Programiranje I: 3. izpit

28. avgust 2024

Čas reševanja je 120 minut. Veliko uspeha!

1. naloga

a) Napišite predikat `obrnljiva : (int * int) * (int * int) -> bool`, ki za dano dvodimenzionalno matriko pove, ali je obrnljiva, torej da njena determinanta ni enaka 0. (Determinanta matrike $((a,b),(c,d))$ je enaka $ad - bc$.)

```
# obrnljiva ((3, 4), (0, 0));;
- : bool = false
# obrnljiva ((1, 0), (0, 1));;
- : bool = true
```

b) Napišite funkcijo `natanko_en : int option -> int option -> int option`, ki sprejme morebitna elementa ter vrne element natanko tedaj, ko dobi ravno enega.

```
# natanko_en (Some 1) (Some 2);;
- : int option = None
# natanko_en (Some 1) None;;
- : int option = Some 1
# natanko_en None (Some 2);;
- : int option = Some 2
# natanko_en None None;;
- : int option = None
```

c) Napišite funkcijo `razlika_aritmeticnega : int list -> int option`, ki pove, ali je zaporedje števil aritmetično. Če je, vrne razliko, sicer pa `None`. Zaporedja z enim elementom ali manj niso aritmetična.

```
# razlika_aritmeticnega [];;
- : int option = None
# razlika_aritmeticnega [1; 5; 9];;
- : int option = Some 4
# razlika_aritmeticnega [3; 4; 5; 2];;
- : int option = None
```

d) Napišite funkcijo `filtriraj : 'a list -> bool list -> 'a list`, ki sprejme seznam elementov in seznam logičnih vrednosti. Vrne naj seznam tistih elementov prvega seznama, pri katerih se v drugem seznamu na istem mestu nahaja `true`. Funkcija naj ignorira logične vrednosti na koncu seznama, če jih je preveč. Če je seznam logičnih vrednosti prekratek, naj se obnaša, kot da so manjkajoče vrednosti `true`.

```
# filtriraj [1; 2; 3; 4; 5] [true; false; true; false; true];;
- : int list = [1; 3; 5]
# filtriraj [1; 2] [false; true; false; true];;
- : int list = [2]
# filtriraj [1; 2; 3; 4; 5] [false];;
- : int list = [2; 3; 4; 5]
```

2. naloga

Imejmo preprost procesor, sestavljen iz treh komponent:

1. pomnilnika, ki je predstavljen s tabelo celih števil `memory`,
2. kazalca na trenutni ukaz `instruction_pointer`,
3. tabele ukazov `instructions`.

Mesta v pomnilniku so oštevilčena z naslovi `#0`, `#1`, ..., možni ukazi procesorja pa so:

- `INC a`, ki za 1 poveča vrednost na mestu `a` v pomnilniku,
- `DEC a`, ki za 1 zmanjša vrednost na mestu `a` v pomnilniku,
- `COPY a1 a2`, ki na mesto `a2` shrani vrednost, shranjeno na mestu `a1`,
- `WRITE x a`, ki na mesto `a` shrani število `x`,
- `JMP ip`, ki kazalec ukaza nastavi na `ip`,
- `JMPZ a ip`, ki kazalec ukaza nastavi na `ip`, če je na mestu `a` v pomnilniku shranjena ničla.

Program se konča, ko kazalec ukaza skoči iz tabele ukazov. Na primer, program, ki na mesto `#2` zapiše vsoto števil na mestih `#0` in `#1`, bi napisali z zaporedjem ukazov, ki najprej `#0` prekopira v `#2`, `#1` pa v `#3`, nato pa toliko časa povečuje `#2` in zmanjšuje `#3`, dokler `#2` ne doseže 0:

```
0 COPY #0 #2
1 COPY #1 #3
2 JMPZ #3 6
3 INC #2
4 DEC #3
5 JMP 2
```

Stanje procesorja v OCamlu predstavimo s tipi:

```
type address = Address of int

type instruction =
  | INC of address | DEC of address
  | JMP of int | JMPZ of address * int
  | COPY of address * address | WRITE of int * address

type state = {
  instructions : instruction array;
  instruction_pointer : int;
  memory : int array;
}
```

pri čemer v tipu `address` uporabimo konstruktor zato, da ne pride do zmešnjave s tipom `int`.

a) Sestavite funkcijo `increase_instruction_pointer : state -> state`, ki za 1 poveča kazalec ukaza.

b) Sestavite funkciji `read_memory : state -> address -> int`, ki prebere vrednost, shranjeno na danem mestu pomnilnika, ter `write_memory : int -> address -> state -> state`, ki na dano mesto v pomnilniku zapiše dano vrednost. Pomnilniško tabelo prvotnega stanja naj funkcija pusti nespremenjeno, vrne pa naj stanje s posodobljeno kopijo prvotne tabele.

c) Sestavite funkcijo `run_instruction : state -> instruction -> state`, ki na danem stanju izvede dani ukaz ter vrne novo stanje. Pri tem ne pozabite, da razen ob skokih po vsakem izvedenem ukazu povečamo kazalec ukaza. Tudi tu pomnilniško tabelo prvotnega stanja pustite nespremenjeno in v končnem stanju vrnite njeno ustrezno posodobljeno kopijo.

d) Sestavite funkcijo `run_until_end : state -> int array`, ki na procesorju v danem stanju toliko časa izvaja ukaze, dokler kazalec ukazov ne skoči iz tabele. Ko se to zgodi, vrnite končno stanje pomnilnika.

3. naloga

Nalogo lahko rešujete v Pythonu ali OCamlu.

Babica in dedek sta na njivi posejala repo. Iz vseh zrn je zrasla samo ena ogromna repa, težka 219.01 kg. Dedku (80 kg) je samemu ni uspelo izpuliti, zato je na pomoč poklical babico (65 kg). Naprezala sta se in naprezala, pa brez uspeha. Mimo je prišel vnuk (42 kg) in začel vleči tudi on. Čez nekaj časa je mimo priskakljala še njegova mlajša sestra (20 kg) in se pridružila ekipi. Mimo je prišel tudi voliček (600 kg), ki pa mu niso dovolili, da bi pomagal, saj v pravljicah na pomoč priskoči lahko le nekdo še manjši. Za njim je pritekel psiček (12 kg), ki je zagrabil vnučko za srajco in začel vleči, pa še vedno ni šlo. Na koncu je mimo prišla še miška (0.02 kg), in s skupnimi močmi jim je uspelo izvleči repo.

Napišite funkcijo `izvleci_repo`, ki kot argument prejme seznam oseb, ki v tem vrstnem redu prihajajo mimo njive z repo. Funkcija naj vrne maso največje repe, ki bi jo lahko izvlekli, ter seznam oseb, ki pri tem sodelujejo. Vsaka oseba je predstavljena s parom imena in mase. Da zadostimo pravljičnim pravilom, mora biti vsaka naslednja oseba lažja od prejšnje, osebe pa se lahko pridružijo prejšnjim, ali pa odidejo naprej.

V zgornjem primeru bi za vhod dobili spodnji seznam:

$$\big[(\mathit{dedek}, 80),\ (\mathit{babica}, 65),\ (\mathit{vnuk}, 42),\ (\mathit{vnukinja}, 20),\ (\mathit{voliček}, 600),\ (\mathit{psiček}, 12),\ (\mathit{miška}, 0.02)\big]$$

Dedkova ekipa lahko izvleče repo s težo kvečjemu 219.02 kg, če pa bi repo vlekli voliček, psiček in miška, pa bi lahko izvlekli repo z maso 612.02 kg. Na seznamu

$$\big[(A, 5),\ (B, 2),\ (C, 4),\ (D, 3),\ (E, 1),\ (F, 2),\ (G, 1)\big]$$

pa je optimalna izbira $[A, C, D, F, G]$ s skupno maso 15.