Programiranje I: 2. izpit

7. februar 2022

Čas reševanja je 120 minut. Veliko uspeha!

1. naloga

a) Napišite predikat `je_idempotent : (int * int) * (int * int) -> bool`, ki za dano dvodimenzionalno matriko P pove, ali je idempotentna, torej da velja $P^2 = P$.

```
# je_idempotent ((1, 0), (0, 1));;
- : bool = true
# je_idempotent ((3, -6), (1, -2));;
- : bool = true
# je_idempotent ((1, -6), (1, -2));;
- : bool = false
```

b) Napišite funkcijo `produkt : int list -> int`, ki izračuna produkt neničelnih elementov v seznamu. Če v seznamu ni neničelnih elementov, naj bo produkt enak 1. Funkcija naj bo repno rekurzivna.

c) Napišite funkcijo `stalin_sort : int list -> int list`, ki dani seznam "uredi" tako, da iz njega zaporedoma odstrani vsak element, ki ni večji od vseh svojih predhodnikov.

```
# stalin_sort [6; 3; 10; 5; 16];;
- : int list = [6; 10; 16]
```

d) Collatzovo zaporedje dobimo $a_1, a_2, \ldots$ dobimo na sledeči način. Če je a_n sodo, potem je naslednji člen $a_{n+1} = a_n/2$, sicer pa je $a_{n+1} = 3n + 1$. Ko pridemo do števila 1, se ustavimo, saj bi se na tisti točki zaporedje sicer začelo ponavljati. Na primer, Collatzovo zaporedje števila 12 je enako $12, 6, 3, 10, 5, 16, 8, 4, 2, 1$.

Zaporedje lahko posplošimo tako, da si izberemo poljubni dve funkciji f in g ter predikat p, s katerim odločimo, ali bo a_{n+1} enak $f(a_n)$ ali $g(a_n)$. Napišite funkcijo `splosni_collatz` ki za dani funkciji `f, g : int -> int`, pogoj `p : int -> bool`, začetno število `z` ter končno število `k`. Funkcija naj vrne zaporedje členov splošnega Collatzovega zaporedja, dokler ne doseže iskanega končnega števila. Na primer, originalno Collatzovo zaporedje dobimo kot:

```
# let pogoj x = (x mod 2) = 0;;
# let f x = x / 2;;
# let g x = 3 * x + 1;;
# splosni_collatz f g pogoj 12 1;;
- : int list = [12; 6; 3; 10; 5; 16; 8; 4; 2; 1]

# let pogoj x = (x mod 2) = 1;;
# let f x = x - 1;;
# let g x = x / 2;;
# splosni_collatz f g pogoj 42 1;;
- : int list = [42; 21; 20; 10; 5; 4; 2; 1]
```

Pozor: ko preverjate delovanje funkcije, upoštevajte, da se pri nekaterih vhodnih podatkih ne bo nujno ustavila.

2. naloga

Računanje aritmetičnih izrazov lahko predstavimo s skladom, kjer aritmetični izraz predstavimo kot zaporedje števil in operatorjev na skladu (seznamu). Izraz `(3 + 4) / 5` predstavimo kot: `"3 4 + 5 /"`. Aritmetične izraze v taki obliki evalviramo od leve proti desni tako, da najprej s sklada vzamemo oba argumenta in operator, operator apliciramo na operanda in na sklad vrnemo rezultat. To počnemo toliko časa, dokler ni na skladu zgolj en element, ki je končni rezultat. V kolikor med izvajanjem na vrhu sklada ni pravilnih oblik elementov, ali na koncu ostane nepravilno število elementov je tekom izvajanja prišlo do napake.

```
type 'a operator = 'a -> 'a -> 'a

type 'a atom = Value of 'a | Operator of 'a operator

type 'a term = 'a atom list

type 'a result = Finished of 'a | Term of 'a term | Error

let plus = Operator ( + )

let krat = Operator ( * )

let deljeno = Operator ( / )

let primer : int term = [ Value 3; Value 4; plus; Value 5; deljeno ]
```

a) Definirajte izraza, ki ju predstavljata sledeča niza: `"1 2 + 4 - 5 *"`, `"5.3 4.6 /. 1.7 *."`. Definirajte tudi poljuben izraz, ki predstavlja neveljaven izraz (kjer bi med izvajanjem prišlo do napake)

b) Definiraje funkcijo `korak 'a term -> 'a result`, ki sprejme izraz (sklad) in poskuša narediti en korak evalvacije. Če podan izraz predstavlja končni rezultat naj vrne obliko `Finished`, če je mogoče narediti en korak naj naredi en korak in vrne stanje po opravljenem koraku v obliki `Term`, v nasprotnem primeru pa naj vrne obliko `Error`.

```
# korak primer;;
- : int result = Term [Value 7; Value 5; Operator <fun>]
# korak [Value 7; Value 5; deljeno];;
- : int result = Term [Value 1]
# korak [Value 1];;
- : int result = Finished 1
```

c) Definirajte funkcijo `izvedi : 'a term -> 'a option`, ki sprejme izraz in ga evalvira. Če kjerkoli med izvajanje pride do napake naj funkcija vrne `None`.

d) Definirajte predikat `valid : 'a term -> bool`, ki sprejme izraz in preveri, ali je podan izraz veljaven (ali bi izvedba izraza v celoti pripeljala do rezultata, ali bi vmes prišlo do napake). Za vse točke naj bo funkcija repno rekurzivna in ne gre v celoti izvesti izraza (ne aplicira funkcij).

e) Očitno lahko iz seznama operatorjev in vrednosti pripravimo izraz. Vaša naloga je pripraviti funkcijo `combine : 'a list -> 'a operator list -> 'a term option`, ki sprejme seznam vrednosti in seznam operatorjev ter vrne končen izraz, če je to mogoče, ali `None`.

3. naloga

Na predvečer Prešernovega dne utrujeni natakar na večerji streže N gostom, ki v eni sami vrsti sedijo za dolgo mizo, vsak pa ima pred sabo h_i hrane in p_i pijače. Začne skrajno levo in bi rad čimprej prišel do izhoda na drugem koncu mize, vendar imajo gostje seveda svoje muhe, zato mora upoštevati sledeča pravila:

1. Ob vsakem premiku se odloči, koliko korakov na desno bo naredil.
2. Ko pride na mesto i, vso pijačo (torej p_i enot) pretoči v vrč, a ga gost takoj pošlje za h_i mest v levo, saj mora "še v miru pojesti". Ko odide, gost seveda ne poje ničesar, si pa takoj nazaj natoči pijačo, da je ima zopet p_i enot.
3. Ob taki vrnitvi na levo natakar ne pretoči nič dodatne pijače niti se ne vrača še bolj na levo, ne glede na to, kaj ima pred sabo gost na mestu $i - h_i$.
4. Nikoli se ne vrne preko levega roba mize.
5. Ob vsakem koraku (tako naprej kot nazaj) mu iz vrča iz neznanega razloga izgine ena enota pijače.
6. Če bi se slučajno zgodilo, da bi naredil korak s praznim vrčem, bi ga lahko kdo od razburjenja razbil in s tem uničil še zadnjo možnost za pobeg. Zaradi tega se z vrčem, v katerem je p pijače, za vsak slučaj premakne za največ p mest (v levo ali v desno). Na primer, s štirimi enotami lahko obišče mesto, ki ima dve enoti hrane in je oddaljeno tri mesta, ampak le, če ima to mesto vsaj eno enoto pijače (drugače bi ob premiku nazaj ostal s praznim vrčem).

Napišite funkcijo `pobegni`, ki sprejme seznam parov celih števil. Prva komponenta para pove količino hrane, druga pa količino pijače na posameznem mestu. Funkcija naj vrne vsa mesta, na katerih bi se moral natakar ustaviti, da bi se v čimmanj premikih prebil do konca mize. Vaša rešitev lahko predpostavi, da bo to vedno mogoče.

Na primer, za omizjem `[(1, 2), (1, 3), (3, 5), (5, 0), (5, 1)]` je najboljša rešitev sledeča:

1. Na indeksu `0` v vrč natoči 2 enoti ter se premakne na indeks 2, kamor pride s praznim vrčem.
2. Na indeksu 2 hitro natoči 5 enot, vendar ga gost pošlje nazaj na začetek `0`, kamor pride s 3 enotami.
3. Na indeksu `0` zopet natoči 2 enoti ter se ponovno premakne na indeks 2, kamor pride z 1 enoto.
4. Na indeksu 2 si spet natoči 5 enot, gost pa ga zopet pošlje nazaj na mesto `0`, kamor pride s 4 enotami.
5. Na indeksu `0` si še v tretje natoči 5 enot ter se premakne na indeks `1`, kamor pride s 5 enotami.
6. Na indeksu `1` si natoči 3 enote in se vrne nazaj na začetek, kamor pride s 7 enotami.
7. Zdaj ima na indeksu `0` v vrču dovolj zaloge, da lahko doseže konec mize.

Alternativno bi lahko v prejšnjem koraku namesto na `1` šel na indeks 2 in še vedno v enakem številu premikov dosegel konec mize. Vaša funkcija bi torej vrnila `[0, 2, 0, 2, 0, 1, 0]` ali `[0, 2, 0, 2, 0, 2, 0]`.