Tema LFA 2022-2023
Sincronizarea automatelor

Decembrie 2022

Rezumat

Tema constă în implementarea, într-un limbaj de programare la alegere, a unui algoritm de sincronizare a unui automat fără informație completă despre starea curentă.

Cuprins

1 Specificații temă

1.1 Specificații program

1.1.1 Rularea

Programul primește un singur argument în linia de comandă, numele problemei rezolvate.
Numele problemei poate fi:

- accessible (vezi 2.1)
- sync (vezi 2.2)

Pentru a permite apelarea executabilului cu argumente, regula de run din Makefile trebuie să aibă următoarea formă:

```
run: build
	./<exec> $(problem)
```

Exemplu de rulare:

```
make run problem=sync <input
```

1.1.2 Bonus

Pentru bonus, va exista o regulă de make separată, *labyrinth*, care va transforma intrarea de bonus în intrare compatibilă cu sincronizarea.

```
labyrinth: build
	./<lab-exec>
```

Checker-ul va rula următoarele comenzi, necesitând program funcțional de sincronizare pentru a acorda punctajul (vezi 2.2).
Exemplu de rulare:

```
make labyrinth <input-lab >output-lab
make run problem=sync <output-lab
```

1.1.3 Intrări

1. Prima linie conține trei numere n, m și s, unde n e numărul de stări, m e numărul de simboluri și s e numărul de stări sursă.
2. Următoarele n linii conțin câte m numere, reprezentarea ca matrice a funcției de tranziție, liniile reprezentând stările, coloanele simbolurile
3. Următoarea linie conține s stări

Restricții și precizări:

- $1 < n \leq 2^{13}$
- $0 < m \leq 2^{13}$
- Stările sunt numerotate de la 0 la $n - 1$

- Simbolurile sunt numerotate de la 0 la m; pentru simplitate, nu folosim simbolurile sau alfabetul explicit, folosim doar indicii
- $0 \leq s \leq n$ (vezi 2.2, 2.3)
- Dacă $s = 0$, atunci e echivalent cu $s = n$, urmat de toate n stările.

Intrările se consideră corecte.

1.1.4 Intrări (bonus)

Un labirint este o matrice de dimensiune $l \cdot c$, care conține celule care pot fi separate de pereți.

Specificații pentru intrarea care descrie structura labirintului

1. Prima linie conține trei numere, l, c și r unde l e numărul de linii, c e numărul de coloane, r e numărul de roboți
2. Următoarele l linii conțin câte c numere, o matrice a codificărilor celulelor
3. Următoarea linie conține r perechi de numere l_i c_i separate prin spații: robotul i se află la coordonatele (l_i, c_i)

Restricții:

- $1 < l, c \leq 2^8$
- $1 < r \leq l \cdot c$
- $0 \leq l_i \leq l$
- $0 \leq c_i \leq c$

O celulă are patru pereți, fiecare reprezentat pe un bit: est (2^0), nord (2^1), vest (2^2), sud (2^3). Informația despre configurația pereților unei celule este un număr pe patru biți pe care programul îl va primi în baza 10.

Exemple:

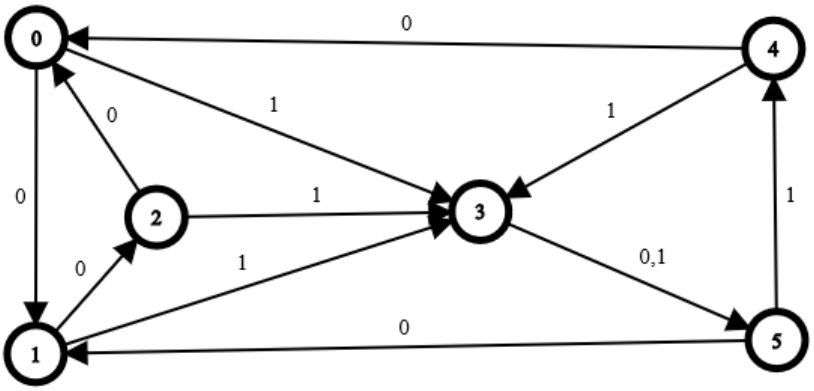
- $12 = 1010_{(2)}$: est(0), nord(1), vest(0), sud(1), o cameră cu pereți la nord și sud
- $0 = 0000_{(2)}$: est(0), nord(0), vest(0), sud(0), o cameră fără pereți

1.1.5 Ieșiri

Programul va afișa la ieșirea standard răspunsul la problema *problem* conform cerințelor din secțiunea 2, câte un element pe linie.

1.1.6 Exemplu intrări/ieșiri

Fie următorul automat:



Intrare:

```
6  2  4
1  3
2  3
0  3
5  5
0  3
1  4
0  2  3  4
```

Ieșire(*accessible*):

```
0
1
2
3
4
5
```

Nu e nimic special, toate stările sunt accesibile

Ieșire(*sync*):

```
0
0
1
```

O să urmărim fiecare stare sursă prin automat pentru fiecare simbol din secvență:

	0	2	3	4
0	1	0	5	0
0	2	1	1	1
1	3	3	3	3

Secvența 001 este secvență de sincronizare pentru că automatul ajunge în aceeași stare, 3, din orice stare sursă ar pleca

1.1.7 Suport

Pentru această temă nu există schelet de cod, dar vi se pune la dispoziție un model de Makefile.

1.1.8 Versiuni

Mașina de test are instalat Ubuntu 22.04 LTS și următoarele versiuni:

- make: 4.3
- gcc/g++: 11.3.0
- clang: 14.0.0
- javac: 11.0.17
- python: 2.7.18, 3.11
- ghc: 8.8.4

1.1.9 Limbaje acceptate

Limbajele acceptate inițial sunt C, C++, Java, Python și Haskell, urmând ca ulterior lista să poată fi extinsă, dacă este cazul.

2 Cerință

Să se implementeze un program care primește la intrare reprezentarea unui automat și găsește secvențe de sincronizare în cazul în care există.

2.1 Stări accesibile (accessible)

La primirea *problem*=`accessible`, programul va afișa toate stările accesibile din stările sursă.

Fie K' mulțimea stărilor sursă (vezi 1.1.3).

Notăm cu $A = \mathcal{A}(K', \delta)$ mulțimea stărilor accesibile din K'.

$$\forall q \in K, (q \in A \Leftrightarrow (\exists w \in \Sigma^*, \exists p \in K', (p, w) \vdash^* (q, e)))$$

În limbaj natural, numim o stare q accesibilă dacă și numai dacă există un cuvânt w cu care la intrare automatul să ajungă dintr-o stare $p \in K'$ în q.

2.2 Sincronizare (synchronize)

La primirea *problem*=`synchronize`, programul va afișa un cuvânt care sincronizează δ.

Automatele sunt folosite pentru a modela sisteme cu număr finit de stări interne.

Pentru că realitatea nu este simplă sau plăcută, se poate întâmpla ca starea curentă să nu fie cunoscută, de exemplu în cazul unui sistem haotic sau al unui agent extern care modifică starea internă a sistemului.

Scenarii:

1. Fie un sistem care procesează obiecte implicate în cel puțin un pas de cădere liberă. Căderea liberă este un proces greu de controlat sau prezis.

   Dat fiind faptul că se știe în ce stare/orientare e obiectul, există o secvență de acțiuni asupra lui care să îl aducă într-o stare cunoscută în care să se poată acționa determinist asupra lui (etichetare, imprimare, aplicare de piese)?

2. Fie un automat de cafea cu display defect și un student obosit care l-a văzut pe Gigel acționând asupra automatului fără să ia cafeaua.

   Presupunând că studentul știe cum funcționează automatul, există o secvență de butoane care să îi dea o cafea fără să fie prea dulce după cum se știe că îi place lui Gigel?

3. Fie o situație în care ai nevoie de telefon, dar fără să poți fi atent la ecran, situații normale: film, condus, sesiune.

   Există o secvență de acțiuni care să se asigure că ecranul este pornit în cazul în care e nevoie, dar fără să necesite atenție?

4. Fie un dezvoltator de jocuri care pregătește mediul dinaintea unui moment important al unui joc.

   Există elemente de design (obstacole, inamici, indicii subtile) care să îl facă pe jucător să reacționeze în așa fel încât să intre în scena următoare în modul în care consiederă autorul că ar fi mai bună experiența fără a forța în mod brutal mișcarea camerei?

5. Fie un fișier sursă deschis în vim care are sau nu indentare corectă.

   Există o comandă pe care o poți pune într-un macro, dicta cuiva sau tasta printr-o conexiune ssh cu latență mare astfel încât la final fisierul să fie salvat corect indentat? Da: gg=G:w

Procesul prin care automatul este adus într-o stare cunoscută se numește sincronizare.

Fie K o mulțime de stări, Σ un alfabet și $\delta : K \times \Sigma \to K$ o funcție de tranziție.

$$\delta \text{ admite sincronizare} \Leftrightarrow \exists w \in \Sigma^*, \exists q \in K, \forall p \in K, (p, w) \vdash^* (q, e)$$

În limbaj natural, spunem că δ admite sincronizare dacă și numai dacă există un cuvânt care dat la intrare duce automatul în aceeași stare, indiferent din ce stare a pornit.

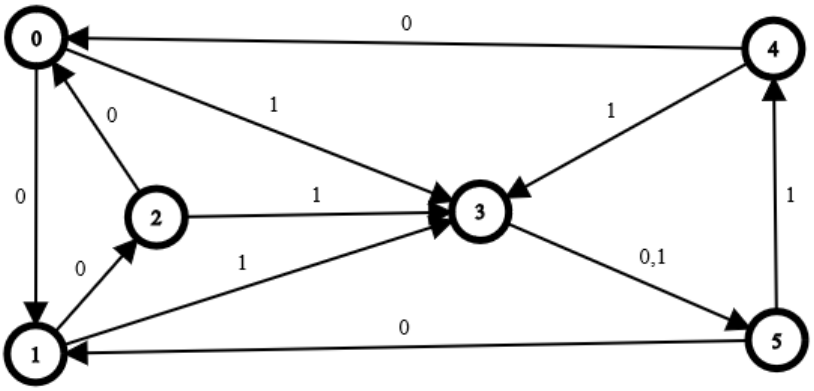
Există funcții de tranziție pentru care nu există cuvânt de sincronizare, dar care pot ajunge într-o stare cunoscută presupunând că starea sursă este într-o anumită submulțime. Un exemplu ar fi orice automat cu mai multe stări de eroare.

Fie $K' \subseteq K$, o submulțime a lui K.

$$\delta \text{ admite sincronizare din } K' \Leftrightarrow \exists w \in \Sigma^*, \exists q \in K, \forall p \in K', (p, w) \vdash^* (q, e)$$

Este foarte similar cu cazul anterior, dar stările sincronizate sunt doar cele din K', nu toate din K.

Exemplu de sincronizare
Fie un automat cu următoarele tranziții



δ admite sincronizare pentru că secvența 001 duce automatul din orice stare în starea 3 (vezi 1.1.6.

2.3 Labirint (Bonus)

Se dă un labirint codificat *lab* ca în sectiunea 1.1.4.

1. În interiorul labirintului sunt r roboți.
2. Obiectivul roboților este să se întâlnească în labirint, adică să ocupe simultan aceeași celulă.
3. Roboții nu pot comunica între ei, dar vor executa instrucțiuni.
4. O instrucțiune este executată de toți roboții simultan, în cazul în care pot face asta.
5. Instrucțiunile sunt direcții în care să se miște roboții, reprezentate ca numere în felul următor: est = 0, nord = 1, vest = 2, sud = 3.
6. Dacă un robot primește o instrucțiune de a se deplasa în direcția unui perete, rămâne în celula curentă.

Codificați labirintul ca un automat și configurația roboților ca parametri pentru o problemă de sincronizare.

Programul apelat de regula `make labyrinth` va primi la intrare codificarea *lab* și va scrie la ieșire codificarea automatului și parametrii de sincronizare astfel încât secvența de sincronizare calculată pe ieșirea programului să ducă toți roboții în aceeași celulă. Se garantează că există o soluție pentru teste.

2.4 Conținutul arhivei

Arhiva trebuie să conțină:

- surse, a căror organizare nu vă e impusă

- un fișier Makefile care să aibă un target de `build` și unul de `run`
- un fișier README care să conțină linii de maxim 80 de caractere în care să descrieți sumar reprezentarea și algoritmii aplicați. Cu cât mai scurt, cu atât mai bine!

Arhiva trebuie să fie zip. Nu rar, 7z, ace sau alt format ezoteric. Fișierul Makefile și fișierul README trebuie să fie în rădăcina arhivei, nu în vreun director.

Fișierul trebuie să se numească README, nu readme, ReadMe, README.txt, readme.txt, read-me.doc, rEADME, README.md sau alte variante asemănătore sau nu.

Nerespectarea oricărui aspect menționat mai sus va duce la nepunctarea temei.

3 Elemente de teorie

3.1 Automate orientabile

Un automat este orientabil (sau monoton) dacă există o ordine ciclică a stărilor care este păstrată de toate tranzițiile automatului (pentru mai multe detalii consultați [3](Pagina 22) și [4], secțiunea Definitions and Lemmas).

3.2 Sincronizare

Pentru mai multe detalii legate de sincronizare incluzând descrierea unui algoritm puteti consulta [2] (Algorithm 2, Theorem 1.14, 1.15) și [4], secțiunea Definitions and Lemmas si Theorem 1.

4 Checker

Vi se pune la dispoziție un checker cu teste publice, care dă punctaje de până la 125(100 + 25 bonus).

Punctajele sunt distribuite astfel:

- stări accesibile - 30p
- sincronizare - 70p
- bonus (labirint) - 25p

Timp de lucru estimat post procesare enunț: 4-8h.

Deadline: 20 Ianuarie 2023 , 23:59. Upload-ul va rămâne deschis pâna la ora 09:59 a doua zi.

Bibliografie

[1] Laborator 1 SO: Makefile

[2] Sven Sandberg, Homing and Synchronizing Sequences

[3] Mikhail V. Volkov, Synchronizing Automata and the Cerny Conjecture

[4] David Eppstein, Reset Sequences for Monotonic Automata