Teoria Współbieżności - Zadanie domowe

Maciej Grzybacz

06.11.2024

Treść zadania

Dane

- Alfabet A, w którym każda litera oznacza akcję,
- Zestaw transakcji na zmiennych.

Słowo w oznacza przykładowe wykonanie sekwencji akcji.

Specyfikacja programu

1. Wyznacza relację zależności D
2. Wyznacza relację niezależności I
3. Wyznacza postać normalną Foaty FNF($[w]$) śladu $[w]$
4. Rysuje graf zależności w postaci minimalnej dla słowa w

Wymagania dodatkowe

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

Przykład

Dla danych:

- (a) $x := x + y$
- (b) $y := y + 2z$
- (c) $x := 3x + z$
- (d) $z := y - z$
- $A = \{a, b, c, d\}$
- $w = baadcb$

Wyniki:

- $D = \{(a,a), (a,b), (a,c), (b,a), (b,b), (b,d), (c,a), (c,c), (c,d), (d,b), (d,c), (d,d)\}$
- $I = \{(a,d), (d,a), (b,c), (c,b)\}$
- $\mathrm{FNF}([w]) = (b)(ad)(a)(bc)$
- 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"];
}
```

Opis programu

Program został napisany w języku C++ z wykorzystaniem narzędzia CMake. Kompilator użyty do budowania programu to GCC w wersji 14.2.

Kluczowym elementem programu jest klasa `Solver`, która zawiera metody do przetwarzania danych oraz obliczania formy normalnej Foaty (FNF) i budowy grafu zależności. Ponadto w programie zaimplementowano klasy `Transaction` do przechowywania informacji o pojedynczej transakcji oraz `InputParser` do wczytywania danych z pliku tekstowego.

Najważniejszym punktem programu jest metoda `run()` z klasy `Solver`, która odpowiada za wykonanie obliczeń. Metoda ta wykonuje następujące kroki:

1. `determine_relations()`: Oblicza relacje zależności i niezależności między akcjami. Iteruje przez akcje i ich transakcje, dodając relacje zależności do `dependency_relations_`. Następnie, dla każdej pary akcji, jeśli nie są już zapisane w `dependency_relations_` dodaje je do `independent_relations_`.

2. `calculate_fnf()`: Oblicza formę normalną Foaty (FNF) dla słowa `w`. Najpierw tworzy stos dla każdej litery alfabetu, a następnie przetwarza słowo od końca, dodając litery na odpowiednie stosy i markery na stosy zmiennych zależnych. Po przetworzeniu słowa tworzy zbiory zmiennych, usuwając markery ze stosów zmiennych zależnych.

3. `build_dependency_graph()`: Służy do wyznaczenia minimalnej postaci grafu zależności. Najpierw wyznacza macierz sąsiedztwa między akcjami w słowie. Następnie, dla każdej pary akcji (i, j), gdzie i $<$ j, sprawdza za pomocą funkcji `has_path()` czy krawędź i `->` j wyznacza jedyną ścieżkę między i a j. Jeśli tak, dodaje tę krawędź do grafu zależności i zapisuje ją w formacie .dot. W przeciwnym razie, jeśli istnieje inna ścieżka, krawędź jest uznawana za nadmiarową i nie należy do minimalnego grafu zależności.

Instrukcja budowania i uruchamiania programu

Do zbudowania programu potrzebne są następujące komponenty:

- CMake w wersji 3.28.3 lub nowszej
- Kompilatory GCC i G++ obsługujące standard C++20 (najlepiej w najnowszej wersji)
- MinGW32-make w najnowszej wersji

Proces budowania projektu przebiega w następujący sposób:

Należy otworzyć konsolę w głównym katalogu projektu.

Należy utworzyć katalog roboczy o nazwie `build`:

```
mkdir build
cd build
```

Następnie uruchomić CMake, określając generator `MinGW Makefiles`:

```
cmake .. -G "MinGW Makefiles"
```

Zbudować projekt poleceniem:

```
cmake --build .
```

Po pomyślnym zbudowaniu, program może zostać uruchomiony poleceniem:

```
./FNF.exe
```

Po uruchomieniu programu, zostanie wyświetlony komunikat o konieczności wybrania pliku z katalogu `data`. Należy wpisać samą nazwę pliku, bez rozszerzenia, na przykład `case1`.

Wyniki działania programu

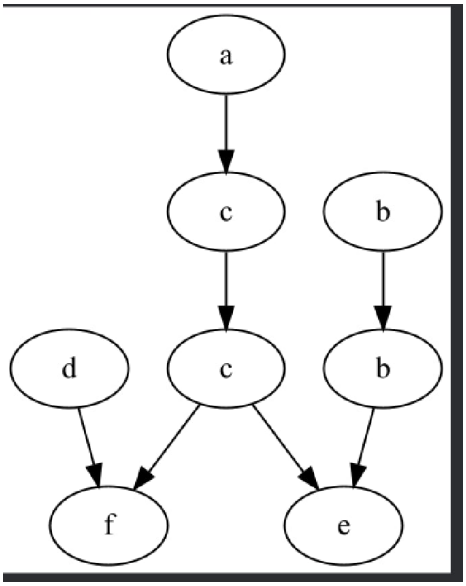
Plik case1.txt

```
Dependency Relations (D):
(d, d) (d, f) (b, b) (b, e) (e, b) (e, c) (e, e) (c, a) (c, c)
(c, e) (c, f) (a, a) (a, c) (a, f) (f, a) (f, c) (f, d) (f, f)

Independent Relations (I):
(f, b) (f, e) (e, a) (e, d) (e, f) (d, a) (d, b) (d, c) (d, e)
(c, b) (c, d) (b, a) (b, c) (b, d) (b, f) (a, b) (a, d) (a, e)

FNF: (adb)(cb)(c)(fe)

digraph g {
1[label=a]
2[label=c]
3[label=d]
4[label=c]
5[label=f]
6[label=b]
7[label=b]
8[label=e]
1 -> 2
2 -> 4
3 -> 5
4 -> 5
4 -> 8
6 -> 7
7 -> 8
}
```

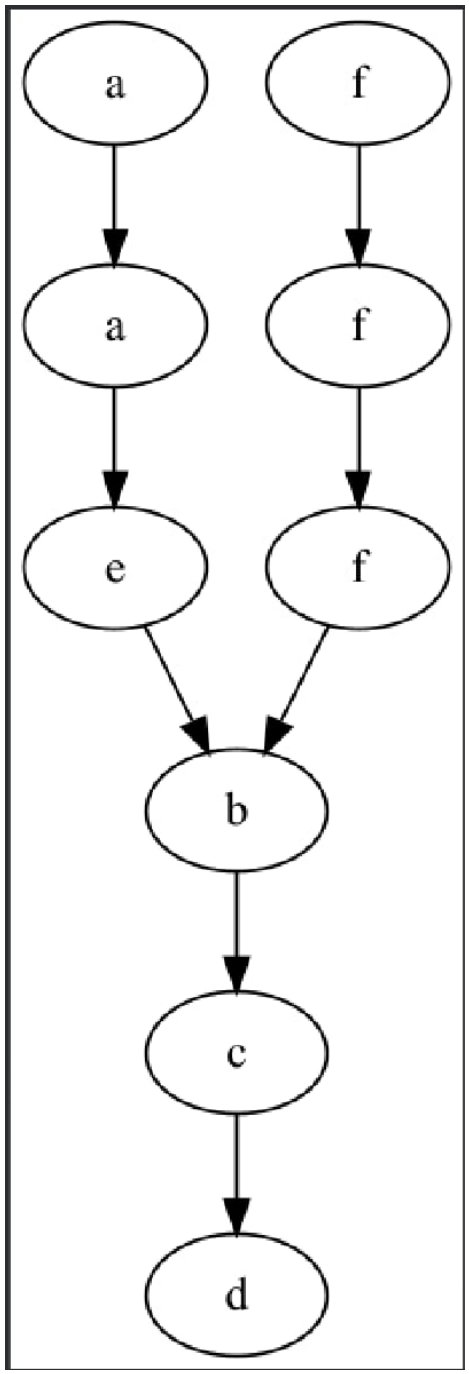
plik case2.txt

```
Dependency Relations (D):
(b, a) (b, b) (b, c) (b, d) (b, e) (b, f) (a, a) (a, b) (a, c)
(a, d) (a, e) (e, a) (e, b) (e, c) (e, d) (e, e) (c, a) (c, b)
(c, c) (c, d) (c, e) (c, f) (d, a) (d, b) (d, c) (d, d) (d, e)
(d, f) (f, b) (f, c) (f, d) (f, f)

Independent Relations (I):
(f, a) (f, e) (e, f) (a, f)

FNF: (af)(af)(fe)(b)(c)(d)

digraph g {
1[label=a]
2[label=f]
3[label=a]
4[label=e]
5[label=f]
6[label=f]
7[label=b]
8[label=c]
9[label=d]
1 -> 3
2 -> 5
3 -> 4
4 -> 7
5 -> 6
6 -> 7
7 -> 8
8 -> 9
}
```

Wizualizacje grafów z plików .dot



Rysunek 1: Wizualizacja grafu wygenerowanego na podstawie case1.txt



Rysunek 2: Wizualizacja grafu wygenerowanego na podstawie case2.txt