'), ('b', 'e'), ('e', 'b'),
('c', 'd'), ('d', 'c'), ('c', 'f'), ('f', 'c')}
    word = 'acdcfbbe'
    dependent_relation = dependent_relations(alphabet,
independent_relation)

    foata = foata_form_string(
        get_foata_form(alphabet=alphabet,
independent_relation=independent_relation, word=word))

    graph = determine_dependency_graph(word, independent_relation)
    foata_from_graph = foata_normal_form_from_graph(word, graph)

    print(f'{dependent_relation=}')
    print(f'{foata=}')
    print(f'{graph=}')
    for node, edges in graph.items():
        for edge in edges:
            print(f"{node} -> {edge}")
    print(f'{foata_from_graph=}')


if __name__ == '__main__':
    main()
```

7.

Wyniki otrzymane dla zadanych danych wejściowych.

```
--------------EXAMPLE_I--------------
dependent_relation={('b', 'a'), ('c', 'c'), ('d', 'd'), ('d', 'c'), ('a',
'b'), ('a', 'a'), ('c', 'd'), ('c', 'a'), ('a', 'c'), ('b', 'd'), ('d',
'b'), ('b', 'b')}
foata='(b)(ad)(a)(bc)'
graph={'b1': ['a2', 'd4'], 'a2': ['a3'], 'a3': ['c5', 'b6'], 'd4': ['c5',
'b6'], 'c5': [], 'b6': []}
b1 -> a2
b1 -> d4
a2 -> a3
a3 -> c5
a3 -> b6
d4 -> c5
d4 -> b6
foata_from_graph='(b)(ad)(a)(bc)'

--------------EXAMPLE_II--------------
dependent_relation={('b', 'a'), ('e', 'c'), ('b', 'c'), ('f', 'a'), ('c',
'b'), ('a', 'e'), ('e', 'f'), ('b', 'f'), ('a', 'b'), ('f', 'f'), ('c',
'a'), ('c', 'c'), ('e', 'd'), ('b', 'd'), ('e', 'e'), ('d', 'f'), ('f',
'd'), ('f', 'e'), ('a', 'a'), ('a', 'c'), ('d', 'd'), ('b', 'b'), ('d',
'e'), ('f', 'b'), ('c', 'e'), ('a', 'f'), ('d', 'b'), ('e', 'a')}
foata='(ad)(c)(cf)(be)(b)'
graph={'a1': ['c2', 'f5'], 'c2': ['c4'], 'd3': ['f5'], 'c4': ['b6', 'e8'],
'f5': ['b6', 'e8'], 'b6': ['b7'], 'b7': [], 'e8': []}
a1 -> c2
a1 -> f5
```

```
c2 -> c4
d3 -> f5
c4 -> b6
c4 -> e8
f5 -> b6
f5 -> e8
b6 -> b7
foata_from_graph='(ad)(c)(cf)(be)(b)'
```

Wnioski i obserwacje

1. *Zależności i niezależności: Analiza relacji zależności i niezależności pozwala na zrozumienie struktury słowa w kontekście wykonywalnych operacji. Relacje niezależności umożliwiają reorganizację sekwencji działań bez zmiany wyniku końcowego.*
2. *Forma normalna Foaty (FNF): Forma Foaty umożliwia przedstawienie słowa w postaci zbiorów operacji, które można wykonać równolegle. Jest to szczególnie użyteczne w przypadku analizy równoległości w systemach współbieżnych.*
3. *Graf zależności: Graficzne przedstawienie relacji między akcjami w postaci grafu zależności jest intuicyjnym narzędziem do wizualizacji ograniczeń między operacjami. Usuwanie redundantnych krawędzi upraszcza analizę.*
4. *Spójność wyników: Forma Foaty wyznaczona zarówno na podstawie stosów, jak i grafu zależności jest spójna, co potwierdza poprawność implementacji algorytmów.*
5. *Praktyczne zastosowanie: Prezentowane metody znajdują zastosowanie w projektowaniu systemów rozproszonych, analizy algorytmów współbieżnych oraz w optymalizacji planowania zadań.*
6. *Efektywność algorytmów: Wykorzystane algorytmy, oparte na publikacjach teoretycznych, zapewniają wysoką precyzję w obliczeniach, co czyni je odpowiednimi narzędziami dla formalnej analizy w dziedzinie języków i automatów.*

3.Bibliografia

- https://www.researchgate.net/publication/280851316_Partial_Commutation_and_Traces
- https://web.archive.org/web/20170908153838/https://pdfs.semanticscholar.org/d67a/c4c1e5967f7e114f390245f28909f259c034.pdf
- https://www.mimuw.edu.pl/~sl/teaching/22_23/TW/LITERATURA/book-of-traces-intro.pdf