

UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Nemanja Subotić

PROGRAMSKI JEZICI ELM I ELIXIR U
RAZVOJU STUDENTSKOG VEB PORTALA

master rad

Beograd, 2020.

Mentor:

dr Milena VUJOŠEVIĆ JANIČIĆ, docent
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Filip MARIĆ, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Ivan ČUKIĆ, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Naslov master rada: Programski jezici Elm i Elixir u razvoju studentskog veb portala

Rezime: Apstrakt rada

Ključne reči: elm, elixir, ...

Sadržaj

1	Uvod	1
2	Elm	2
2.1	Uputstvo za instalaciju	3
2.2	Osnovne odlike	3
2.3	Elm kao platforma	4
2.4	Elm kao jezik	5
2.5	Elm arhitektura	20
3	Razvojno okruženje Phoenix i Elixir	24
4	Implementacija MSNR portala	25
5	Zaključak	26
	Bibliografija	27

Glava 1

Uvod

Funkcionalno programiranje kao programska paradigma nastaje 1959.godine sa pojavom LISP-a, prvog funkcionalnog programskog jezika... Elm... Phoenix i Elixir... MSNR Poral...

Glava 2

Elm

Evan Čapliki (Evan Czaplicki) je 2012. godine objavio svoju tezu „Elm: Konkurentno FRP ¹ za funkcionalne GUI-je ²” (eng. „*Elm: Concurrent FRP for Functional GUIs*”) [1] i, s ciljem da GUI programiranje učini prijatnijim, dizajnirao novi programski jezik — Elm. Na slici 2.1 prikazan je logo jezika. Elm je statički tipiziran, čisto funkcionalni programski jezik koji se kompilira, tačnije transpilira u JavaScript i namenjen je isključivo za kreiranje korisničkog interfjesa veb aplikacija. Takođe,



Slika 2.1: Logo programskog jezika Elm

Elm nije samo programski jezik već i platforma za razvoj aplikacija. Zbog svoje funkcionalne prirode i prisustva kompilatora, Elm spada među najstabilnija i najpouzdanija razvojna okruženja, a za Elm aplikacije važi da, u praksi, ne izbacuju neplanirane greške tokom izvršavanja (eng. *No Runtime Exceptions*).

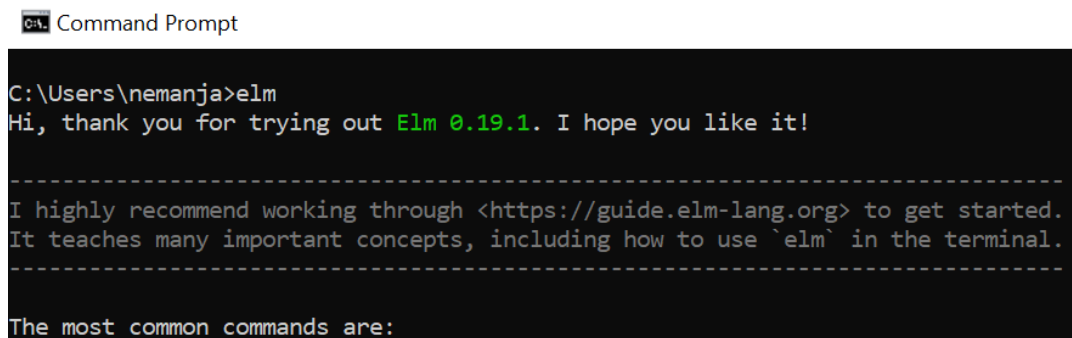
¹FRP je skraćenica za funkcionalno reaktivno programiranje

²GUI je skraćenica za grafički korisnički interfjes

2.1 Uputstvo za instalaciju

Pored želje da frontend programiranje učini prijatnijim, kreator jezika nastoji da ono bude i pristupačnije. Stoga, da bi se počelo sa korišćenjem Elma instalacija nije potrebna, dovoljeno je otići na zvaničnu veb stranicu i pokrenuti dostupan interaktivni kompilator [14], gde se može naći dosta primera, kao i vodič kroz Elm.

Za zahtevnije projekte neophodno je izvršiti instalaciju, koja je vrlo jednostavna. Potrebno je pratiti instrukcije sa zvanične stranice [5]. Na slici 2.2 prikazana je provera uspešne instalacije, koja može se izvršiti pokretanjem komande **elm** u komandnoj liniji, gde će se prikazati poruka dobrodošlice i spisak mogućih komandi o kojima će biti reč u sledećim poglavljima.



```
C:\> Command Prompt
C:\Users\nemanja>elm
Hi, thank you for trying out Elm 0.19.1. I hope you like it!

-----
I highly recommend working through <https://guide.elm-lang.org> to get started.
It teaches many important concepts, including how to use `elm` in the terminal.
-----
The most common commands are:
```

Slika 2.2: Provera uspešne instalacije Elma

Takođe, moguća je instalacija pomoću **npm**³ alata [10].

2.2 Osnovne odlike

Pored *No Runtime Exceptions*, jedna od glavnih odlika ovog jezika jeste kompilator, koji je izuzetno ugodan za rad. Mnogi programeri smatraju da Elm kompilator proizvodi najbolje poruke o greškama. Za razliku od drugih, Elm kompilator objašnjava zašto je došlo do greške i daje predloge za njihovo rešavanje, a takođe nema kaskadnih poruka. Kreator se vodio razmišljanjem da kompilator treba da bude asistent, ne samo alat.

Elm koristi svoju verziju *virtualnog DOM*⁴-a, koncepta koji se koristi u mnogim frontend okruženima. Ideja je da se u memoriji čuva „virtualna” reprezentacija

³npm — Node Package Manager predstavlja alat za upravljanje paketima u JavaScript programskom jeziku

⁴DOM — Obejktni model dokumenta (eng. *Document Object Model*)[11]

korisničkog interfejsa na osnovu koje se ažurira „stvarni” DOM. Još jedna bitna karakteristika Elma je nepromenljivost podataka, što znači da se jednom definisani podaci ne mogu više menjati. Direktna posledica nepromenljivosti podataka je veoma brzo renderovanje HTML-a, jer se poređenja u virtuelnom DOM-u mogu vršiti po referenci. Verzija Elm 0.17 imala je najbrže renderovanje u poređenju sa tadašnjim aktuelnim verzijama popularnih okruženja.

Elm se može integrisati i u postojeće JavaScript projekte za implementaciju pojedinačnih komponenti. Takođe, moguća je i komunikacija između Elma i JavaScripta.

2.3 Elm kao platforma

Elm sa sobom donosi niz alata (tabela 2.1) i Elm okruženje (eng. *Elm Runtime*), koji su neophodni za razvoj i izvršavanje aplikacija. Elm kôd se nalazi u datotekama sa *.elm* ekstenzijom i prilikom kompilacije kreira se jedna izlazna *.js* datoteka. U izlaznoj datoteci se pored prevedenog koda iz ulaznih *.elm* datoteka nalaze i (*runtime*) funkcije iz Elm okruženja potrebne za izvršavanje programa.

Alati	Kratat opis
repl	Pokretanje interaktivne sesije (eng. <i>Read-Eval-Print-Loop</i>)
init	Inicijalizacija projekta
reactor	Pokretanje lokalnog servera
make	Upotreba kompilatora
install	Preuzimanje paketa
diff	Prikazivanje razlika između različitih verzija istog paketa
bump	Određivanje broja naredne verzije paketa
publish	Publikacija paketa

Tabela 2.1: Elm alati komandne linije

Kao zaseban jezik Elm ima i zaseban sistem za upravljanje paketima. Pokretanjem komande **elm init** kreira se prazan *src* direktorijum i datoteka *elm.json*, u kojoj se pored informacije o tipu projekta (aplikacija ili paket), Elm verzije i liste direktorijuma sa kodom, nalazi i spisak paketa koji se koriste u projektu. Dodavanje novog paketa se vrši pomoću komande **elm install naziv-paketa**. Svi paketi nalaze se na <https://package.elm-lang.org/>, nazivi paketa su oblika *autor/ime-paketa*.

Kompilacija se vrši naredbom **elm make <jedna-ili-više-elm-datoteka>**, ukoliko se ne navede izlazna datoteka pomoću argumenta *--output* generisaće se *index.html*

datoteka sa prevedenim JavaScript kodom. Ostali argumenti ako i više informacija o drugim alatima može se videti pomoću naredbe **elm naziv-alata --help**

2.4 Elm kao jezik

Kao i većina funkcionalnih jezika, Elm se zasniva na ML programskom jeziku, a budući da su u Haskell-u napisani Elm kompilator i ostali alati, Haskell je morao ostaviti uticaj i na sam jezik.

„Rekao bih da je Elm ML sa sintaksom poput Haskell-a. Ako poredimo semantiku, Elm je dosta sličniji OCaml-u i SML-u.” *Evan Czaplicki [2]*

ML (eng. *Meta Language*)[6] je statički tipiziran programski jezik opšte namene koji je razvio Robin Miler 1978. godine na Univerzitetu u Edinger. Nastao je pod uticajem LISP-a i ISWIM-a (eng. *If you See What I Mean*) jezika, pripada funkcionalnoj i imperativnoj paradigmi. Osnovna karakteristika jeste uvođenje automatskog zaključivanja tipova, a odlikuje ga i poklapanje obrazaca, Karijeve funkcije, poziv po vrednosti i posedovanje sakupljača otpadaka. ML nije čist funkcionalan jezik i nema ugrađenu podršku za lenjo izračunavanje. U porodicu ML jezika, između ostalih, spadaju i **Standard ML**, **OCaml** i **F#**.

Haskell[4] je čist funkcionalni programski, naziv je dobio po matematičaru i logičaru Haskellu Bruks Kariju (Haskell Brooks Curry). Haskell je strogo tipiziran, poseduje automatsko zaključivanje tipova i lenjo izračunavanje. Jezik je opšte namene, pruža podršku za paralelno i distirbuirano programiranje. Haskell omogućava manje grešaka i veću pouzdanost kroz kraći, čistiji i održiviji kôd.

Osnovni tipovi podataka

Osnovni tipovi podataka u Elmu su **Char**, **String**, **Bool**, **Int** i **Float**. U Listingu 1 prikazani su osnovni tipovi korišćenjem interpretera (**elm repl**), budući da i Elm poseduje zaključivanje tipova, nakon izračunate vrednosti unetog izraza ispisuje se tip. U konkretnom primeru broj 42 se može posmartati i kao tip **Int** i kao tip **Float**, pa interpreter vraća **number** kao tip, iako **number** nije tip podataka.

Tip **Char** služi za predstavljanje unikod (eng. *unicode*) karaktera. Karakteri se navode između dva apostorfa (**'a'**, **'0'**, **'\t'...**), a moguće je koristiti i unikod zapis **'\u{0000}'** - **'\u{10FFFF}'**.

```
> 'Z'
'Z' : Char
> "Zdravo!"
"Zdravo!" : String
> True
True : Bool
> 42
42 : number
> 42 / 10
4.2 : Float
> 42 // 10 --celobrojno deljenje
4 : Int
```

Listing 1: Osnovni tipovi podataka prikazani u interpreteru

Za razliku od Haskell-a, gde je **String** zapravo lista karaktera, u Elmu je poseban tip i predstavlja sekvencu unikod karaktera. Sekvenca se navodi između jednostrukih ili trostrukih navodnika.

```
> "\t String u jednom redu: escape navodnici \"Zdravo!\""
"\t String u jednom redu: escape navodnici \"Zdravo!\"" : String
>
> """String u više redova
  sa "navodnicima"! """
"String u više redova\n  sa \"navodnicima\"! " : String
```

Listing 2: Stringovi

Tip **Bool** predstavlja logički tip i može imati vrednost **True** ili **False**.

Tip **Int** se koristi za prikazivanje celih brojeva. Siguran opseg vrednosti je od -2^{31} do $2^{31} - 1$, van toga sve zavisi od cilja kompilacije. Ukoliko je cilj kompilacije JavaScript (što je trenutni slučaj), opseg se proširuje na -2^{53} do $2^{53} - 1$ u nekim operacijama, što ne bi važno ukoliko bi se, nekada kasnije, umesto JavaScript koda generisao WebAssembly, tada bi postojalo prekoračenje celih brojeva (eng. *integer overflow*). Vrednosti se mogu navoditi i u heksadecimalnom obliku (0x2A, -0x2b).

Tip **Float** služi za predstavljanje brojeva u pokretnom zarezu po standardu *IEEE 754*. Vrednosti se mogu navoditi i pomoću eksponencijalnog zapisa, a decimalna tačka se mora nalaziti između dve cifre. Takođe, u skup vrednosti spadaju NaN i Infinity (listing 3) .

```
> 1e3
1000 : Float
> 0/0
NaN : Float
> 1/0
Infinity : Float
```

Listing 3: Brojevi u pokretnom zarezu

Osnovni operatori

Kod aritmetičkih operacija, operatori `+`, `-`, `*` se mogu koristiti sa realnim i celim brojevima, dok imamo posebne operatore za deljenje (`/` i `//` - listing 1). Elm ne podržava implicitne konverzije tipova, pa prilikom sabiranja celog broja sa realnim, bez eksplicitne konverzije, kompilator prijavljuje grešku. Postoji još i eksponencijalni operator `^`, a za celobrojno deljenje sa ostatkom koriste se funkcije `modBy` i `remainderBy`.

```
> toFloat (9 // 3) + 3.2
6.2 : Float
> 9 // 3 + round 3.2
6 : Int
> 9 // 3 + 3.2 -- TYPE MISMATCH error
```

Listing 4: Konverzija tipova

Elm pruža `&&` i `||` logičke operatore kao i funkcije za negaciju `not` i ekskluzivno ili `xor`. Operator `&&` ima viši prioritet od `||`, oba su levo asocijativna i lenjo izračunljiva. Od operatora poređenja `==`, `/=`, `<`, `>`, `>=` i `<=` jedino operator različitosti (`/=`) ima drugačiju od uobičajene. Pored navednih, Elm podržava i `++` operator konkatencije stringova i listi.

```
> not (1 + 1 /= 2) && 2 + 2 <= 5 || 1^0 == 0^1
True : Bool
> 2^6 - 0x100 / 4 * (1 + 2)
-128 : Float
> "Spojen " ++ "string!" == "Spojen string!"
True : Bool
```

Listing 5: Operatori

Komentari

Komentari se mogu navoditi na dva načina:

- Korišćenjem `--` za linijske komentare
- Navođenjem teksta između `{-` i `-}` za komentare u više redova.

Funkcije

Sintaksa za definisanje funkcija je veoma jednostavna i prikazana je u listingu 2.4.

```
{-
  nazivFunkcije param1 param2 ... =
    izraz
-}
deljivSa x y =
  modBy x y == 0

dobarDan x = "Dobar dan, " ++ x ++ "!"
```

Listing 6: Funkcije

Ime funkcije obavezno počinje malim slovom, nakon čega sledi niz slova (velikih i malih), simbola `_` i brojeva. Po konvenciji, sva slova se navode u neprekidnoj sekvenci, stoga je preporučena kamilja notacija (`camelCase`). Parametri se odvajaju razmakom, dok se zagrade ne navode ni prilikom definisanja, ni pozivanja funkcije. Ipak, apikacija funkcije je levo asocijativna, pa je česta upotreba zagrada za ograđivanje izraza. Telo funkcije predstavlja jedan jedini izraz koji se izvršava prilikom pozivanja, a izračunata vrednost predstavlja povratnu vrednost funkcije. Ne koriste se vitičaste zagrade, ni naredba `return`. Izraz se, po konvenciji, piše u novom redu, ali je moguće i u istom.

Konstante

U Elmu ne postoje promenljive, jednom definisani podaci se ne mogu promeniti, ali je moguće definisati konstante. Često se u literaturi definisanje konstanti naziva *imenovanjem vrednosti izraza* i ne dovodi se u vezu sa funkcijama, ali se konstante mogu posmartati kao *konstantne funkcije*, koje se izvrše tokom kompilacije. Definišu se kao i funkcije, ali bez parametara 7.

Anonimne funkcije

Anonimne funkcije se definišu slično kao i regularne, umesto imena navodi se simbol `\` koji predstavlja grčko slovo lambda - λ , dok se `->` koristi umesto znaka jednakosti.

```
> broj3 = 3
3 : number
> (\x y -> x + y) broj3 4
7 : number
```

Listing 7: Primer anonimne funkcije

Funkcije u Elmu mogu prihvatati funkcije kao parametre i vraćati funkcije kao povratne vrednosti, što ih čini funkcijama višeg reda. Nije moguće navoditi podrazumevane vrednosti parametara, kao ni preopterećivanje funkcija.

Moduli

Moduli se koriste za grupisanje funkcija u logičke jedinice i kreiranje imenskih prostora (eng. *namespace*). Svaki modul predstavlja jedanu *.elm* datoteku, koja se mora zvati isto kao i modul, dok ime modula mora počinjati velikim slovom. Za definisanje modula koristi se ključna reč `module` nakon koje sledi ime modula, ključna reč `exposing` i lista funkcija kojima se može pristupiti van modula.

```
module Krug exposing (povrsina, obim)
-- module Krug exposing (..) - otkrivanje svega iz modula
pi = 3.14

povrsina r =
    naKvadrat r * pi

obim r =
    2 * r * pi

naKvadrat x =
    x * x
```

Listing 8: Primer modula

Da bi se modul iz listinga 8 koristio u interpreteru (`elm repl`), prvo je potrebno inicijalizovati elm projekat (`elm init`) i u `src` folderu napraviti *Krug.elm* datoteku

sa prikazanim sadržajem. Zatim, u interpreteru naredbom `import` uvesti modul. Načini korišćenja funkcija iz modula *Krug* prikazani su u listingu 9.

```
import Krug                                -- Krug.obim Krug.povrsina
import Krug as K                          -- K.obim K.povrsina

import Krug exposing (obim)                -- obim, Krug.povrsina
import Krug exposing (..)                 -- obim, povrsina
import Krug as K exposing (povrsina)      -- K.obim, povrsina
```

Listing 9: Primer korišćenja modula

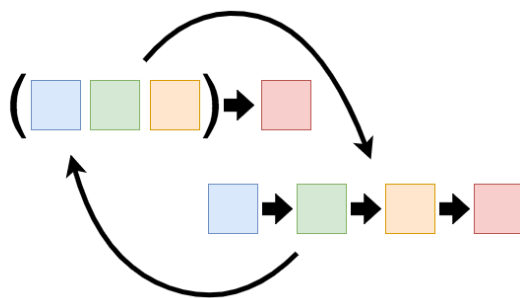
Tip funkcije

Prilikom definisanje funkcije u interpreteru ili poziva funkcije bez parametara, kao vrednost izraza vraća se `<function>` i tip funkcije. U listing-u 10 prikazano je nekoliko primera tipova funkcija u interpreteru.

```
> not
<function> : Bool -> Bool
> deljivSa
<function> : Int -> Int -> Bool
> \x y z -> x + y + z
<function> : number -> number -> number -> number
> deljivSa 3 --parcijalna aplikacija
<function> : Int -> Bool
```

Listing 10: Tipovi funkcija

U primeru funkcije `not` vidimo da je njen tip `Bool -> Bool`, što je dovoljno samorazumljivo i znači da se radi o funkciji jednog argumenta koja prihvata vrednost tipa `Bool` i vraća vrednost tipa `Bool`. U slučaju funkcija koje imaju više argumenata, tip funkcije ne mora biti toliko jasan i može se posmartati na način da poslednji tip u nizu razdvojenim strelicama (`->`) predstavlja povrati tip funkcije, dok tipovi pre njega predstavljaju tipove argumenata funkcije. Postavlja se pitanje zašto se i tipovi argumenata međusobom odvajaju strelicama. Razlog za to su Karijeve (*Curried*) funkcije u Elmu, što znači da su sve `n`-arne funkcije zapravo funkcije jednog argumenta koje kao povratnu vrednost imaju funkciju — slika 2.3. Strelica (`->`) je desno asocijativna, a zagrade se izostavljaju zbog jendostavnosti. Tako da funkcija



Slika 2.3: Regularne i Karijeve funkcije

`deljivSa` ima tip `Int -> (Int -> Bool)`, što znači da prihvata vrednost tip `Int` i vraća funkciju tipa `Int -> Bool`. Karijeve funkcije nam omogućavaju veću fleksibilnost i parcijalnu aplikaciju funkcija, odnosno vezivanje argumenata za konkretne vrednosti 10.

Anotacija tipa funkcije

Kao što je prethodno prikazano, Elm sam zaključuje tip funkcije, ali dozvoljava i korisniku da sam navede tip u liniji iznad definicije.

```
> deljivSa: Int -> Int -> Bool
| deljivSa x y =
|   modBy x y == 0
|
<function> : Int -> Int -> Bool
```

Listing 11: Anotacija tipa funkcije

Korišćenje anotacije tipova nije obavezno, ali je vrlo preporučljivo iz više razloga. Prilikom kompilacije proverava se poklapanje anotacije sa stvarnim tipom funkcije, što dovodi do lakšeg uočavanja i otklanjanja grešaka. Pored toga, anotacije predstavljaaju veoma dobar vid dokumentacije, a činjenica da kompilator uvek poredi navedeni i stvarni tip nam garantuje da je dokumentacija uvek važeća.

Funkcijski operatori

Operatore nad funkcijama možemo podeliti prema tipu, na operatore prosleđivanja i operatore kompozicije funkcija, i prema smeru u kom se primenjuju, unapred ili unazad.

Operatori prosleđivanja ili **pipe** operatori su zapravo operatori aplikacije funkcije i omogućavaju pisanje čitljivijeg koda sa manje zagrada.

- `<|` - pipe operator unazad radi isto što i aplikacija funkcije, stim što nas oslobađa pisanja zagrada. Tako da je `f <| x` identično `f (x)`
- `|>` - pipe operator inspirisan je Unix pipe-om, odatle i naziv, i služi za prosleđivanje argumenta funkciji. `x |> f` je zapravo `f (x)`

Operator kompozicije unazad `<<` zapravo predstavlja operator matematičke kompozicije funkcija `o`. Tako da se definicija kompozicije $(g \circ f)(x) = g(f(x))$ u Elmu može posmatrati kao: `(g << f) x == (\x_ -> g (f x_)) x`. Dok je operator kompozicije `>>` obrnut i simetričan je operatoru `<<` – `(f << g) x == (g >> f) x`.

Osnovne strukture podataka

Osnovne strukture podatka u Elmu čine liste, torke i slogovi.

Liste

Lista u Elmu predstavlja kolekciju u obliku jednostruko povezane liste. Elementi se navode unutar uglastih zagrada – `[]` i moraju biti istog tipa. Funkcije za rad sa listama nalaze se unutar `List` modula. Pored operatora za nadovezivanje, postoji i operator `::` koji dodaje element na početak liste.

```
> "aaa" :: ["bbb", "ccc"]
["aaa", "bbb", "ccc"] : List String
> List.map (List.member 2) [[1,2,3], [2,2], [42]]
[True, True, False] : List Bool
> [1,2]++[3,4,5] |> List.filter (\x -> modBy 2 x == 0) |> List.length
2 : Int
> List.foldl (\x y -> x + y) 0 <| List.range 1 5
15 : Int
```

Listing 12: Liste

Torke

Za razliku od listi koje mogu imati promenljivi broj elemenata istog tipa, torke predstavljaju kolekcije fiksne dužine čiji elementi ne moraju biti istog tipa. Mogu

sadržati samo dva ili tri elementa, navode se unutar običnih zagrada - () i ne mogu se ubacivati ili uklanjati elementi. Funkcije nad torkama koje imaju dva element nalaze se u modulu Tuple.

```
> (1,"2",'3')
(1,"2",'3') : ( number, String, Char )
> (1,2) == Tuple.pair 1 2
True : Bool
> Tuple.second ("nebitan", "drugi")
"drugi" : String
> Tuple.mapFirst String.length ("mapiran", 1)
(7,1) : ( Int, number )
```

Listing 13: Torke

Slogovi

Slog (eng. *Record*) predstavlja strukuru podataka koja može sadržati više vrednosti različitih tipova, pri čemu je svakoj vrednosti dodeljen naziv. Liče na objekte u JavaScript-u, čak je i sintaksa veoma slična, umesto dvotačke slog koristi znak jednakosti za dodelu naziva. Prilikom definisanja sloga, Elm kreira funkcije za pristup njegovim svojstvima. Nije moguće dodavanje, ni uklanjanje svojstava, ali je dozvoljena promena njihovih vrednosti. Zbog imutabilnosti, ne vrši se promena nad postojećim slogom već se pravi novi.

```
> pera = {ime = "Pera", prezime = "Perić", godine = 23}
{ godine = 23, ime = "Pera", prezime = "Perić" }
  : { godine : number, ime : String, prezime : String }
> {pera | prezime = "Petrović", godine = 24}
{ godine = 24, ime = "Pera", prezime = "Petrović" }
  : { godine : number, ime : String, prezime : String }
> pera.ime
"Pera" : String
> .godine pera
23 : number
> .prezime
<function> : { b | prezime : a } -> a
```

Listing 14: Slog

Pored navedenih struktura podataka, Elm pruža podršku za rad sa nizovima (`Array`), skupovima (`Set`) i rečnicima (`Dict`).

Tipske promenljive

U listingu 14 vidimo da funkcija za pristup prezimenu ima tip: `{ b | prezime : a } -> a`. Ovo znači da funkcija kao argumenat prima slog, koji može biti bilo kog tipa, ali mora imati svojstvo `prezime`, koje takođe može biti bilo kog tipa, i čiji tip je ujedno povraća tip funkcije. Promenljive `a` i `b` nazivaju se *tipske promenljive*, a prisustvo dve tipske promenljive nam govori da one mogu, ali ne moraju, predstavljati različite tipove. U konkretnom primeru `a` i `b` su uvek različitog tipa, dok u slučaju funkcije `Tuple.pair : a -> b -> (a, b)` mogu biti istog tipa. Prilikom anotacije tipova mogu se koristiti i duža imena tipskih promenljivih, a pravila imenovanja su ista kao i za funkcije.

Tipske promenljive u Elmu ukazuju na prisustvo **parametarskog polimorfizma**, jedine vrste polimorfizma u ovom jeziku.

Uslovne tipske promenljive

Za razliku od Haskell-a, Elm nema toliko složen sistem tipova i umesto tipskih klasa (eng. *typeclasses*)[3] poseduje jednostavniji koncept – *uslovne tipske promenljive*.

Uslovne tipske promenljive omogućavaju da se na određni način ograniči skup tipova koji se može koristiti u izrazima. Najčešći primer je `number`, koji dozvoljava isključivo `Int` ili `Float` tipve.

U trenutnoj verziji (*0.19.1*) postoje četiri uslovne tipske promenljive:

1. `number` - dozvoljava `Int` i `Float`
2. `appendable` - dozvoljava `String` i `List a`
3. `comparable` - dozvoljava `Int`, `Float`, `Char`, `String`, liste i torke koje sadrže `comparable` vrednosti
4. `compappend` - dozvoljava `String` i `List comparable`

Operatori i tipske promenljive

```
> (+) 1 2
3 : number
> (*)
<function> : number -> number -> number
> (++)
<function> : appendable -> appendable -> appendable
> (==)
<function> : a -> a -> Bool
> (>=)
<function> : comparable -> comparable -> Bool
```

Listing 15: Uslovne tipske promenljive

Operatori u Elmu predstavljaju funkcije koje se mogu pozivati u infiksnoj notaciji. Takođe, mogu se pozivati i u prefiksnoj ukoliko ih navedemo unutar zagrada: (+), (++) , (>=)... , dok pozivanjem bez argumenata možemo videti i kog su tipa.

U ranijim verzijama jezika bilo je moguće definisati korisničke operatore, ali je ta opcija izbačena u verziji 0.19.0., a od verziji 0.18.0 nije moguće pozivanja binarnih funkcija u infiksnoj notaciji.

Alias tipova

Prilikom definisanja funkcija koje rade nad istim strukturama podataka višestruko navodimo iste anotacije tipova, pritom anotacije podataka mogu biti predugačke, samim tim i teško čitljive. Alias tipova nam omogućavaju ponovnu upotrebu anotacija i bolju čitljivost. Definišu se pomoću ključnih reči `type alias`, nakon kojih sledi ime koje mora počinjati velikim slovom.

```
>type alias MatricaInt3 = List (List (List Int))
>
>type alias Osoba = {ime : String, prezime : String, godine : Int }
> Osoba
<function> : String -> String -> Int -> Osoba
> Osoba "Pera" "Perić" 23
{ godine = 23, ime = "Pera", prezime = "Perić" } : Osoba
```

Listing 16: Alias tipova

Prilikom kreiranja aliasa za slog kreira se i konstruktor za slog, što se može videti u listingu 16. Redosled argumenata funkcije za konstrukciju identičan je redosledu

u aliasu.

Korisnički definisani tipovi

Pored korišćenja aliasa za postojeće tipove podataka, Elm pruža mogućnost kreiranja novih tipova. Korisnički definisani tipovi se često nazivaju *unijski tipovi*, jer mogu predstavljati uniju više varijanti definisanog tipa. Definišu se ključnom reči `type`, a varijante se odvajaju simbolom `|`.

Slično kao kod aliasa tipova za slogove i ovde se kreiraju konstruktori za definisani tip.

```
> type VectorF4 = Vector4F Float Float Float Float
> Vector4F
<function> : Float -> Float -> Float -> Float -> VectorF4
>
> type StatusPrijave
  = NaCekanju
  | Greska String
  | Uspesno {id : Int, token : String}
> NaCekanju
NaCekanju : StatusPrijave
> Greska
<function> : String -> StatusPrijave
> Uspesno
<function> : { id : Int, token : String } -> StatusPrijave
```

Listing 17: Korisnički definisani tipovi

Kontorla toka

U Elmu nema izvršavanja naredbi, već samo evaluacije izraza, tako da umesto naredbi grananja imamo izraze `if` i `case`, a rekurziju umesto petlji.

Izraz `if`

Izraz `if` se može posmatrati kao ternarni operator u JavaScript-u, C++-u i mnogim drugim programskim jezicima.

Ključna reč `else` je sastavni deo izraza `if` tako da `else` „grana” uvek postoji. Mogu se navoditi i ugnježdeni izrazi `if`, pa `else if` grana predstavlja korišćenje izraza `if` nakon ključna reč `else`.

```
{-      uslov  ?  izraz1  :  izraz2  - ternarni operator
      if uslov then izraz1 else izraz2  - if izraz
-}
if x >= 0 then "pozitivan" else "negativan"
if x == 1 then x * 2 else if x == 2 then x / 2 else x
```

Listing 18: Izraz if

Izraz case

Izraz `case` predstavlja pandan `switch` naredbi u drugim programski jezicima. Mogu da rade samo nad jednim tipom vrednosti, a prilikom korišćenja izraza `case` moraju se pokriti sve mogućnosti. Za podrazumevani slučaj može se koristiti simbol `_`. Izrazi `Case` zauzimaju značajno mesto u Elmu, jer se koriste u **poklapanu obrazaca**.

```
case mesto of
  1 -> "zlato"
  2 -> "srebro"
  3 -> "bronza"
  _ -> "zahvalnica"
```

Listing 19: Izraz case

Izraz let

Budući da ne postoje blokovi naredbi, izrazi `let` nam omogućavaju da ograničimo oblast važenja — dosega (eng. *scope*) funkcija i konstanti u okviru jedne funkcije. Doprinosu boljoj čitljivosti koda, a moguće je koristiti anotacije tipova unutar njih.

```
let
  nula: Int
  nula = 0

  pozitivan: Int -> Bool
  pozitivan =
    \x -> x > nula
in pozitivan 10
```

Listing 20: Izraz let

Rekurzija

Rekurzivno definisana funkcija poziva samu sebe, čime se postiže ponavljanje koje imamo korišćenjem petlji. U nekim slučajevima, umesto rekurzije mogu se koristiti funkcije nad listama.

```
factorial n =  
  if n <= 1 then 1  
  else n * factorial (n - 1)  
  
factorialFold n =  
  List.foldl (*) 1 (List.range 1 n)
```

Listing 21: Rekurzija

Poklapanje obrazaca

Poklapanje obrazaca može se posmatrati kao pokušavanje usklađivanja (poklapanja) ulaznog podatka sa unapred definisanim obrascem. Ukoliko dođe do usklađivanja, poklopljenim vrednostima se može prisupiti putem identifikatora definisanim u obrascu. Pored spomenutih `case` izraza, u Elmu se poklapanje obrazaca može koristiti u vidu destrukcije slogova ili torki prilikom definisanja funkcija ili korišćenja `let` izraza.

Unutar `case` izraza sekvencijalno se vrši poklapanje obrazaca. Kada dodje do poklapanja izračunava se izraz dodeljen datom obrascu, ne nastavlja se sa poklapanjem. Kompilator prepoznaje ukoliko može doći do nepoklapanja nijednog obrasca i prijavljuje grešku. Takođe, greška se prijavljuje i ukoliko se navede redundantan obrazac, tj. obrazac čiji je skup vrednosti zapravo podskup skup vrednosti prethodno definisanog obrasca. Simbol `_` služi za poklapanje vrednosti koje se ne koriste, a ključna reč `as` se može koristiti ukoliko je potrebno pristupiti celom ulaznom podatku. U listingu 22 mogu se videti primeri poklapanja obrazaca.

Obrada grešaka pomoću Maybe i Result

U Elmu ne postoje `try catch` blokovi, kao ni `undefined`, `null`, `nil` i ostale slične vrednost prisutne u drugim programskim jezicima. Umesto njih koriste se `Maybe` i `Result` koji potiču iz Haskell-a, gde imamo `Maybe` i `Either`.

```
-- liste
case lista of
  [] -> "prazna lista"
  [_] -> "jedan element"
  [a,b] -> "dva elementa:" ++ a ++ " i " ++ b
  a :: _ -> "više od dve elemenata, prvi je: " ++ a

-- unijski tipovi
case prijava of
  NaCekanju -> "Molimo za strpljive"
  Greska poruka -> "Došlo je do grške: " ++ poruka
  Uspesno {id} -> "Uspešna prijava, id: " ++ String.fromInt id

-- torke
case tacka3D of
  (0, 0, 0) -> "centar"
  (0, _, _) -> "na x-osi"
  _ -> "van x-ose"

-- destrukcija
let
  (x,_,_) = tacka3D
in "x koordinata je " ++ String.fromFloat x

--nije moguće poklapanje ugnježđenih slogova
prikaziPodatke ({ime, adresa} as osoba) =
  ime ++ " " ++ osoba.prezime ++ " " ++ adresa.ulica
```

Listing 22: Poklapanje obrazaca

Maybe

Ukoliko bismo želeli da napišemo funkciju koja vraća prvi element liste, u slučaju da on postoji vratili bismo **baš** njega, a u slučaju prazne liste ne bismo vratili ništa, tj. vratili bi **ništa**. Srećom, funkcija `List.head : List a -> Maybe a` radi upravo to pa ne moramo da je pišemo i ukoliko je pozovemo **možda** name vrati prvi element.

`Maybe` se definiše kao `type Maybe a = Just a | Nothing` i može se koristiti za opcione argument, obradu grešaka i u slogovima sa opcionim svojstvima. Zapravo svuda gde očekivani podatak može, ali ne mora, postojati.

Result

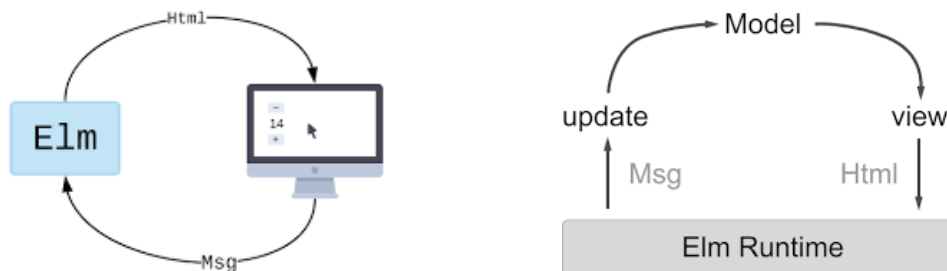
Za razliku od `Maybe`, koji bi u slučaju greške vratio `Nothing`, `Result` nam daje mogućnost pružanja dodatnih informacija o grešci. Definiše se na sledeći način:

```
type Result error value
  = Ok value
  | Err error
```

2.5 Elm arhitektura

Elm arhitektura je veoma jednostavan obrazac projektovanja veb aplikacija koji se pojavljuje nešto kasnije nakon Elma. Nastaje prirodno, tako što rani Elm programeri uočavaju iste obrasce u svom kodu, koji se mogu podeliti u tri dela:

1. Model - stanje aplikacije
2. Pogled (eng. *View*) - transformacija stanja u HTML
3. Ažuriranje (eng. *Update*) - promena stanja



Slika 2.4: Elm arhitektura

U levom delu slike 2.4 prikazano je kako radi svaki Elm program, generiše se HTML koji se prikazuje u pretraživaču, nakon čega pretraživač šalje poruku programu ukoliko se nešto dogodilo. U desnom delu slike vidimo šta se dešava unutar programa, na osnovu primljene poruke funkcija `update` kreira novi `model`, koji se prosleđuje funkciji `view`, na osnovu koje se generiše HTML.

Ovaj obrazac projektovanja ce često naziva i MVU (eng. *Model-View-Update*). Za razliku od obrazaca MVC [7] (eng. *Model-View-Controller*) i MVVM [9] (eng.

Model-View-ViewModel), koji stanje aplikacije dele na više manjih modela, u Elm arhitekturi celokupno stanje aplikacije se nalazi na jednom mestu, tj. modelu, a protok podataka kroz aplikaciju je uvek u jednom smeru.

Elm arhitektura je uticala na nastajanje velikog broja biblioteka za upravljanje stanja veb aplikacija, među kojima su najpoznatije Redux[13] i Vuex[15]. Takođe, podrška za rad sa MVU obrascem počinje da se pojavljuje i u drugim tehnologijama, jedna od njih je i .NET 5 [8].

Generisanje HTML sadržaja

Za razliku od drugih radnih okruženja koji uglavnom koriste HTML šablone, Elm za opisivanje izgleda stanice koristi funkcije. Stoga imamo funkcije za kreiranje HTML čvorova i atributa.

```
-- <button id="btn-ok" class="btn-default"> Ok </button>
node "button" [id "btn-ok", class "btn-default"] [text "Ok"]
    button [id "btn-ok", class "btn-default"] [text "Ok"]
```

Listing 23: Kreiranje HTML čvorova

Funkcija `node` iz modula `Html` predstavlja generičku funkciju za kreiranje HTML čvorova, koja kao argumente prima HTML oznaku, listu atributa i listu čvorova (deca čvora). U listingu 23 predstavljena je analogija kreiranja HTML čvora funkcijom `node` i HTML sintakse. Modul `Html` pruža veliki broj pomoćnih funkcija, koje imaju nazive po HTML oznakama (npr. `button`) i omogućavaju bolju čitljivost. Stoga se funkcija `node` koristi prilikom upotrebe korisnički definisanih elemenata ili ukoliko ne postoji pomoćna funkcija za željeni element. Funkcija `text` služi za postavljanje teksta u DOM, dok su `id` i `class` funkcije za kreiranje atributa iz modula `Html.Attributes`.

Elm program

Prilikom inicijalizacije Elm projekta se, pored paketa `elm/core` sa osnovnim funkcionalnostima i strukturama podataka i paketa `elm/html` za generisanje HTML stranica, nalazi paket `elm/browser` za kreiranje Elm programa u pretraživaču. Unutar ovog paketa, u modulu `Browser` se nalaze funkcije koje nam omogućavaju kreiranje različitih tipova programa, među kojima je i funkcija `sandbox` namenjena za

učenje osnova Elm arhitekture i omogućava bazičnu interakciju sa korisnicima, bez komunikacije sa spoljnim svetom. Primer jednostavanog Elm programa upotrebom `sandbox` funkcije prikazan je u listingu 24.

Da bi Elm okruženje znalo koja je polazna tačka programa potrebno je da se u glavnom modulu definiše i izloži konstanta `main`, dok naziv modula nije bitan. Pored funkcija iz modula `Browser`, u slučaju statičkih stranica, konstanta `main` se može definisati funkcijama iz modula `Html`, na primer:

```
main = text "Zdravo svete!"
```

Da bi se pokrenuo program potrebno je pozicionirati se u komandnoj liniji na inicijalizovani Elm projekat i pokrenuti `elm reactor`. Nakon toga otvoriti pretraživač na adresi `http://localhost:8000` i unutar `src` direktorijuma kliknuti na `Main.elm` čime se vrši kompilacija i pokretanje programa. Takođe, primer se može naći i na zvaničnom Elm vodiču[12].

U primeru jednostavnog brojača iz listinga 24 vidimo da funkcija `sandbox` kao argument prima slog sa inicijalnom vrednošću modela i funkcijama za ažuriranje i prikazivanje modela. Program počinje sa pozivanjem funkcije `view` sa parametrom `init`. Unutar funkcije `view` se pomoću funkcije za kreiranje atributa iz modula `Html.Events` mogu definisati načini slanja poruka, a kao rezultat izvršavanja nastaje virtualni DOM, na osnovu kog Elm okruženje izmenjuje stvarni DOM. Ukoliko dođe do odgovarajuće akcije korisnika, u ovom slučaju klikom na dugme, Elm okruženje generiše poruku i prosleđuje je zajedno sa modelom funkciji `update` koja kreira novi model nad kojim se poziva ponovo funkcija `view`. Elm okruženje poredi prethodni virtualni DOM sa novim i vrši minimalan broj izmena.

```
module Main exposing (main)

import Browser
import Html exposing (Html, button, div, text)
import Html.Events exposing (onClick)

main =
    Browser.sandbox { init = init, update = update, view = view }

--MODEL
type alias Model = Int

init : Model
init =
    0

-- UPDATE
type Msg
    = Increment
    | Decrement

update : Msg -> Model -> Model
update msg model =
    case msg of
        Increment ->
            model + 1

        Decrement ->
            model - 1

-- VIEW
view : Model -> Html Msg
view model =
    div []
        [ button [ onClick Decrement ] [ text "-" ]
        , div [] [ text (String.fromInt model) ]
        , button [ onClick Increment ] [ text "+" ]
        ]
```

Listing 24: Elm program

Glava 3

Razvojno okruženje Phoenix i Elixir

Glava 4

Implementacija MSNR portala

Glava 5

Zaključak

Bibliografija

- [1] Evan Czaplicki. *Elm: Concurrent FRP for Functional GUIs*. 2012. URL: <https://elm-lang.org/assets/papers/concurrent-frp.pdf>.
- [2] Evan Czaplicki. *Github comment*. 2015. URL: <https://github.com/elm/elm-lang.org/issues/408#issuecomment-151656681>.
- [3] Learn You a Haskell - Types **and** Typeclasses. *Miran Lipovača*. URL: <http://learnyouahaskell.com/types-and-typeclasses>.
- [4] *Haskell programski jezik*. URL: <https://www.haskell.org>.
- [5] *Install Elm*. URL: <https://guide.elm-lang.org/install/elm.html>.
- [6] *ML programski jezik*. URL: [https://en.wikipedia.org/wiki/ML_\(programming_language\)](https://en.wikipedia.org/wiki/ML_(programming_language)).
- [7] *MVC obrazac*. URL: <https://en.wikipedia.org/wiki/Model-view-controller>.
- [8] *MVU obrazac u .NET 5*. URL: <https://devblogs.microsoft.com/dotnet/introducing-net-multi-platform-app-ui>.
- [9] *MVVM obrazac*. URL: <https://en.wikipedia.org/wiki/Model-view-viewmodel>.
- [10] *Npm Elm*. URL: <https://www.npmjs.com/package/elm>.
- [11] *Obejktni model dokumenta*. URL: https://en.wikipedia.org/wiki/Document_Object_Model.
- [12] *Primer Elm programa*. URL: <https://elm-lang.org/examples/buttons>.
- [13] *Redux biblioteka*. URL: <https://redux.js.org/introduction/prior-art>.
- [14] *Try Elm*. URL: <https://elm-lang.org/try>.
- [15] *Vuex biblioteka*. URL: <https://vuex.vuejs.org>.