Programski jezik F#

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

Tijana Todorov, Tamara Garibović,
David Nedeljković, Mihajlo Vićentijević
tijana.todorov710@gmail.com, t.garibovic1995@gmail.com,
dnedeljkovic710@gmail.com, mihajlovicent@gmail.com

3. april 2019

Sažetak

Dodati na kraju sazetak.

Sadržaj

1 Uvod

Dodati na kraju uvod u temu i obavezno izmeniti trenutni radni naslov.

2 Poreklo programskog jezika F#

U 2018. godini F# opisan je u dokumentaciji kao "funkcionalni programski jezik koji se pokreće na .NET platformi"[19], ali odakle je on potekao?

Istorija programskog jezika F# datira još od 1970. godine pa sve do danas. U ranim 70-im godinama na Univerzitetu u Edinburgu Robin Milner sa još nekoliko svojih kolega razvija jezik ML (Meta-Language) baziran na programskom jeziku LISP. Njegova onsnovna namena bila je pragmatičnog karaktera. Osmišljen je da pomogne u dokazivanju LCF (Logical computable functions) [16] teorema. ML jezik je koren svih jezika koji pripadaju familiji strogo tipiziranih funkcionalnih jezika, kao npr: Miranda, Haskell, Standard ML, Ocaml, EdinburghML, ReasonML, PureScript, a među njima i F#.

Sve do danas ključne ideje ove familije jezika ostaju osnova jezika F# koji se nadograđivao kasnije iz dana u dan.

Tokom 80-ih godina dolazi do velike ekspanzije u kompjuterskoj industriji. Kako su se brzo razvijali softverski alati tako su se takođe pojavljivali novi jezici i programske paradigme. U tom periodu i Microsoft doživljava veliku ekspanziju kao kompanija koja razvija operativne sisteme i aplikacije. Međutim, u ovom periodu, a najviše u kasnim 80-im godinama pojavljuje se novi, objektno-orjentisani talas razmišljanja u programiranju koji veoma utiče na razvoj softvera.

Početkom 90-ih godina, dok je Microsoft težio da održi monopol na tržištu i dok je u fokusu i dalje bila objektno-orjentisana paradigma, strogo tipizirani funkcionalni jezici bili su manje aktivni i okrenuti su ka razvitku na drugim poljima. Njihova primena uglavnom je imala ulogu u pronalaženju bagova, održavanju sistema i verifikaciji hardvera i softvera. Primeri jezika koji su se koristili za izgradnju ovakvih sistema su: Edinburgh ML, Standard ML, Ocaml, Caml Light, LISP. Takođe, neki funkcionalni jezici razvijeni su u cilju istraživanja u okviru paralelnog programiranja, kao što su Parallel ML i vrsta paralelnog Haskell-a [19].

2.1 Zašto je nastao jezik F#?

Tokom 90-ih godina Microsoft razvija .NET [12, 19] platformu za razvoj softvera. Ideja je bila da se omogući međusobna kompatibilnost više programskih jezika, odnosno da svaki jezik podržan na ovoj platformi može da koristi kôd nekog drugog jezika platforme. Glavni cilj Microsofta bio je da se na ovoj platformi podrži što veći broj jezika iz različitih paradigmi.

U okviru ovoga Don Syme razija projekat SML.NET koji je za ideju imao preusmeravanje Standard ML-a na .NET. Sistem je imao visok kvalitet, ali nije bio usvojen od strane programera. Početkom 2000-ih platforma .NET je već uveliko zaživela, ali za jezik SML.NET nije bilo većeg interesovanja. Autor je imao veliku želju da implementira strogo tipiziran funkcionalni jezik na način koji bi zainteresovao veliki broj programera. U razmatranje ulazi i jezik OCaml, ali prethodno dolazi do pokušaja implementacije Haskell-a za .NET. Ovaj pokušaj bio je samo delimično uspešan

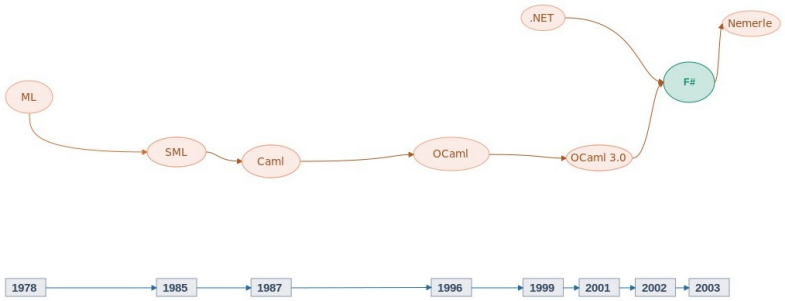
i primenjen na malim programima. Rad na daljoj implementaciji je zaustavljen.

U decembru 2001. godine Don Syme u razmatranje vraća jezik OCaml i razvija projekat Caml.NET koji će se kasnije preimenovati u F#.

Inicijalna ideja F# programskog jezika bila je jednostavna. Trebalo je da poveže prednosti OCaml programskog jezika sa prednostima .NET platforme. 2002. godine pojavljuje se prva verzija F# 0.5 koja bila je slabo primećena. Don Syme 2004. godine nastavlja intenzivno da razvija ovaj jezik, a početkom 2005. godine izbacuje prvu potpunu verziju F#-a [19]. Poslednja aktuelna verzija jezika je F# 4.1 [6].

2.2 Koji programski jezici su najviše uticali na razvoj jezika F#?

Najveći značaj za razvoj jezika F# imaju jezik SML (Standard ML) i CAML (Categorical Abstarct Machine Language) porodica jezika koja je razvijana na Nacionalnom institutu za istraživanje informatike i automatizacije u Francuskoj 1994. godine [7]. U okviru ovog projekta pod nazivom "Cristal project" razvija se jezik Objective Caml koji ima veoma visoke performanse i naučnici ga koriste na Linux i MAC OS X platformama. Kao što je već ranije opisano, jezik OCaml i .NET platforma imali su najveći uticaj na razvoj jezika F#. Na slici 1 može se videti deo razvojnog stabla koji ovo prikazuje.



Slika 1: Razvojno stablo

3 Primena i mogućnosti

Snaga F# programskog jezika leži u svedenoj sintaksi koja omogućava laku čitljivost kôda, kao i efikasan razvoj programa koji zahtevaju primenu složenih matematičkih algoritama. Jezik omogućava brzo generisanje prototipova i njihovu brzu transformaciju u produkcioni kôd. Kôd napisan u F#-u lako se može paralelizovati, što je posebno značajno danas kada svi novi računari imaju više jezgara. F# danas ima široku primenu u obradi baza podataka, finansijskog modelovanja, statistici i bioinformatici. Takodje, domen primene svakodnevno raste.

F# je jezik koji ima pretežno funkcionalne karakteristike, ali on je svoju primenu pronašao u još mnogo vrsta programiranja. Neki od njih su: imperativno, objektno-orjentisano, paralelno, distribuirano, asinhrono, meta

programiranje, veb programiranje, skript programiranje, analitičko programiranje, i sl. Danas se on može koristiti na velikom broju sistema, kao što su: Linux, MAC, Windows, Android, iOS, i sl. O ovome će biti više reči u nastavku.

4 Osnovne osobine programskog jezika F#

Neke od osnovnih osobina programskog jezika F# su:

- **Bezbedan**: F# programi se temeljno proveravaju pre izvršenja, što ga čini bezbednim za pokretanje.
- **Funkcionalan**: Funkcije mogu biti ugnježdene, prosleđene kao argumenti drugim funkcijama i vraćene kao povratna vrednost.
- **Strogo tipiziran**: Provera tipova se vrši u toku kompilacije, kako bi se utvrdilo da su dobro definisani.
- **Automatski zaključuje tipove**: Tipovi vrednosti se automatski zaključuju prilikom kompilacije u zavisnosti od konteksta u kome se javljaju, što dovodi do toga da se eksplicitno navođenje retko dešava.
- **Kompatibilan**: F# programi mogu da pozivaju i da budu pozvani iz drugih programa koji su napisani na nekom od jezika koje podržava .NET platforma, kao na primer C#[10].

Još neke specifičnosti jezika:

- Za razliku od drugih programskih jezika struktura **if/elif/else** u svakoj grani ima povratnu vrednost.
- Povratna vrednost ne mora uvek biti vraćena ključnom reči return. Ukoliko je **return** izostavljen, kao povratna vrednost se uzima poslednja izvršena naredba.
- Korišćenjem ključne reči **let** definišemo promenljive koje se ne mogu menjati što kod čini sigurnijim od slučajnih promena. Ukoliko smo sigurni da će se vrednost promenljive menjati, uz let se dodaje ključna reč **mutable**.
- Struktura **switch/case** je u F# zamenjena pattern matching-om[4].
- Pored uobičajenih tipova podataka podržanih u većini programskih jezika, F# podržava novi tip **option type**, koji označava vrednost promenljive koja može da postoji a i ne mora. Dve moguće vrednosti ovog tipa su: *Some(x)* i *None*.

5 Funkcionalno programiranje

Kao glavna, funkcionalna paradigma programskog jezika F# podržava osnovne koncepte koji su podržani i od strane nekih drugih programskih jezika. U ovom odeljku nećemo davati prevelik značaj osnovama funkcionalne paradigme, već ćemo akcenat staviti na *pattern matching*[4].

Jedan od važnih mehanizama u F# programiranju jeste pattern matching. Predstavlja se kao *match ... with ...* konstrukcija koja kombinuje mehanizme dekompozicije i kontrole toka za poklapanje obrazaca. Kao takav, možemo ga koristiti sa jednostavnim kolekcijama (na primer: *list,tuple*) i *option* vrednostima. Naredni listing 1 pokazuje kako se definiše pattern matching.

```
let urlFilter url agent =
 match (url,agent) with
 | "http://www.control.org", 99 -> true
 | "http://www.kaos.org" , _ -> false
 | _, 86 -> true
 | _ -> false
```

Listing 1: Primer pattern matching-a[4]

Izvedeni tip funkcije je:

```
val urlFilter: string -> int -> bool
```

Izraz nakon ključne reči *match* je tupple tipa *(string * int)*. Svako pravilo obrasca se uvodi sa simbolom | iza koga sledi pravilo. Nakon navodjenja pravila sledi simbol -> nakon čega sledi rezultat. Kada se izvrše, pravila obrasca se koriste jedan po jedan i prvi obrazac koji je poklopljen određuje rezultat. Prvi obrazac se poklapa ukoliko su navedene identične vrednosti, dok poslednja tri, na mestima gde se javlja simbol _ za ulazne parametre, mogu da imaju bilo koju vrednost. U tabeli 1 ćemo prikazati koje vrste obrazaca postoje u F# i na koji način ih možemo formirati.

Tabela 1: Načini formiranja obrazaca u F#

Opšti oblik	**Vrsta**	**Primer**
(pat, ... ,pat)	Tuple pattern	(1,2,("3",x))
[pat, ... ,pat]	List pattern	[x;y;z]
[|pat, ... ,pat|]	Array pattern	['a';'b';'c']
null	Null test pattern	null

6 Asinhrono i paralelno programiranje

Razlog zbog kojeg izučavamo asinhrono i paralelno programiranje se ogleda u maksimalnoj iskorišćenosti hardvera. Ranije se ovaj način programiranja izbegavao zbog dodatne složenosti i rizika od bagova. Sa pojavom velikog broja biblioteka u kombinaciji sa funkcionalnim programiranjem i izbegavanjem bočnih efekata ovaj problem je smanjen. Fokus ćemo staviti na to kako ubrzati računanje u F# koristeći asinhrono i paralelno programiranje. Izvršavanje koda se odvija pomoću niti i F# biblioteka. Mogu se potpuno iskoristiti prednosti više procesora i jezgara tako što se podeli jedan posao na više manjih. Najvažniji razlog uvođenja ova dva načina programiranja je ubrzanje izvršavanja programa.

6.1 Asinhrono programiranje

Opisuje programe i operacije koje se izvršavaju u pozadini i završavaju nakon nekog vremena (na primer: preuzimanje novog email-a bez čestog proveravanja). Asinhroni programi na .NET platformi su pisani korišćenjem modela asinhronog programiranja (APM)[9]. APM je obrazac koji deli asinhrone operacije na dve metode: *BeginOperation* i *EndOperation*. Kada se pozove *BeginOperation* operacije počinju sa asinhronim izvršavanjem

koje po završetku pozivaju *EndOperation* koji dohvata rezultat asinhronog izračunavanja. Problemi koji nastaju korišćenjem ovog obrasca su:

- Izostavljanje poziva *EndOperation* dolazi do neželjenih efekata i curenja memorije.

- Ukoliko se nekoliko asinhronih izračunavanja vrati nazad teško je pratiti tok programa.

- Posledica prethodnog problema uzrokuje pojavu špageti programiranja.

Umesto APM-a sve asinhrone operacije mogu se izvršavati pomoću F# biblioteke asinhronih radnih tokova[3]. Ključna reč *async* se koristi za kreiranje bloka u koji se smešta kod za izvršavanje, korišćenjem *let!* i *do!* operatora. Rezultat ovog izvršavanja će biti tipa *Async<T>*[3].

6.2 Paralelno programiranje

Paralelno programiranje se koristi za podelu poslova na n delova kako bi se dobilo n puta brže izvršavanje što je retko u praksi. Najjednostavniji način koriščćenja paralelnog programiranja na .NET 4.0 platformi se zasniva na upotrebi *Parallel Extesions* biblioteke (PFX)[3]. Jedna od pogodnosti PFX biblioteke je ta što programer ne mora da misli o kontroli rada niti. Za razliku od paralelnog, asinhroni tok izvršavanja je dostupan i na prethodnim verzijama .NET platforme. Task objekat predstavlja osnovnu strukturu PFX paralelizma, koji ima zadatak da izvrši posao nakon nekog vremena. Kao takav, nije dovoljan za paralelne aplikacije. Zbog problema koji se javlja pri radu Task objekta sa deljivim podacima uvodi se zaključavanje podatka (*exclusive lock*). PFX biblioteka uvodi nove tipove kolekcija *System.Collections.Concurrent* [13] kao rešenje prethodnog problema.

7 Radni okviri

Najpoznatiji i najznačajniji radni okvir (eng. framework) za ovaj programski jezik je .NET koji je ranije već pomeut. .NET je pojednostavio razvoj RP(Reactive Programming)[18] aplikacija. Ovaj radni okvir je potpuno besplatan i na njemu je moguće kreirati veliki broj različitih aplikacija. On podržava veliki broj programskih jezika iz različitih paradigmi, editora i biblioteka za igradnju mobilnih i veb aplikacija. Razvijene su dve glavne komponente za korišćenje rekurzivnog paralelnog izračunavanja na lokalnoj mreži i obe su dostupne programima na .NET platformi.[20].

Temelj .NET platforme je zajednička jezička infrastruktura CLI (*Common Language Infrastructure*)[11]. Pokretanje F# kôda na ovoj platformi se vrši tako što na samom početku kompajler prevodi kôd u binarni fajl koji je preveden na asemblerski jezik višeg nivoa koji se zove MSIL (*Microsoft Intermediate Language*)[8]. Implementacija na CLI kompajleru ovog asemblerskog kôda u toku izvršavanja je mnogo brža nego da je kôd samo interpretiran i ova kompilacija se izvršava u trenutku[3]. Kôd koji je preveden u MSIL i izvršen u trenutku, predstavlja kôd za upravljanje za razliku od kodova pisanih na progamskim jezicima kao što su C ili C++ koji su sirovi kodovi.

Postoje neke prednosti za prolazak kroz CLI ili upravljani kôd a ne direktnog kompiliranja na mašinski nivo:

- kompatibilnost medju jezicima

- mogućnost rada na više platformi

- mašinska nezavisnost

.NET platforma ima još jednu prednost a to je prikupljanje smeća sto omogućava programeru da ne razmišlja mnogo o alociranju memorije i oslobađanju, već da se samo skoncentriše na pisanje kôda. Iako ne mora da obraća pažnju na smeće i rad sa memorijom poželjno je da programer zna na koji način taj sakupljač smeća radi, kako on oslobađa memoriju i rešava taj problem.

Naravno da ova platforma nije jedina koja se koristi za ovaj progamski jezik, pored nje postoje: web radni okviri (na primer: *Suave, Fable, ASP.NET Core, Giraffe, WebSharper, Freya, NancyFx, Serving Requests with IHttpHandler, Serving Requests with Azure, Functions, Pure F# Web API 2.0, SignalR, ServiceStack, ASP.NET Blazor*) i radni okviri za testiranje web-a (na primer: *Web Testing, Frameworks, Canopy for Client-side Testing, Unit Testing Libraries*)[5].

8 Uputstvo za instalaciju

U ovom poglavlju biće opisano uputstvo za instalaciju programskog jezika F# na Windows i Linux operativnim sistemima, kao i uputstvo za pokretanje F# programa.

8.1 Uputstvo za Windows

Na ovom operativnom sistemu postoje tri načina za instalaciju F#-a. To su: *Visual studio*[14], *Visual Studio Code* [15] i *JetBrains Rider* [2].

Pokretanje F# koda na Windows-u se vrši tako što prvo označite kod ili deo koda koji želite da se izvrši zatim desnim klikom na miš izaberite opciju pošalji interaktivno ("send to interactive") ukoliko je interaktivni prozor tj prozor na kom se ispisuje rezultat otvoren ili iskoristite prečicu Alt+Enter.

8.1.1 Visual studio

Na Windows operativnim sistemima se uglavnom koristi *Visual Studio* alat. *Visual studio 2017* je alat koji dolazi sa podrškom za F# u svim verzijama: *Professional, Enterprise* i *Community* (verzija je potpuno besplatna). Međutim, ukoliko imate već instaliran *Visual Studio 2012/13/15 Professional* ili *Above* možete ga takođe koristiti zato što i on podržava alate za F# iako nije toliko napredan kao *Visual Studio 2017*.

8.1.2 Visual Studio Code

Visual Studio Code je besplatna platforma koja podržava mnogo jezika,a među njima i F# . On je podržan od strane *Ionide*[1].

1.korak: Instalirati *Visual Studio Code* za Windows.

2.korak: Pritisnuti **Ctrl+Shift+p** i ukucati sledeću naredbu za instalaciju *Ionide* paketa za *Visual Studio Code*:

```
1000 ext install Ionide-fsharp
```

8.1.3 JetBrains Rider

JetBrains Rider je platforma .NET IDE izgrađena je pomoću *InteliJ* i *ReSharper* tehnologije. *JetBrains* nudi podršku .NET i .NET Core aplikacijama na svim platformama.

1.korak: Instalirati *JetBrains* za Windows.

2.korak(opciono): Instalirati poslednju .NED Core SDK.

Takođe vam je potrebno da instalirate ceo *Visual Studio* ili F# kompajler.

8.2 Uputstvo za Linux

Kada koristite jezik koji podržava .NET platforma kao što je F# onda on zahteva Mono[17] softversku platformu. Većina linux distributera sadrži neku verziju te platforme koja omogućava programerima lakše kreiranje aplikacija.

Instalacija F# na Linuxu se izvršava na potpuno identičan način za *Ubuntu, Mint* i *Debian* verziju.

Za pokretanje F# programa na Linuxu preko terminala koristi se sledeća naredba ukoliko se nalazite u direktorijumu u kom se nalazi primer koji pokrećete.

```
1000 fsharpc primer.fs
```

8.2.1 Ubuntu/Mint/Debian

1.korak: Dodati *Mono*[17] repozitorijum vašem menadžeru paketa.

2.korak: Instalirati F# koji će istovremeno povući novu verziju *Mono*-a ukoliko je to potrebno.

```
1000 sudo apt-get update
     sudo apt-get install fsharp
```

9 Upoznavanje sa F#

F# je zamišljen kao jezik sa razumljivom i čitljivom sintaksom i kao takav lakim za održavanje. Programeri "Microsoft"kompanije su se potrudili da ovaj jezik bude intuitivan i prijemčljiv svim programerima. Ono što na prvi pogled vidimo jeste da je sintaksa čista. Naime F# ne koristi tačku-zarez za kraj izraza niti vitičaste zagrade za označavanje bloka.

```
1000 module Ispis =
         printfn "Blok pravimo uvlacenjem izraza"
1002     printfn "Ne koristimo ;"
```

Navedene osobine F# čine kôd kompaktnijim i konciznijim. Pa tako možemo u 3 reda da izdvojimo, kvadriramo i saberemo sve parne brojeve.

```
1000 module SumaKvadriranihParnihBrojeva =
     let kvadrat x = x * x
1002 let paran x = x % 2 = 0
     [1..10] |> List.filter paran |> List.map kvadrat |> List.sum
      |> printfn "sum=%d"
```

Ono što u oba primera vidimo jeste ključna reč **module**. Ona grupiše vrednosti, funkcije, tipove u jednu grupu čime dobijamo logičku celinu i smanjujemo mogućnost da dođe do konflikta imena. Ukoliko ne navedemo modul, F# implicitno smešta u podrazumevani modul koji se zove kao i ime datoteke sa velikim početnim slovom. Takođe ako želimo da pristupimo drugom modulu koristimo tačku pa tako ako želimo da pristupimo funkciji iz gore navedenog modula koristimo komandu: *SumaKvadriranihParnihBrojeva.kvadrat*. Možemo da uočimo čudan operator |> koji se zove pajp (*pipe*). On sprovodi rezultat prvog izraza u argument drugog izraza.

9.1 Sve je izraz

F# je jezik zasnovan na izrazima (*expression-based* [?]). To znači da svaki delić F# programa može biti evaluiran. Gore govorivši o pajpu namerno sam spomenuo izraze a ne funkcije jer je ovde svaka funkcija izraz što znači da svaka funkcija vraća vrednost. Dodatno i sam konstrukt **if...else** je izraz. To možemo da vidimo u sledećoj modifikaciji gore navedenog kôda.

```
1000 module SumaKvadriranihParnihBrojeva =
     let kvadrat x = x * x
1002 let paran x =  if x % 2 = 0 then  true else false
     [1..10] |> List.filter paran |> List.map kvadrat |> List.sum
      |> printfn "sum=%d"
```

Naredbom **let** vezujemo promenljivu za vrednost. Na prvu ruku može pogrešno da se zaključi da je reč o običnoj dodeli. Za razliku od nekih drugih programskih jezika u F# je podrazumevano povezivanje imutabilno tj. nije je moguće promeniti tokom izvršavanja programa. To za posledicu ima odsustvo sporednog efekta (*side-effect*). Ukoliko želimo mutabilno povezivanje, tj. ukoliko želimo da tokom izvršavanje menjamo vrednost, dodajemo ključnu reč **mutable** pa je sintaksa **let mutable**. U našem primeru koristimo istu ključnu reč za definisanje funkcije što može da bude zbunjujuće. Naime jedina razlika između obične vrednosti i funkcije je u tomše što funkcija mora da se primeni na argument kako bi bila evaluirana. Dodatno, funkcija iako nije primenjena na argument, može da bude argument nekih drugih funkcija. Takve funkcije su poznate kao funkcije višeg reda. Tako u potipsu funkcije *List.filter* koju koristimo u našem primeru stoji *List.filter : ('T -> bool) -> 'T list -> 'T list* gde možemo videti da je prvi argument *('T -> bool)* ustvari funkcija koja vraća *bool* vrednost. Ovaj koncept je u biti funkcionalnog razmišljanja.

9.2 Briga o tipovima

Oni koji imaju iskustva sa drugim programskim jezicima mogu da zapaze da se nigde u kôdu ne navode tipovi. Iako tip može da se eksplicitno

navede sam programski jezik poseduje mehanizam za dedukciju tipova. Preciznije, kompajler može na osnovu upotrebe i pojavljivanja vrednosti da zaključi koji je tip u pitanju. F# je statički tipiziran jezik pa se dedukcija tipova odvija u fazi kompilacije. U našem primeru kompajler zaključuje na osnovu date liste *[1..10]* da se radi o celim brojevima pa donosi zaključak da se i množenje odvija nad celim brojevima i da se kao rezultat dobijaju celi brojevi. Ukoliko uzmemo listu sa razlomljenim brojevima kompajler prijavljuje grešku.

```
module SumaKvadriranihParnihBrojeva =
    let kvadrat x = x * x
    let paran x =  if x % 2 = 0 then  true else false
    [1.2; 4.1; 6.3; 7.4] |> List.filter paran  |> List.map kvadrat
      |> List.sum |> printfn "sum=%f"

    // Type mismatch. Expecting a 'float -> bool' but given a 'int
      -> bool'
```

Na osnovu liste sa razlomljenim brojevima kompajler zaključuje da funkcija koja se prosleđuje funkciji *List.filter* treba da operiše sa float vrednostima. Kako je operator za ostatak definisan nad celim brojevima potpis funkcije *paran* je *int -> bool* što dovodi do konflikta. Ovde smo mogli da vidimo da F# nema implicitnu konverziju i da svako nepodudaranje tipova prijavljuje kompajlersku grešku.

9.3 Korisnički tipovi

Videli smo da se mehanizam za dedukciju tipova pokazuje i kao sigurnosni mehanizam koji sprečava i obaveštava programera o problemima sa tipovima (u fazi kompilacije) čime sprečava greške u fazi izvršavanja. Ova osobina nas štiti od potencijalnih problema prilikom definisanja korisničkih tipova.

10 Zaključak

Sta je zakljucak celog rada.

Literatura

[1] Ionide. on-line at: http://ionide.io/.

[2] JetBrains s.r.o. Developed with drive and IntelliJ IDEA. Jet Brains rider, 2000-2019. on-line at: https://www.jetbrains.com/rider/.

[3] Smith Chris. *Programing F#*. O'Reilly Media, 1005 Gravenstein Highway North, 2009.

[4] Antonio Cisternino Don Syme, Adam Granicz. *Expert F#*. Kinetic Publishing Services, LLC, New York, NY 10013., 2007.

[5] F# Software Foundation and individual contributors. List of frameworks, 2012-2018. on-line at: https://fsharp.org/guides/web/.

[6] F# Software Foundation and individual contributors. The F# Language specification, 2012-2018. on-line at: https://fsharp.org/specs/language-spec/.

[7] Jon Harrop. *F# for Scientists*. Wiley-Interscience, New York, NY, USA, 2008.

[8] Microsoft. Assemblies in the Common Language Runtime (MSIL). on-line at: https://docs.microsoft.com/en-us/dotnet/framework/app-domains/assemblies-in-the-common-language-runtime.

[9] Microsoft. Asynchronous Programming Model (APM). on-line at: https://docs.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/asynchronous-programming-model-apm.

[10] Microsoft. C# Guide. on-line at: https://docs.microsoft.com/en-us/dotnet/csharp/index.

[11] Microsoft. Get started with F# with the .NET Core CLI. on-line at: https://docs.microsoft.com/en-us/dotnet/fsharp/get-started/get-started-command-line.

[12] Microsoft. .NET framework. on-line at: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet.

[13] Microsoft. System.Collections.Concurrent, 2019. on-line at: https://docs.microsoft.com/en-us/dotnet/api/system.collections?view=netframework-4.7.2.

[14] Microsoft. Visual Studio, 2019. on-line at: https://visualstudio.microsoft.com/.

[15] Microsoft. Visual Studio Code, 2019. on-line at: https://code.visualstudio.com/.

[16] Robin Milner. Logic for computable functions: Description of a machine implementation. Technical report, Stanford, CA, USA, 1972.

[17] Mono Project. Mono, 2019. on-line at: https://www.mono-project.com/.

[18] ReactiveX. Reactive programing. on-line at: http://reactivex.io/.

[19] Don Syme. The Early History of F#. on-line at: https://fsharp.org/history/hopl-draft-1.pdf.

[20] V. V. Vasilchikov. On the recursive parallel programming for the .net framework. *Automatic Control and Computer Sciences*, 2014.