Optimizacija funkcionalnih kompajlera koristeći osnove teorije kategorija

Stefan Isailović

21.4.2023

Uvod

Napisati program koji će da iz date liste, u kojoj su vrednosti pomnožene sa 2, izvuče element

Uvod

Napisati program koji će da iz date liste, u kojoj su vrednosti pomnožene sa 2, izvuče element

```
(head . double) [1,2,3]
----------------------------
~= head(double([1,2,3]))
     = head([2,4,6])
          = 2
```

Uvod

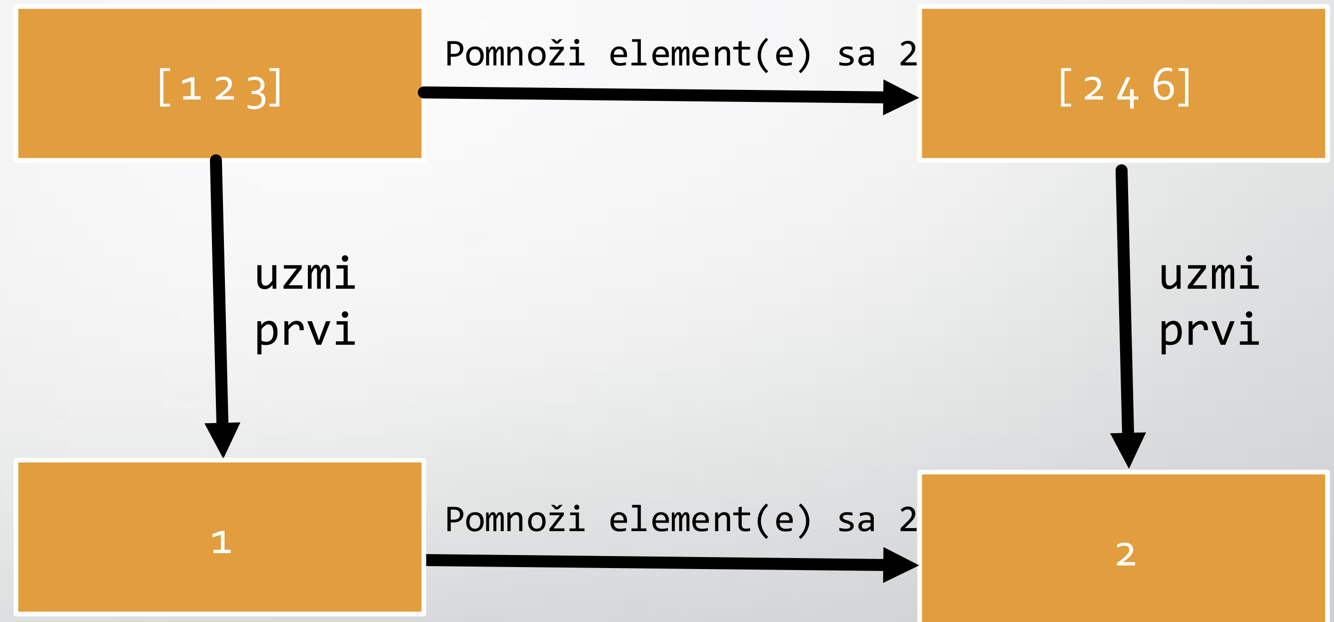
Napisati program koji će da iz date liste, u kojoj su vrednosti pomnožene sa 2, izvuče element

```
(head . double) [1,2,3]
------------------------------
~= head(double([1,2,3]))
      = head([2,4,6])
            = 2
```

```
(double . head) [1,2,3]
------------------------------
~= double(head([1,2,3]))
        = double(1)
            = 2
```

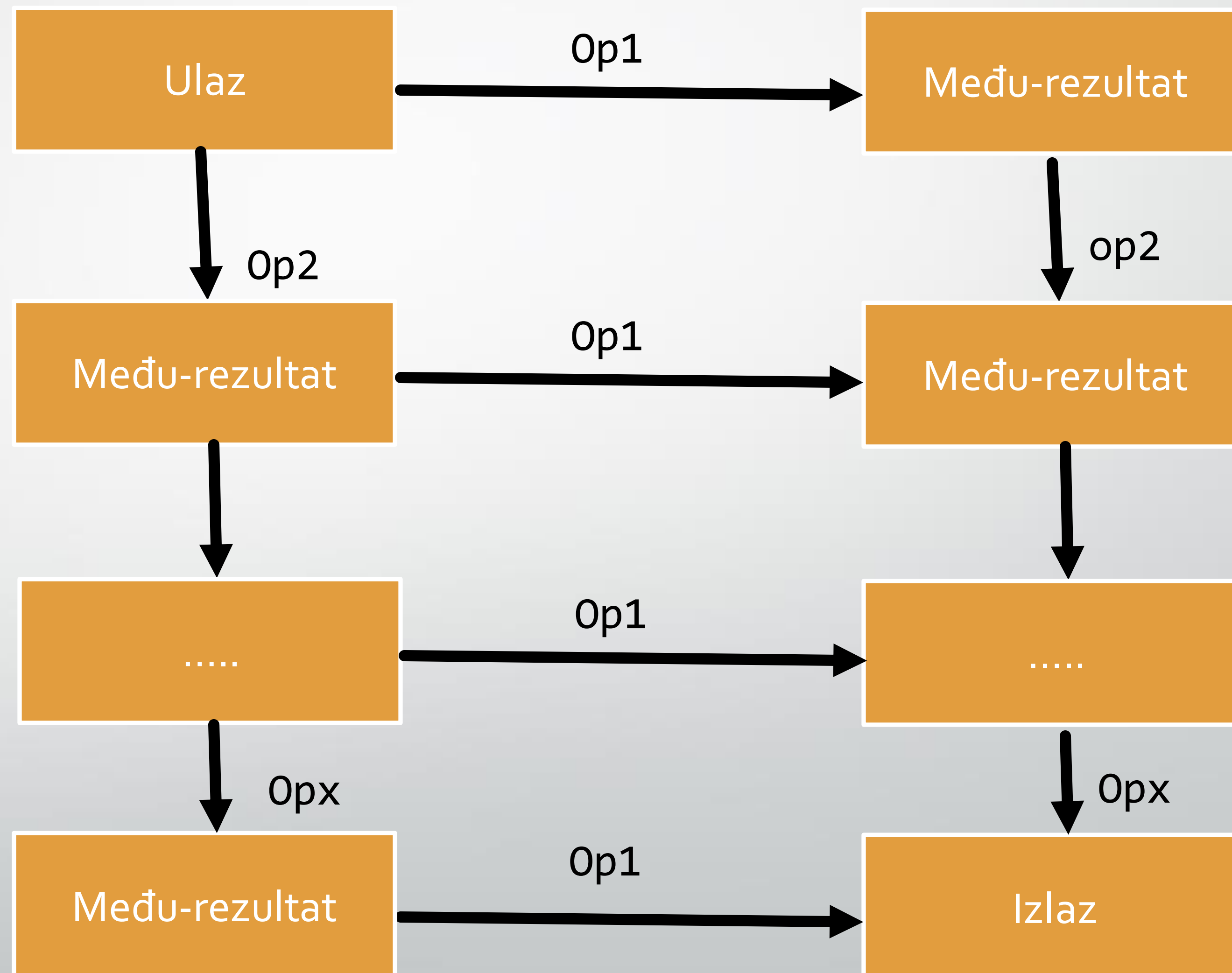
Problem - Dijagram

- Ulaz: [Int]
- Izlaz: Int
  - Prvi element liste pomnožen sa 2



Apstrakcija Problema

- Apstrakcija na n operacija
- Izlaz jedne operacije
  - ulaz sledeće operacije
- **Da li postoji način da kompajler optimizuje izvršavanje operacija?**

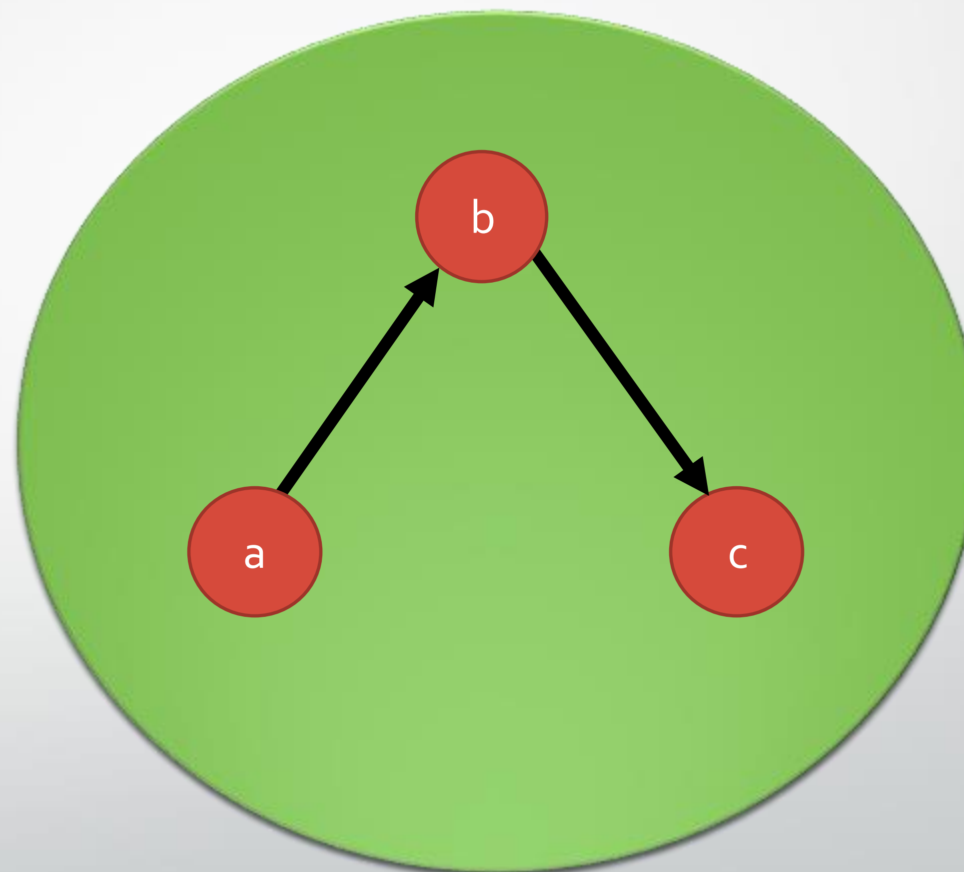


Odgovor je DA!

- Teorija kategorija
- Funktori
- Optimizacija – kada i kako se izvršava

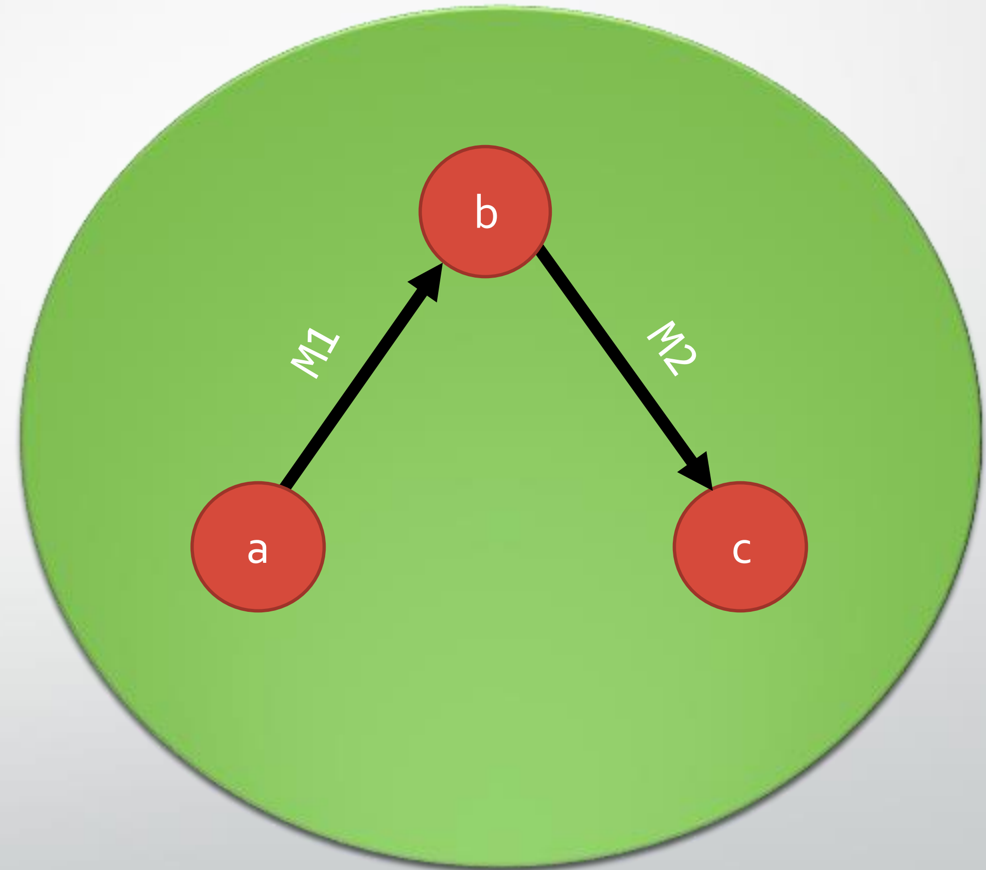
Kategorija

- Šta čini kategoriju?
  - Objekti
    - Skupovi, tipovi, kategorije ...
  - Strelice
    - Funkcije, transformacije



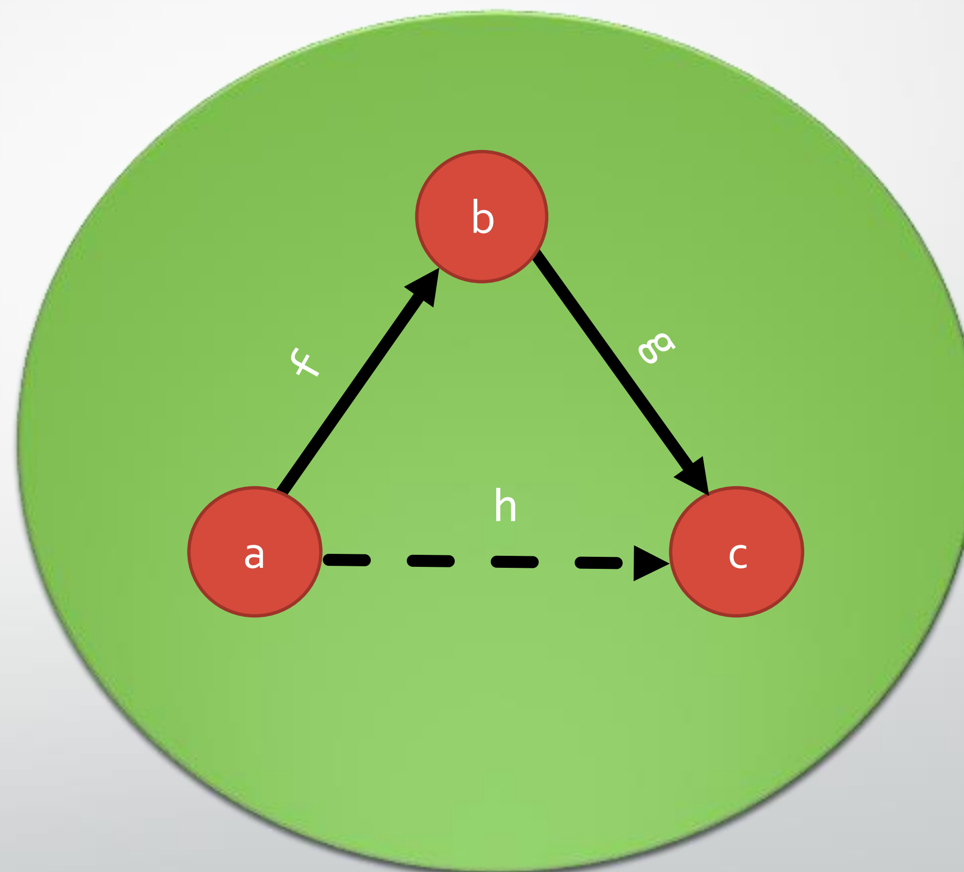
Morfizmi

- Strelice ~ Morfizmi
  - Prelaz iz jednog objekta u drugi
  - Mogu biti funkcije, transformacije
  - Formiraju kompozicije



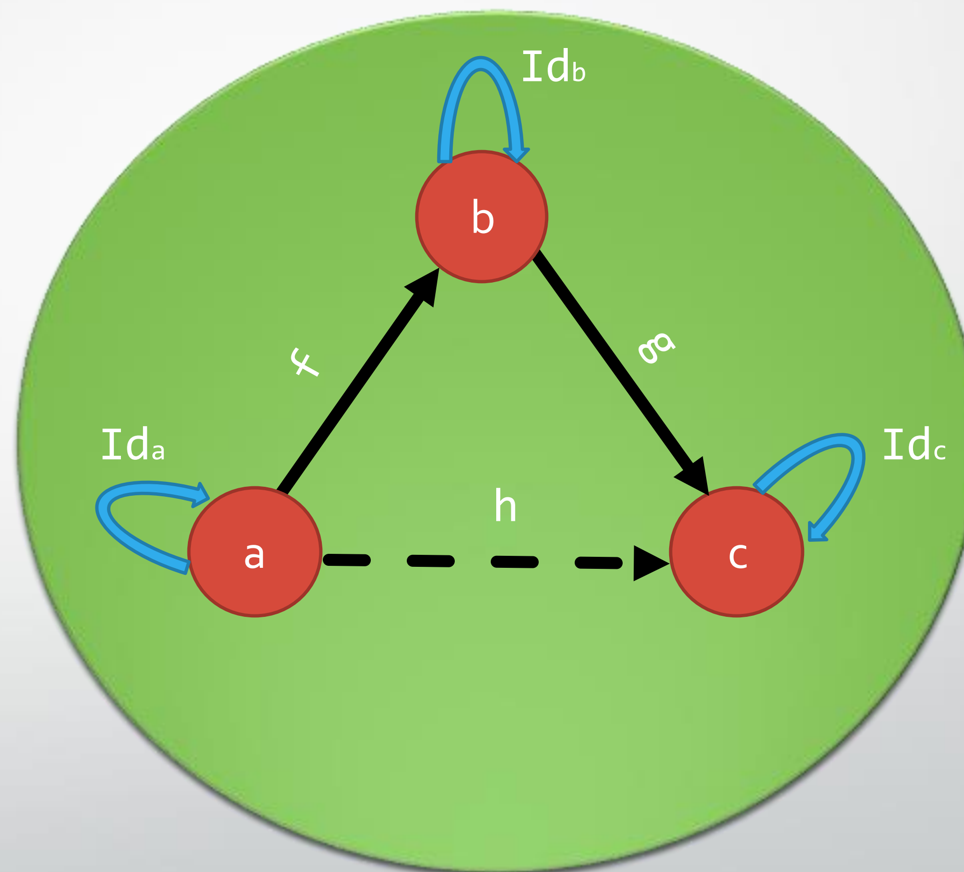
Uslovi Kategorije

- Kompozicija
  - f :: a -> b
  - g :: b -> c
    - tada implicitno postoji morfizam h :: a -> c (kao kompozicija f . g)



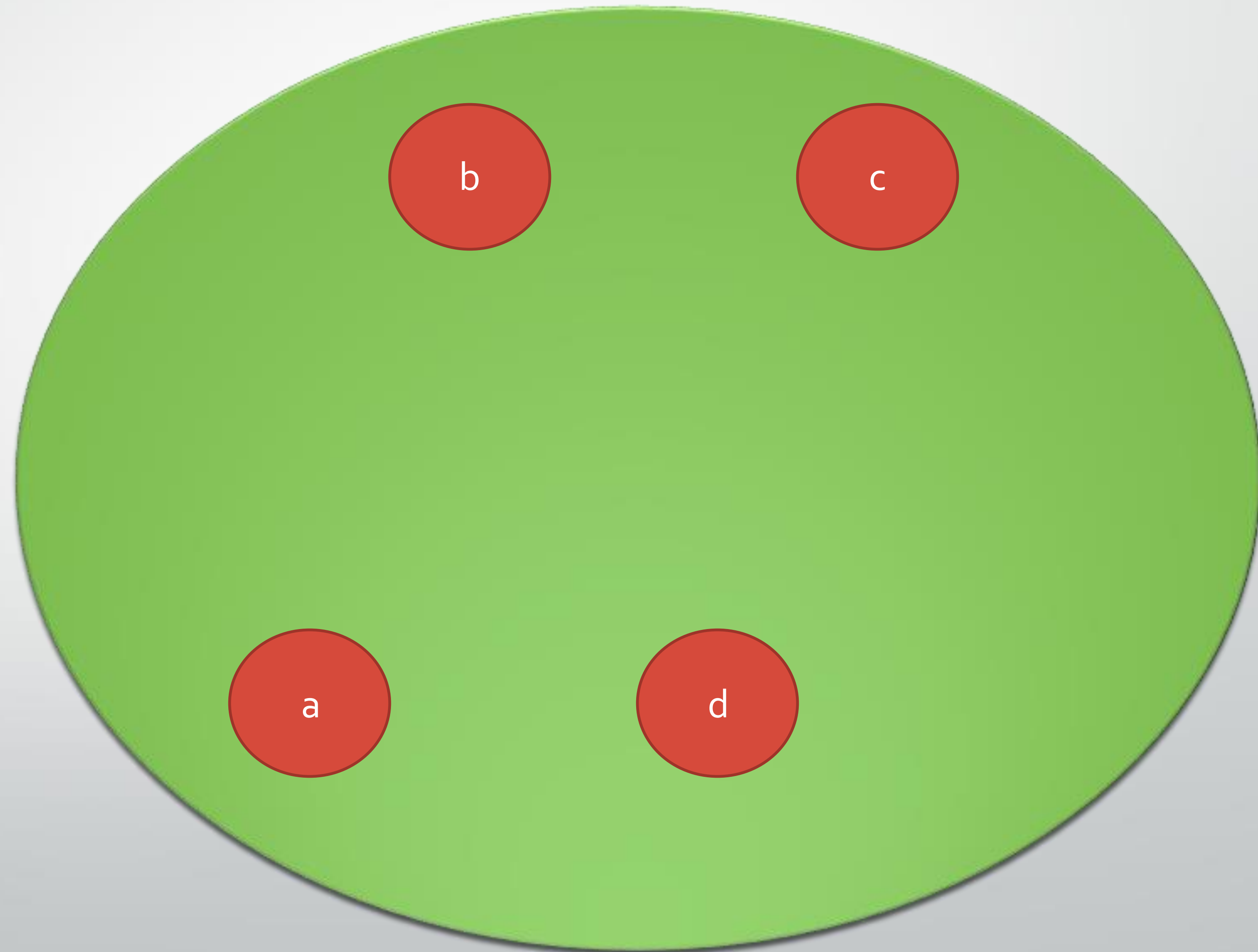
Uslovi Kategorije

- Kompozicija
  - f :: a -> b
  - g :: b -> c
    - tada implicitno postoji morfizam h :: a -> c (kao kompozicija f . g)
- Identitet (Id)
  - Morifizam koji preslikava objekat u samog sebe



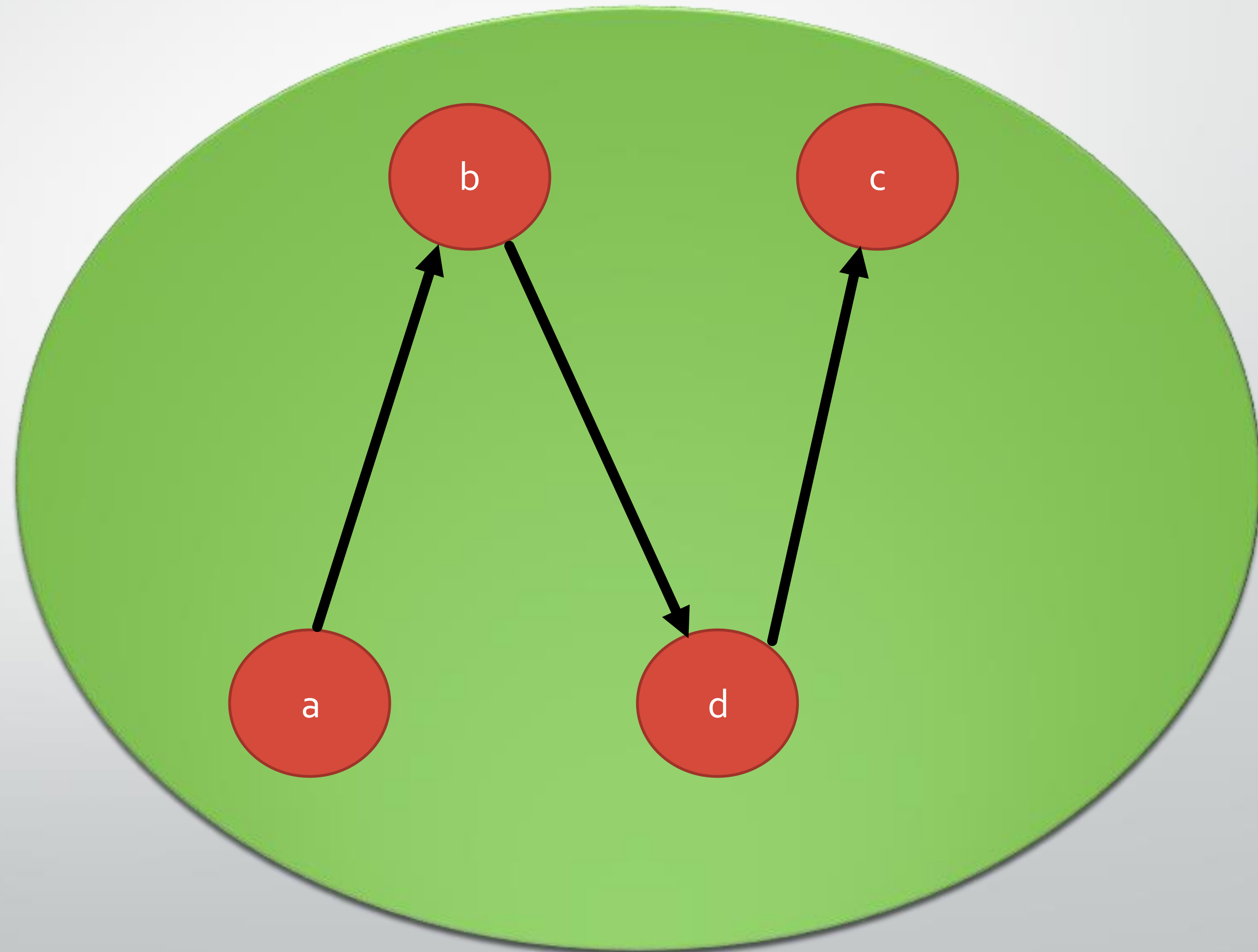
Dijagram

- Dodati objekte na dijagram



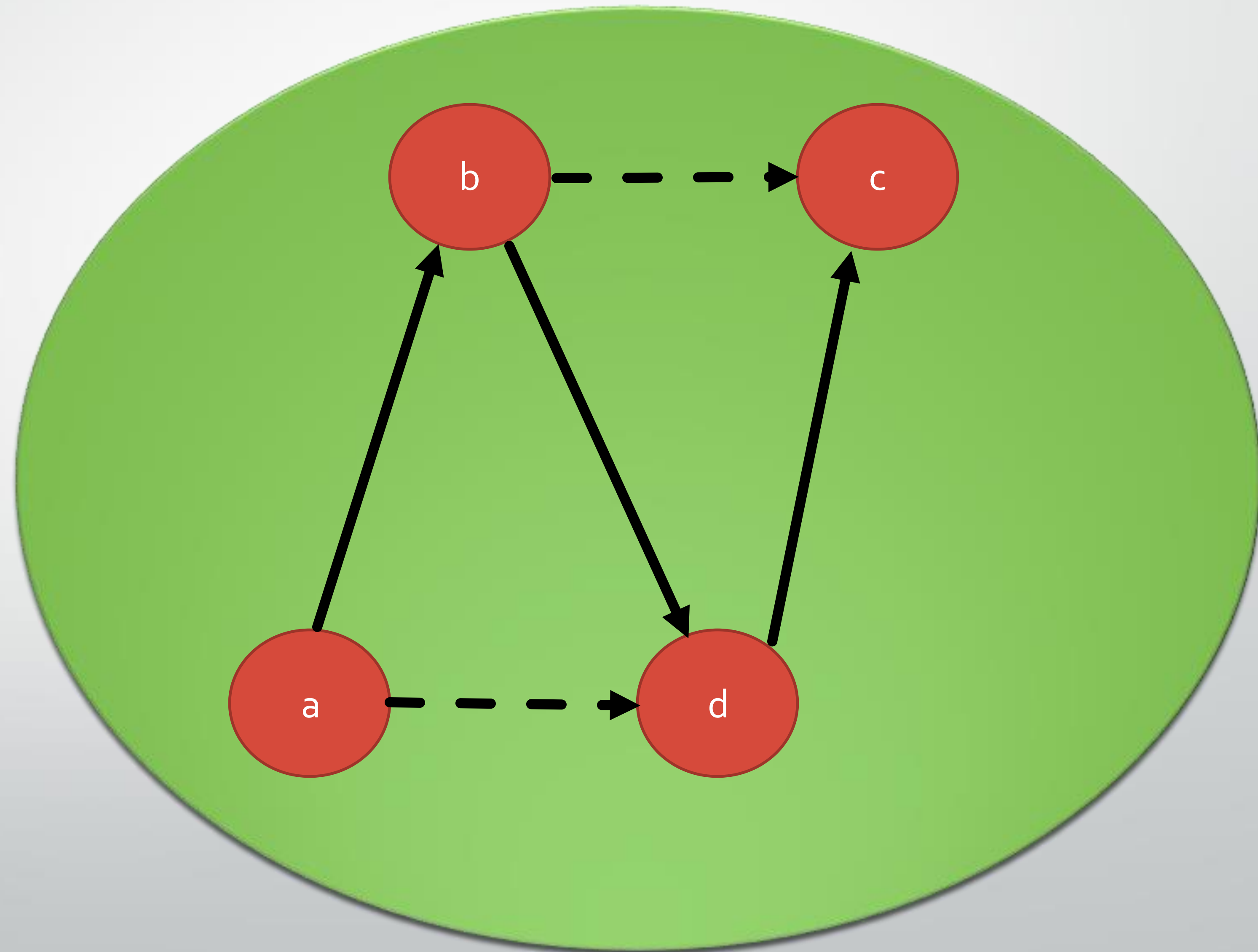
Dijagram

- Dodavanje objekata
- Dodavanje početnih morfizmama



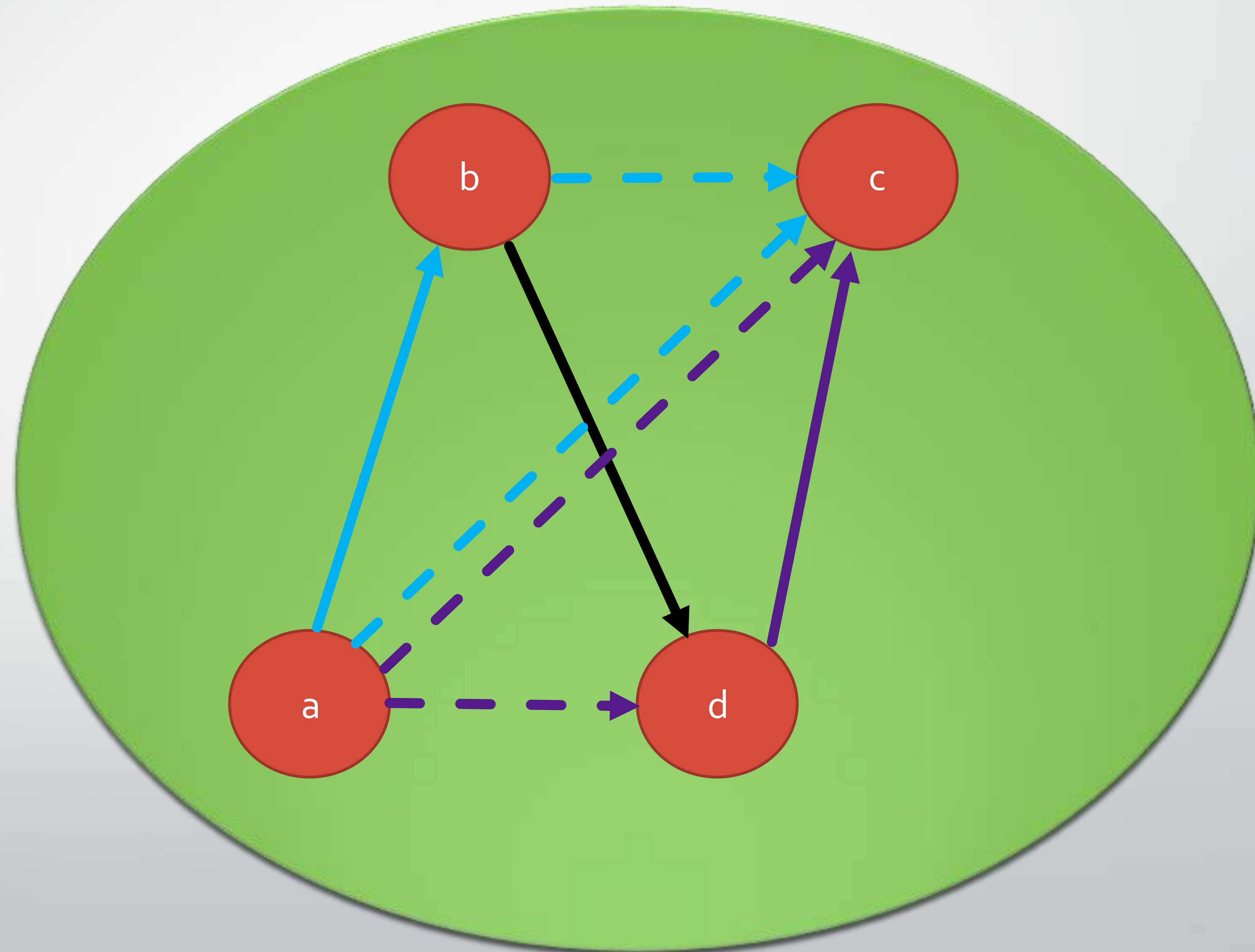
Dijagram

- Dodavanje objekata
- Dodavanje početnih morfizmama
- Izvođenje kompozicija



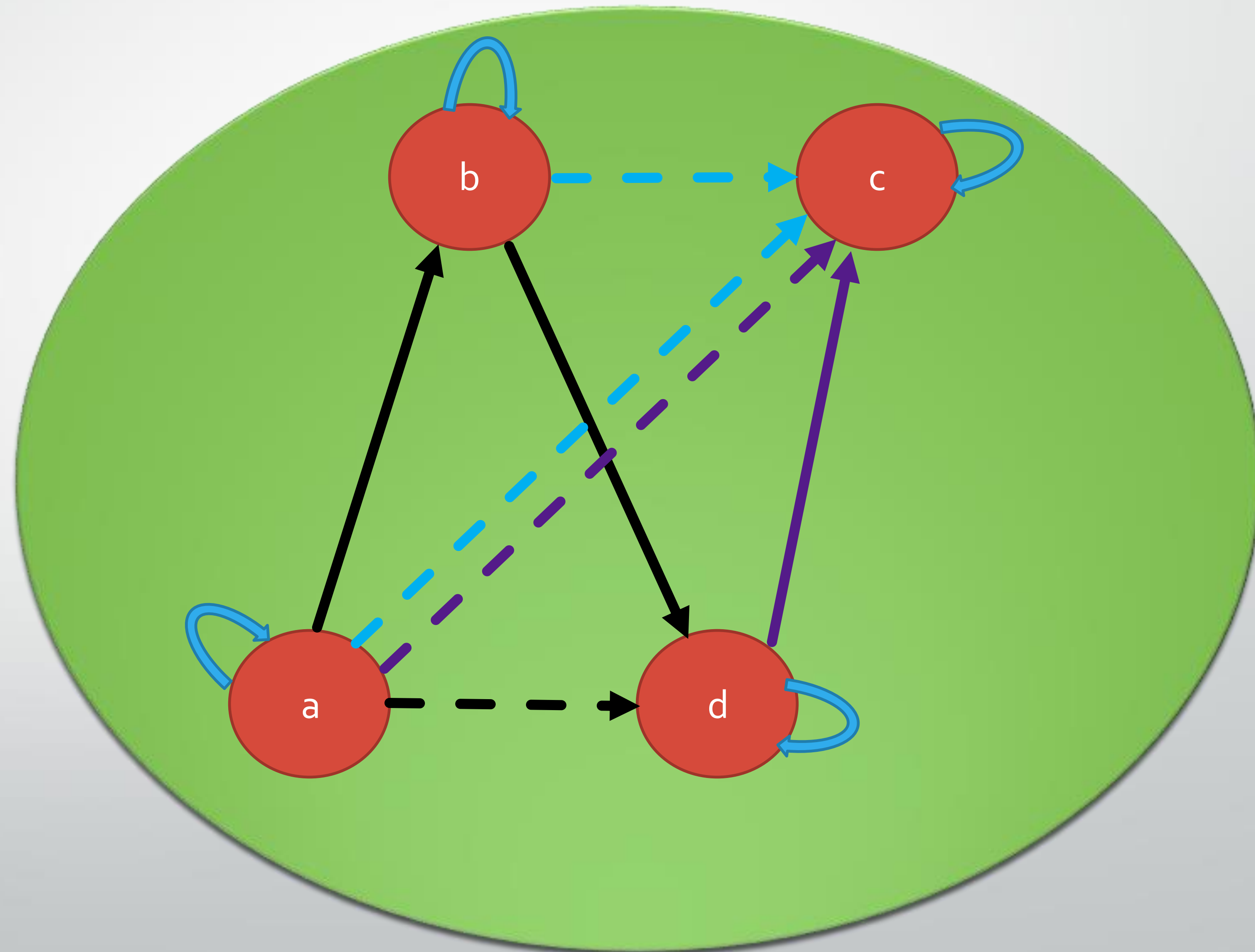
Dijagram

- Dodavanje objekata
- Dodavanje početnih m orfizmama
- Izvođenje kompozicija
- **Komutirajući dijagram**
  - **a-> b** i **b -> c** => **a -> c**
  - **a -> d** i **d -> c** => **a -> c**

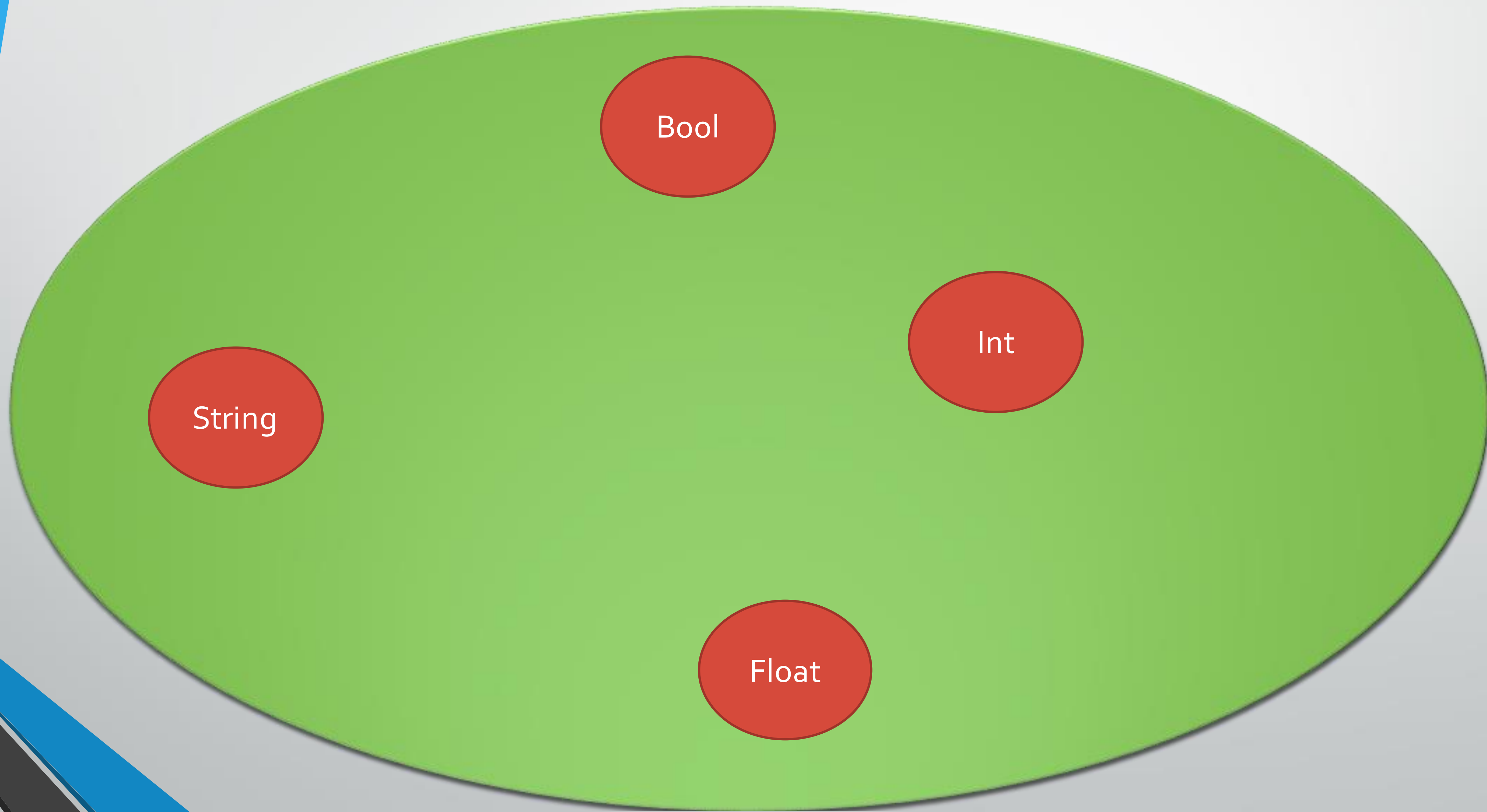


Dijagram

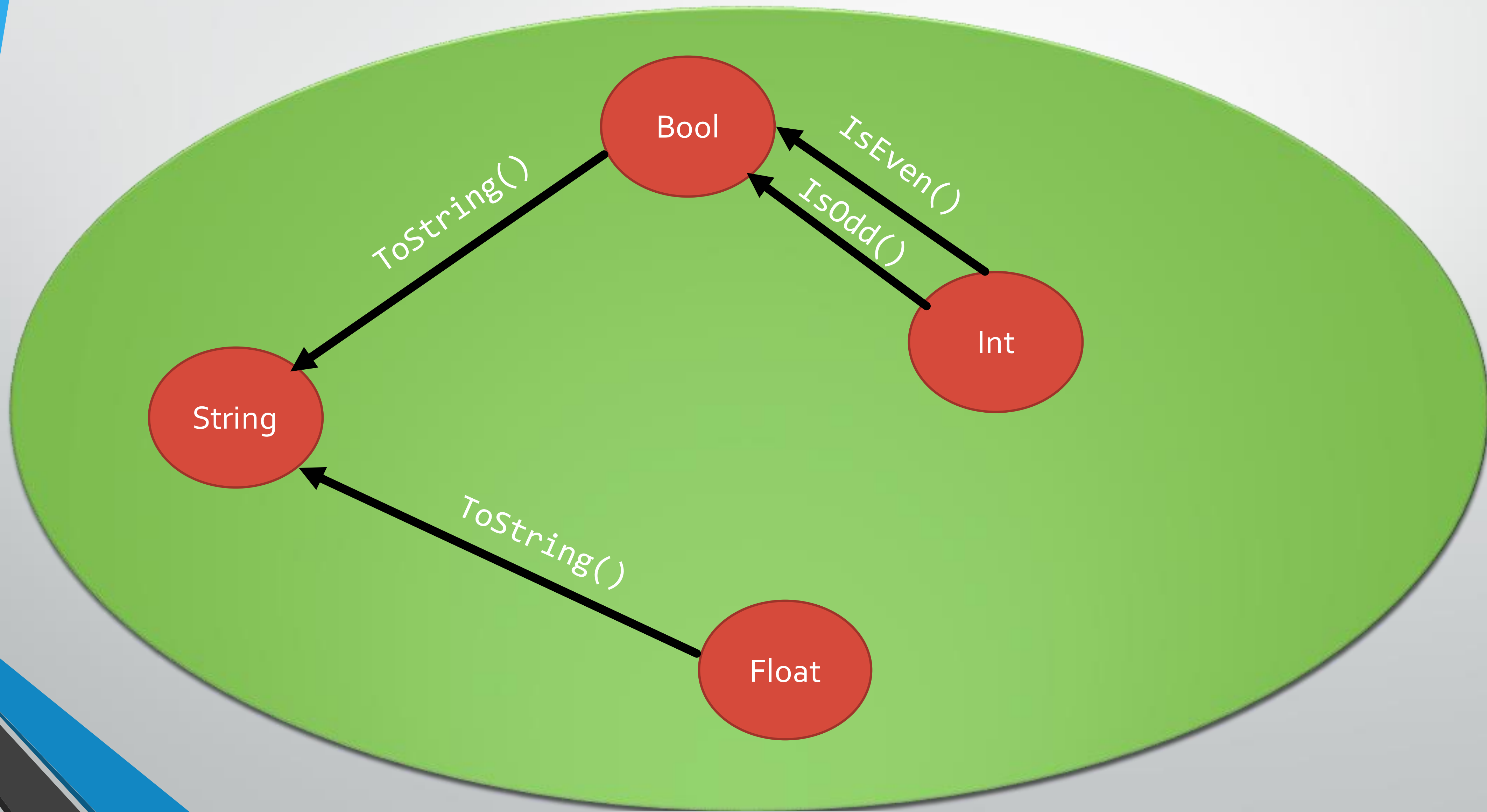
- Dodavanje objekata
- Dodavanje početnih morfizmama
- Izvođenje kompozicija
- Dodavanje Id morfizama



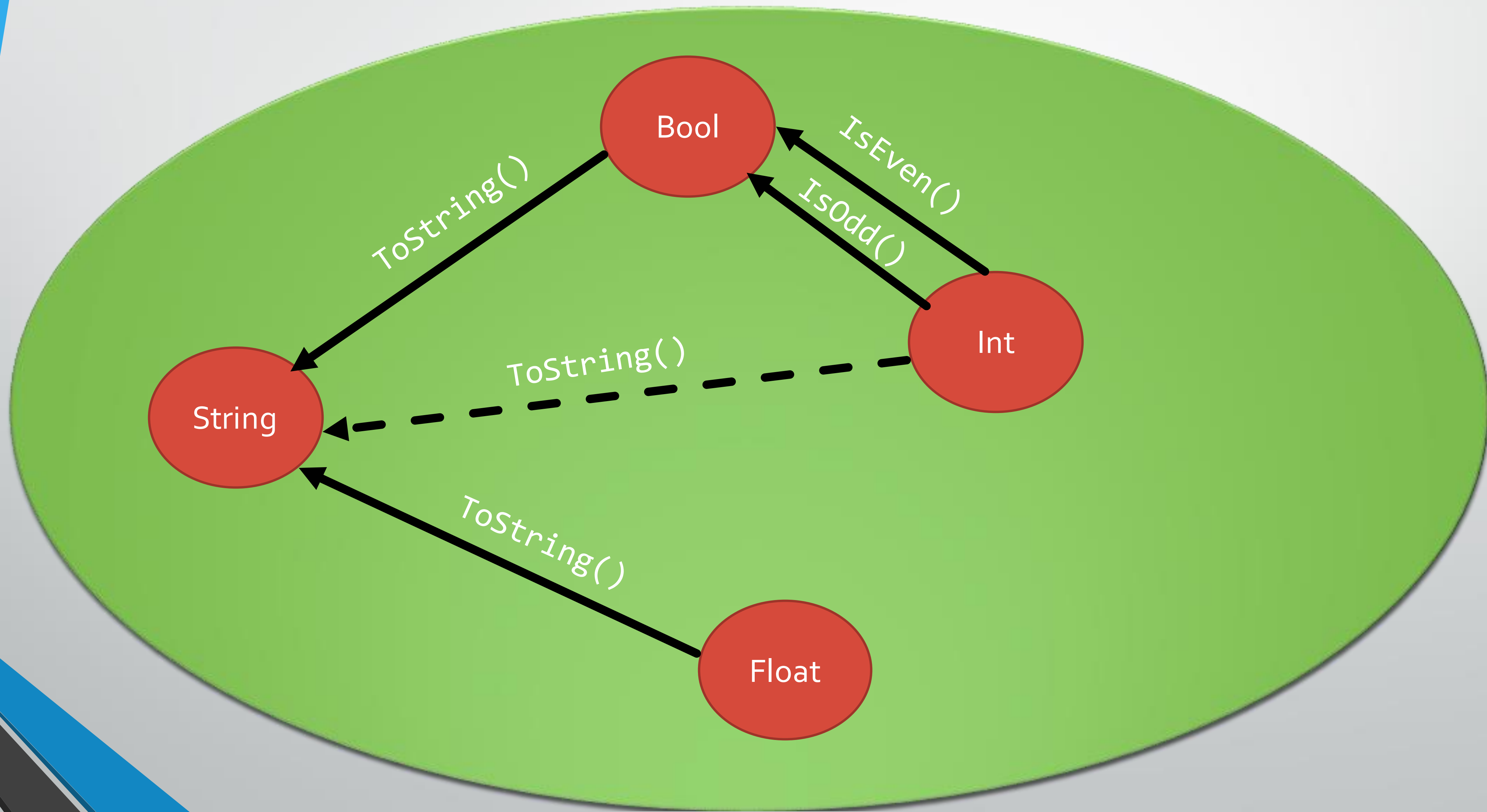
Tipovi kao objekti



Tipovi kao objekti

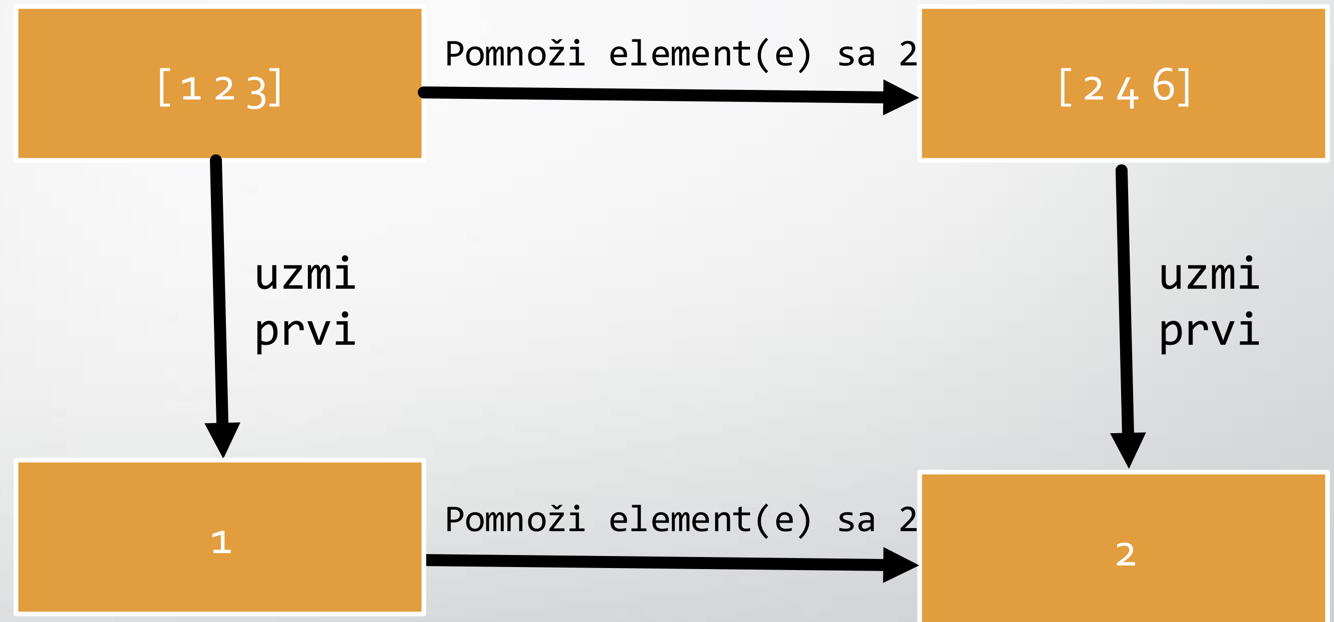


Tipovi kao objekti



Problem - Dijagram

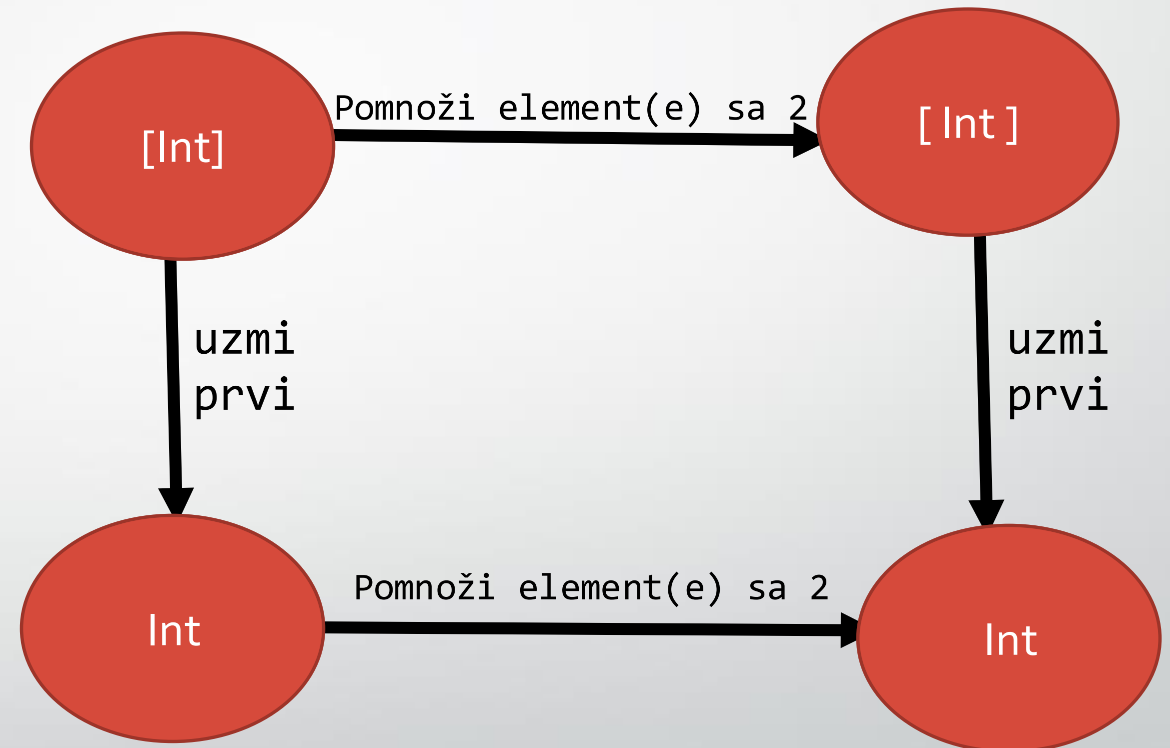
- Ulaz: [Int]
- Izlaz: Int
  - Prvi element liste pomnožen sa 2



Problem - Progres

- Ulaz: [Int]
- Izlaz: Int
  - Prvi element liste pomnožen sa 2

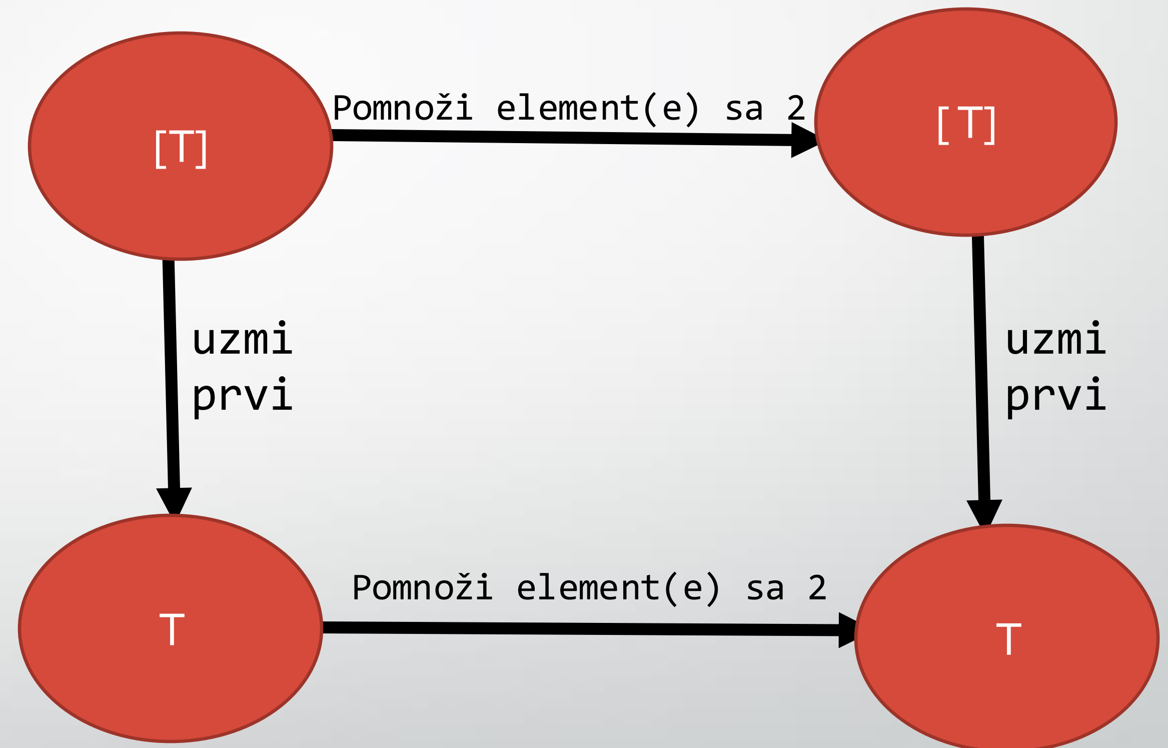
- Da li možemo da generalizujemo problem nezavisno od tipa?



Problem - Progres

- Ulaz: [Int]
- Izlaz: Int
  - Prvi element liste pomnožen sa 2

- Ne moramo da se ograničavamo na tipove

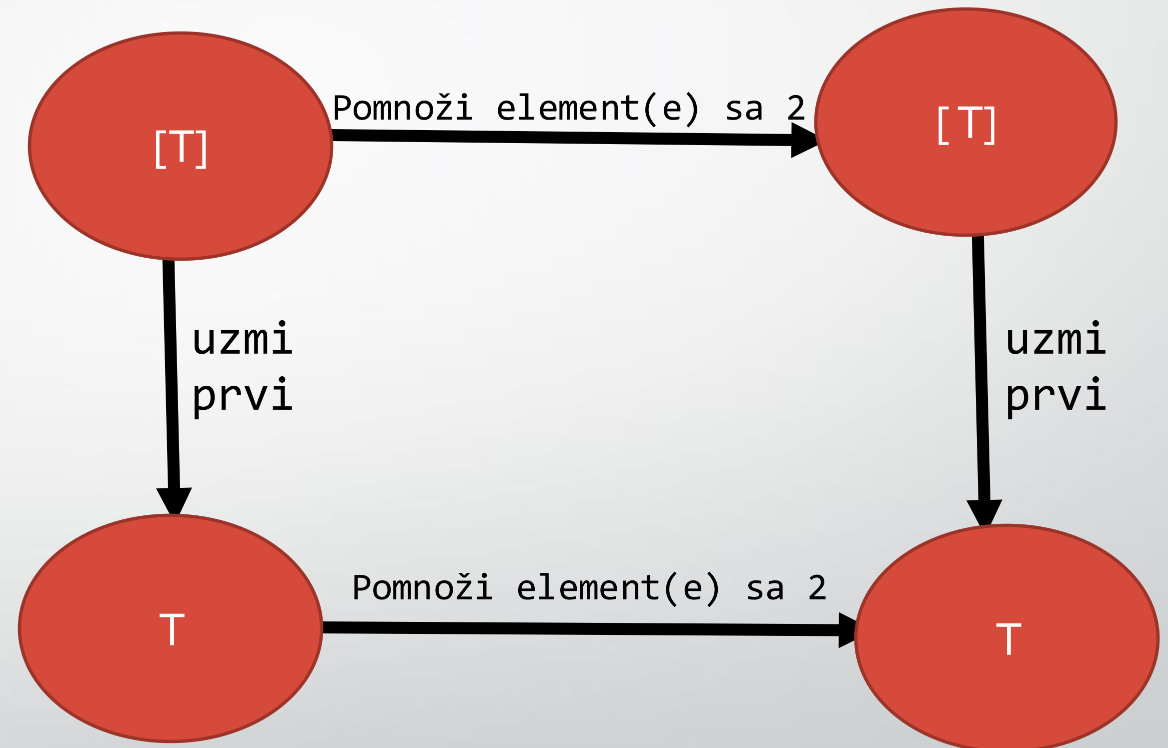


Problem - Progres

- Ulaz: [Int]
- Izlaz: Int

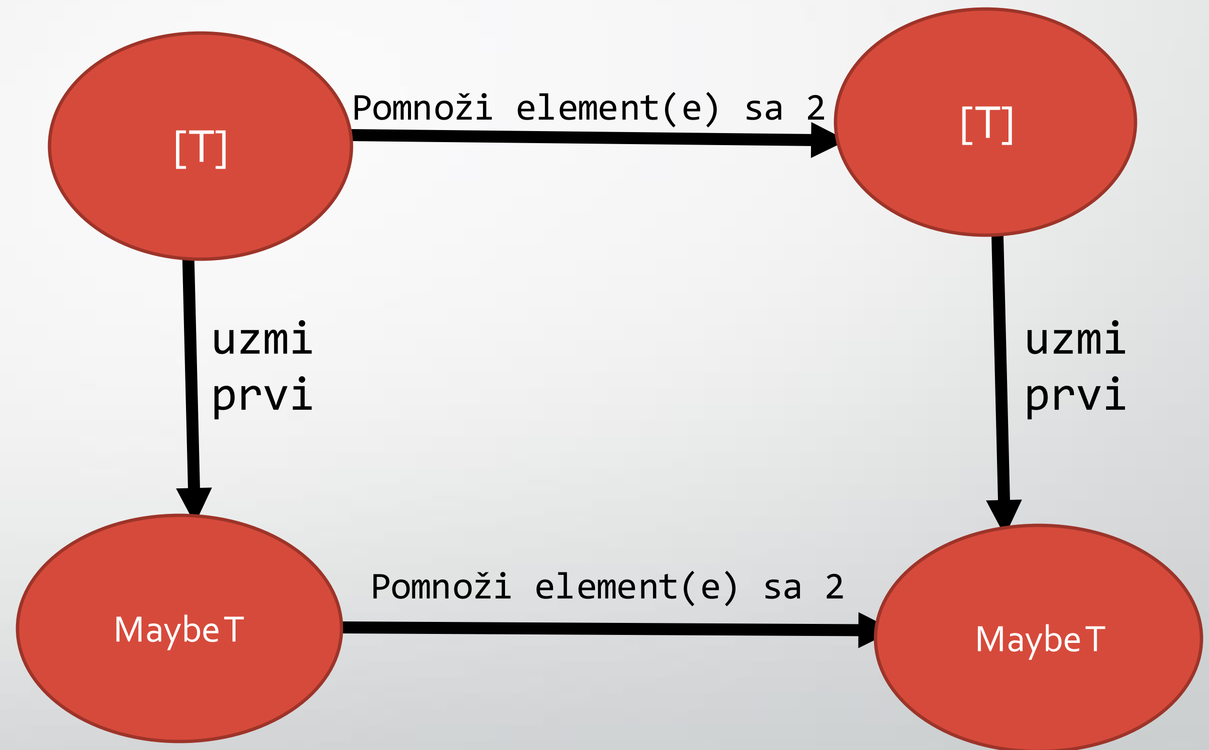
- Ne moramo da se ograničavamo na tipove

- Šta se dešava ako je lista prazna?!



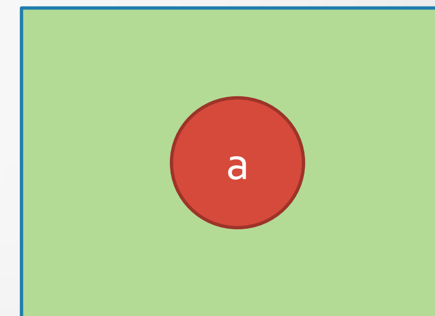
Problem - Progres

- Ulaz: [T]
- Izlaz: **Maybe** T
  - **Maybe** a = **Nothing** | **Just** a

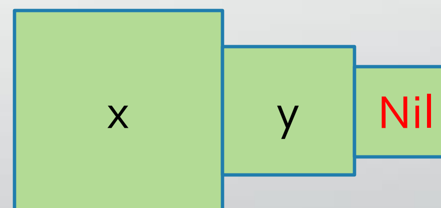


Kutije Sa Vrednostima

- **Maybe** a = **Nothing** | **Just** a



- List a = Nil | Cons a (List a)



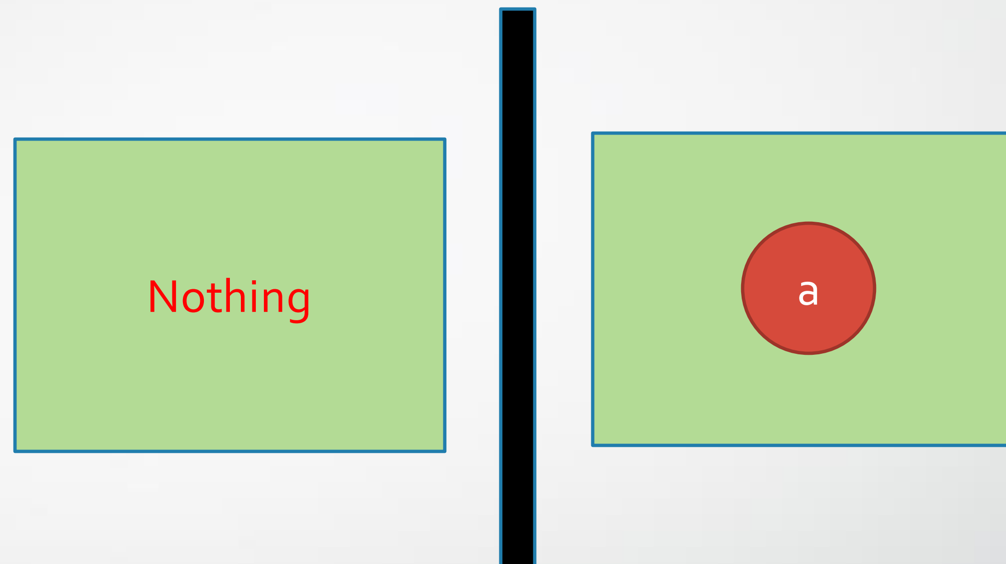
Funkcija headM

Haskell

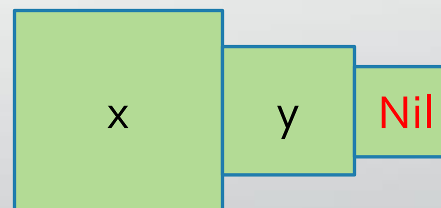
```
headm :: [a] -> Maybe a
headm []     = Nothing
headm (x:xs) = Just x


headm [1,2,3] = Just 1
headm [] = Nothing
```

Kutije Sa Vrednostima

- **Maybe** a = **Nothing** | **Just** a



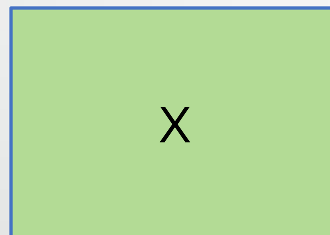
- List a = Nil | Cons a (List a)



- **Nije sigurno pristupati elementima kutija**

Pojam Funktora

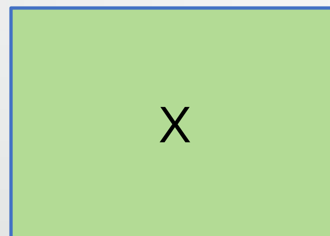
- Kako da sigurno pristupimo vrednosti koja možda ne postoji?

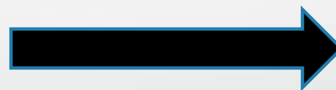


Pojam Funktora

- Kako da sigurno pristupimo vrednosti koja možda ne postoji?
  - "Ubacivanjem" operacije u kutiju

fmap f



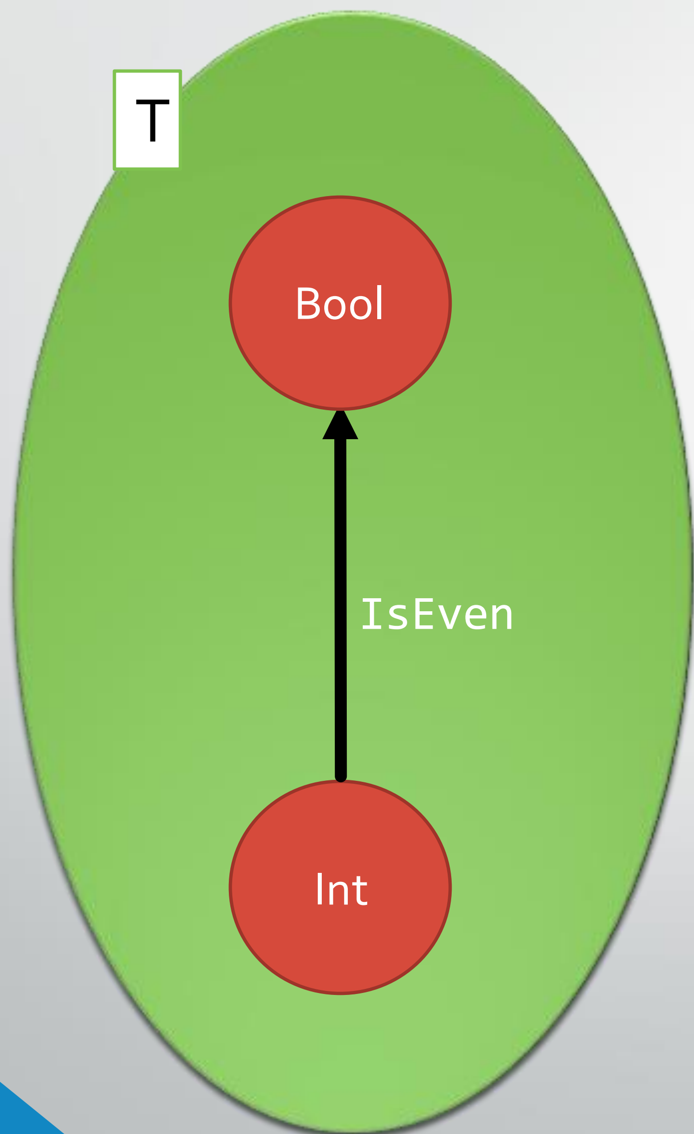


Primer funktora

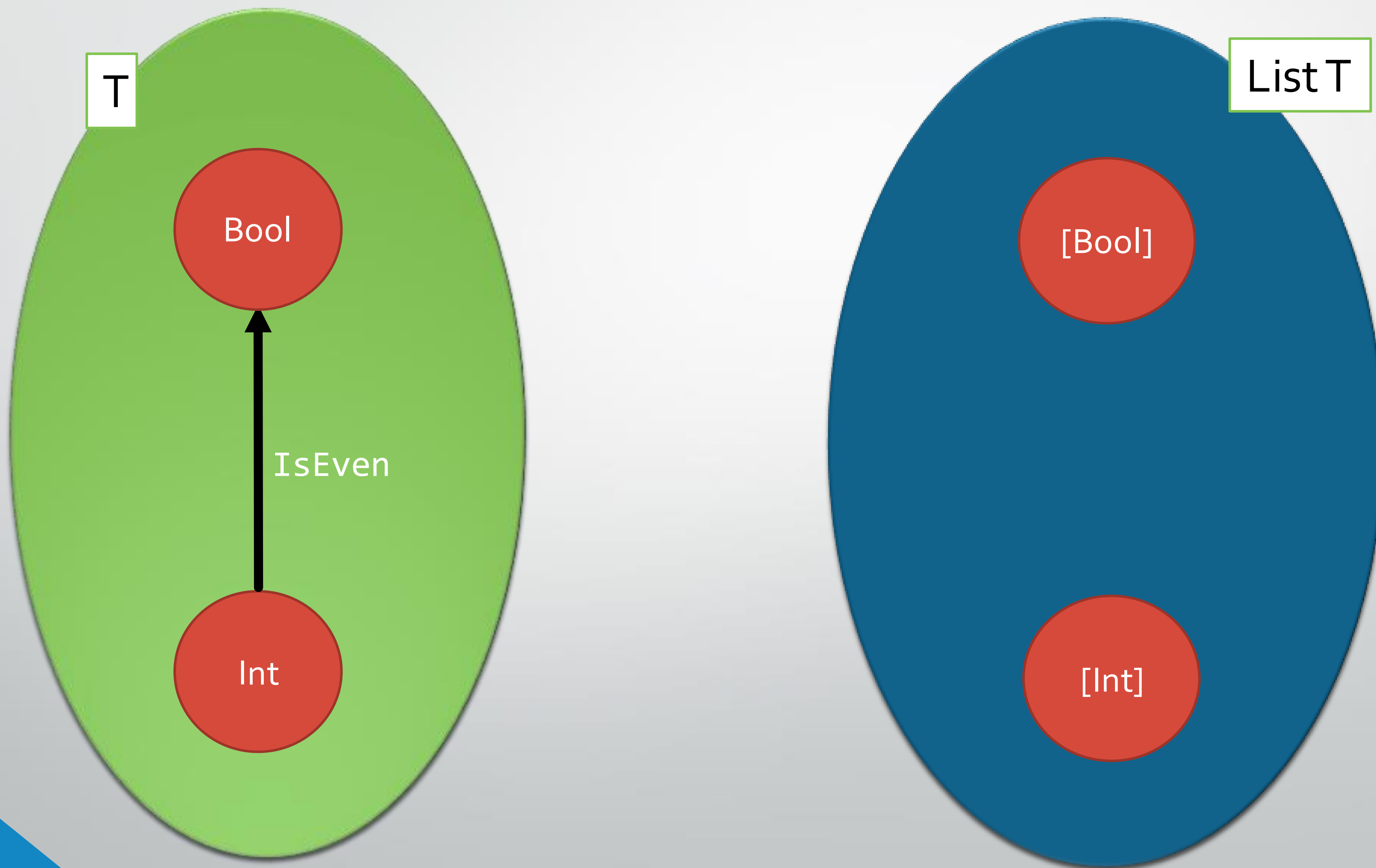
Haskell

```
fmap (*2) [1,2,3]

     = [2,4,6]


fmap (*2) (Just 5)

     = Just (2 * 5)
     = Just 10
```

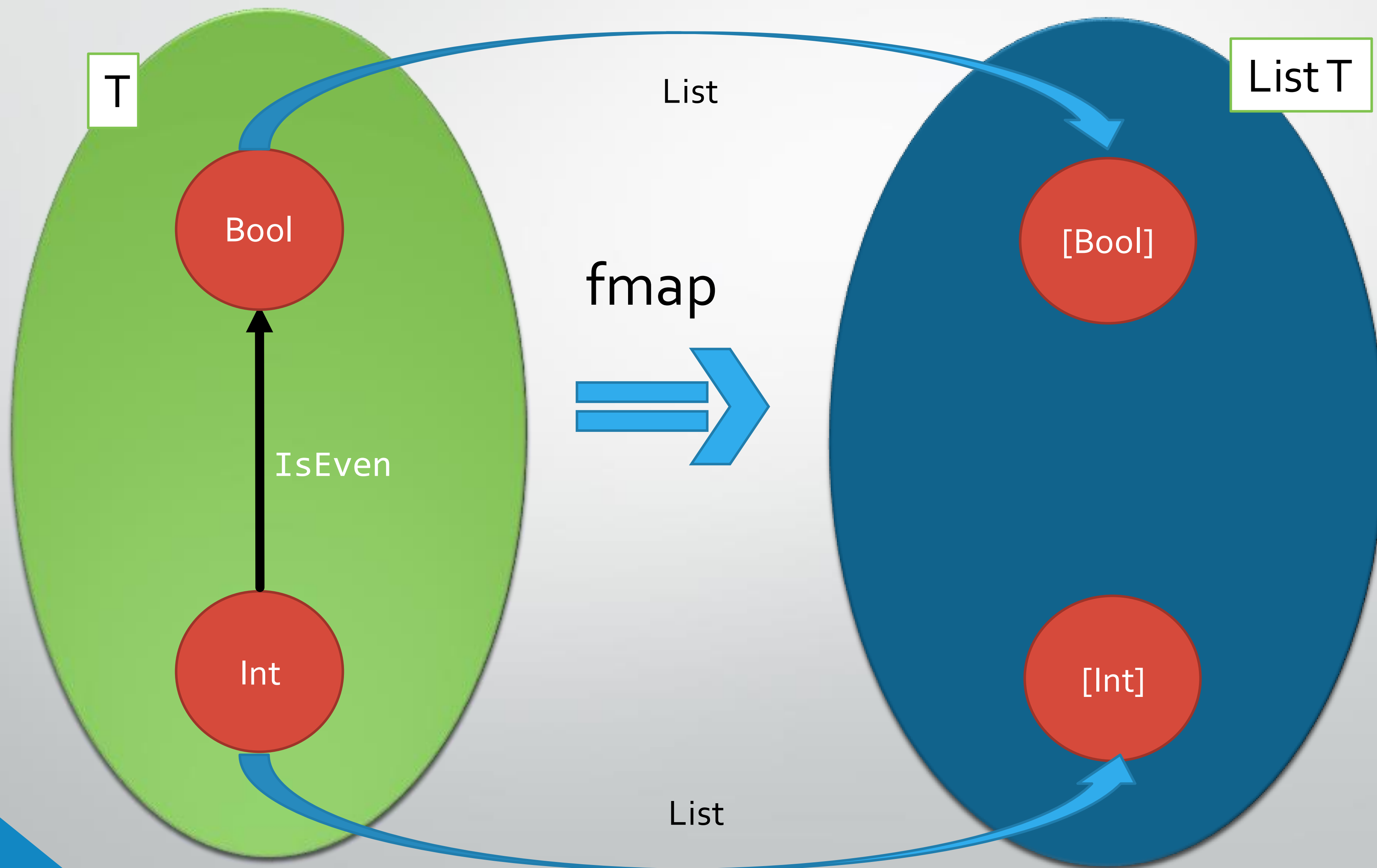
Pojam Funktora



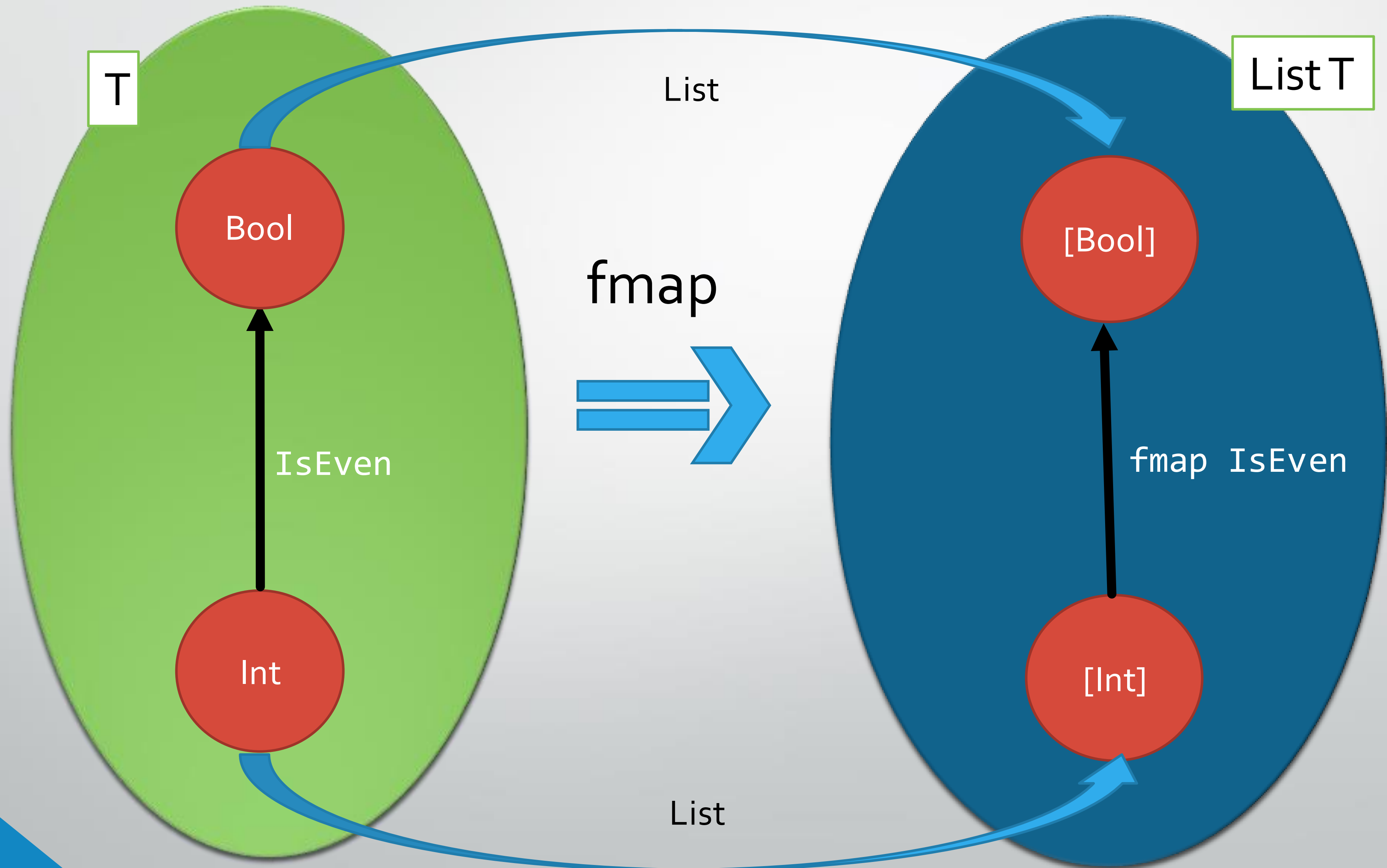
Pojam Funktora



Pojam Funktora



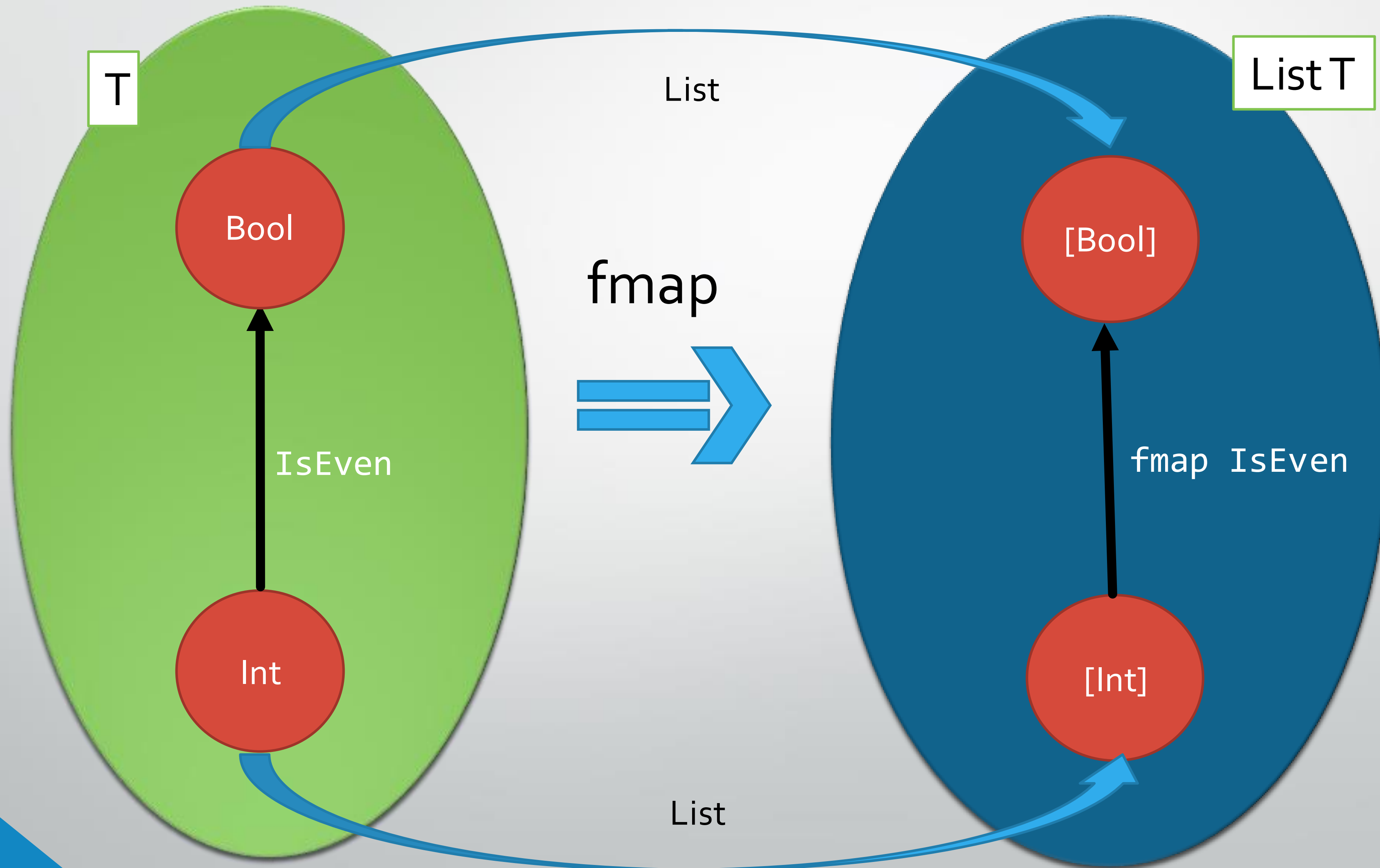
Pojam Funktora



Pojam Funktora

- **Mapiranje** između kategorija
- **Održava** strukturu kategorija

Pojam Funktora



Primer Funktora

```
Array<Int> numbers {1,2,3}
Array<Bool> result;
For(number in numbers)
    result.add(IsEven(number))
```

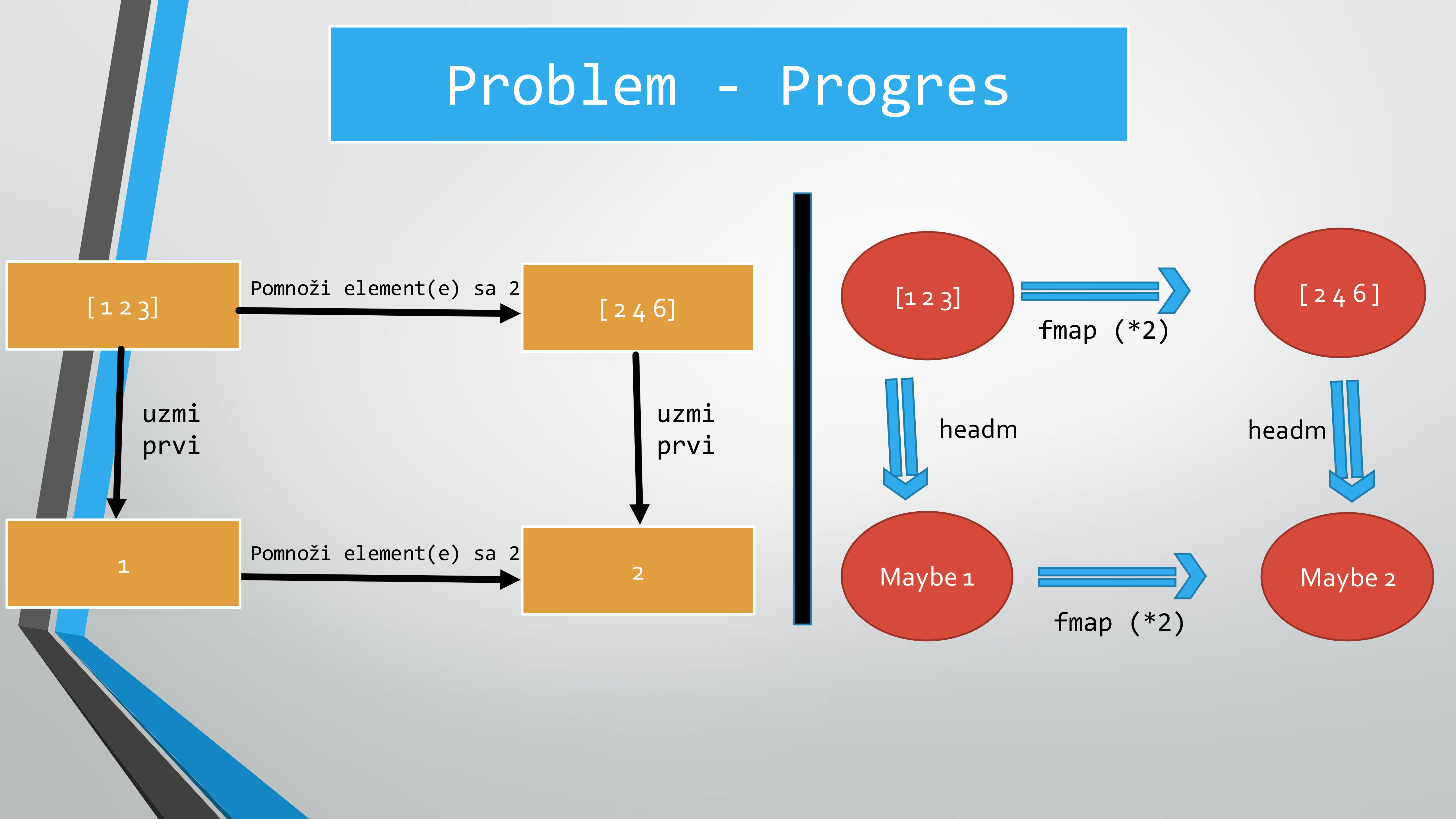
Primer Funktora

```
Array<Int> numbers {1,2,3}
Array<Bool> result;
For(number in numbers)
    result.add(IsEven(number))
```

Haskell

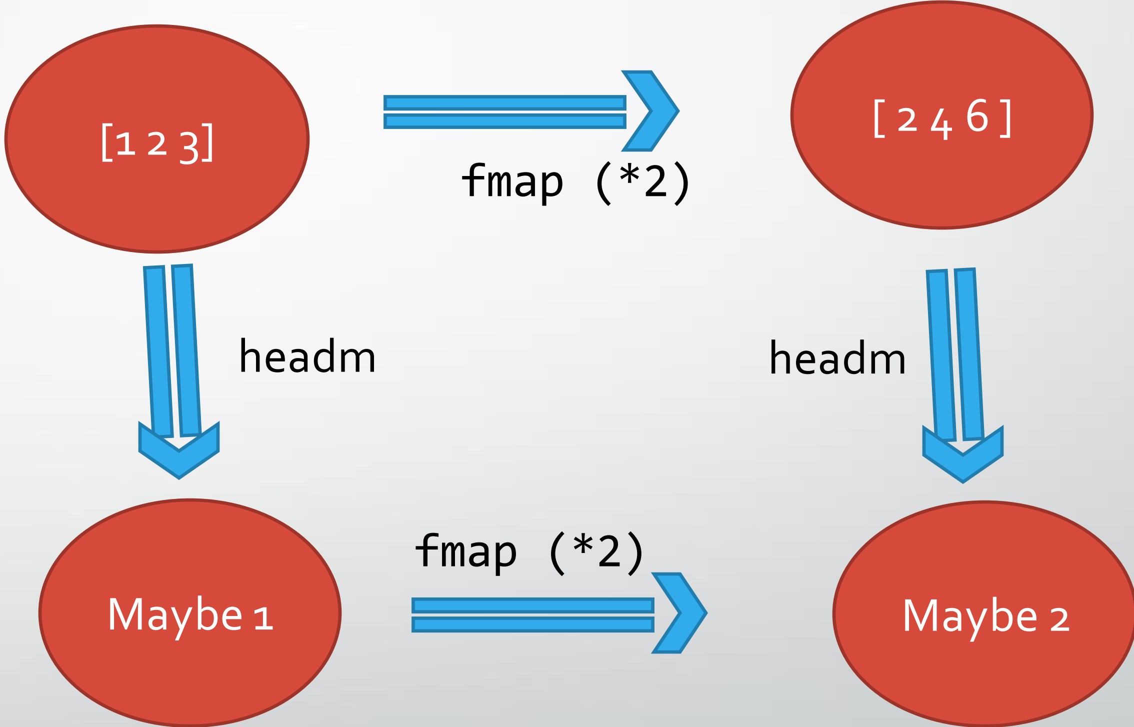
```haskell
isEven :: Int -> Bool
isEven x = (==0).(mod 2)
```

```haskell
fmap isEven [1,2,3]

= [False, True, False]
```


Problem - Progres
[1 2 3]
Pomnoži element(e) sa 2
[2 4 6]
uzmi prvi
uzmi prvi
1
Pomnoži element(e) sa 2
2
[1 2 3]
fmap (*2)
[2 4 6]
headm
headm
Maybe 1
fmap (*2)
Maybe 2

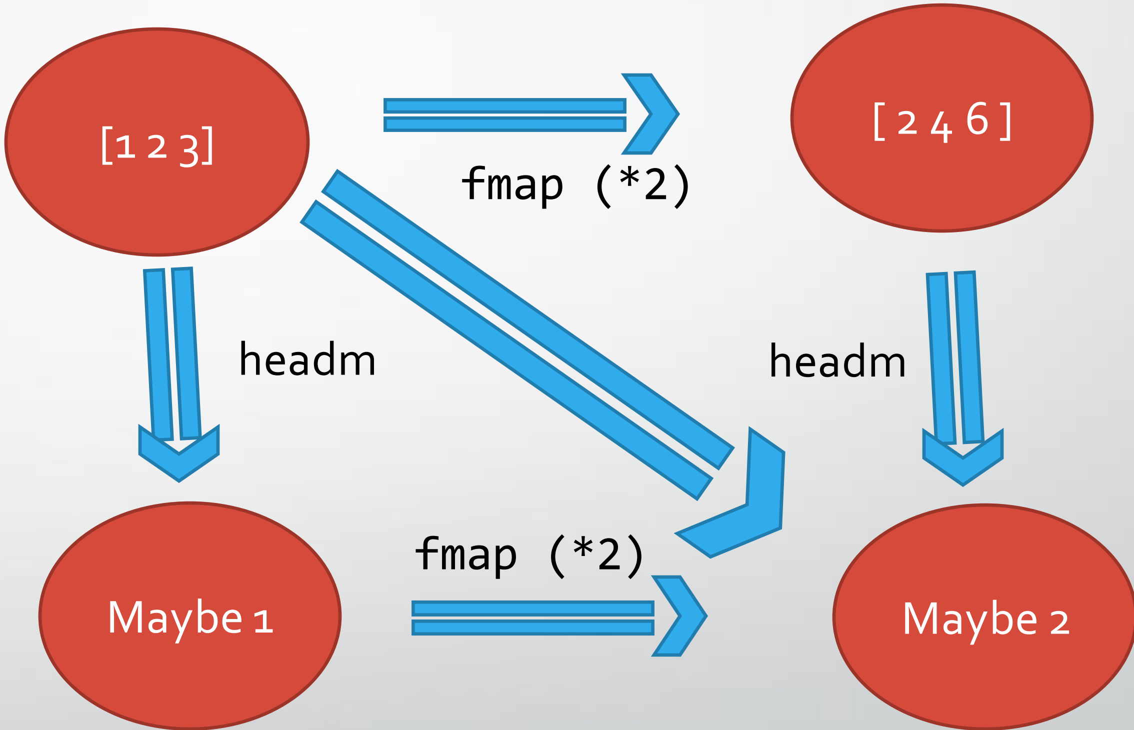
Problem - Progres

- **Dijagram komutira** – oba "puta" su korektna, samo je potrebno izabrati jeftiniji



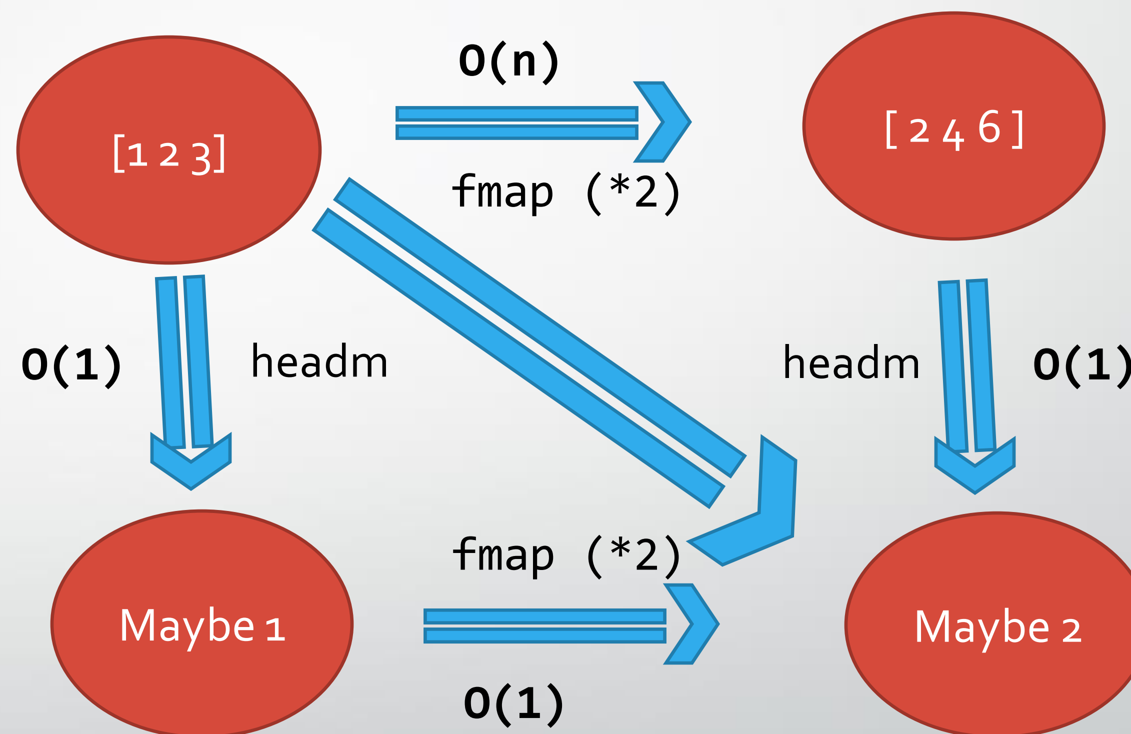
Problem - Progres

- **Dijagram komutira** – oba "puta" su korektna, samo je potrebno izabrati jeftiniji



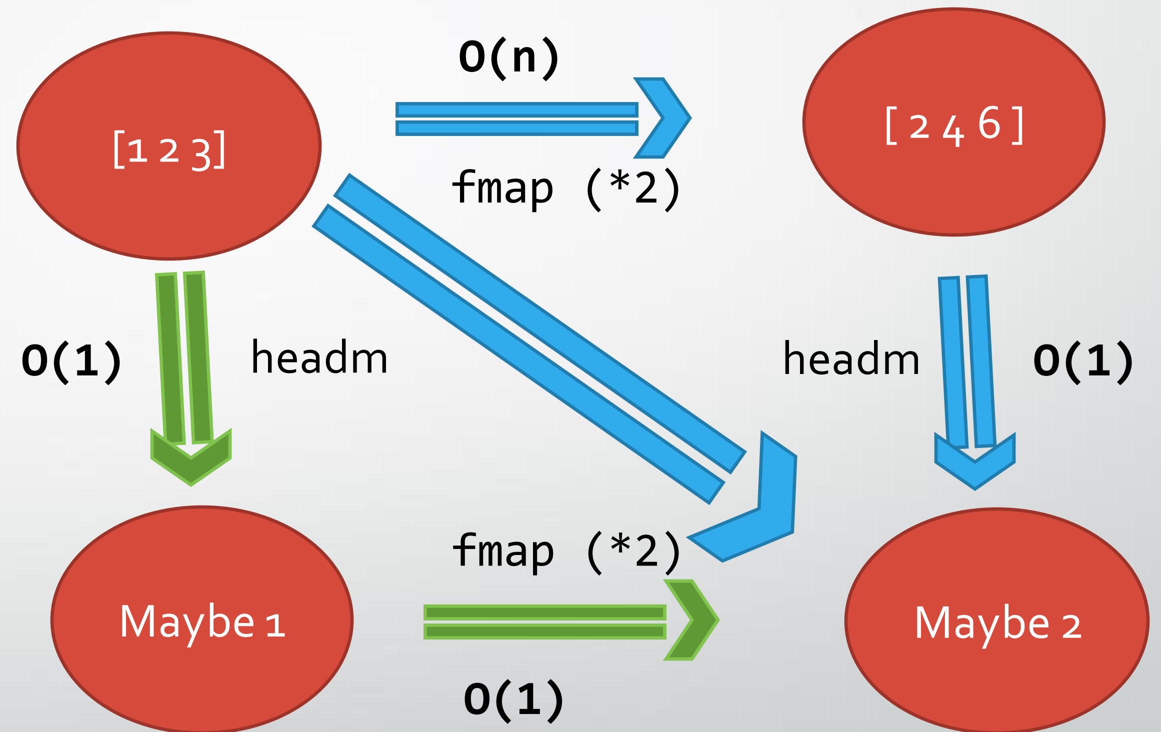
Dijagram optimizacije

- **Dijagram komutira** – oba "puta" su korektna, samo je potrebno izabrati jeftiniji
- Ugrađena tabela "težina" funktora (O notacija)



Dijagram optimizacije

- **Dijagram komutira** – oba "puta" su korektna, samo je potrebno izabrati jeftiniji
- Ugrađena tabela "težina" funktora (O notacija)
- Dešava se u fazi "deforestation" ("fusion")



Zaključak

- **Zašto možemo da uradimo ovakvu optimizaciju?**

Zaključak

- **Zašto možemo da uradimo ovakvu optimizaciju?**
- Kompozicije osiguravaju da strelice postoje i da se dijagrami grade iz prostih interakcija

Zaključak

- **Zašto možemo da uradimo ovakvu optimizaciju?**

- Kompozicije osiguravaju da strelice postoje i da se dijagrami grade iz prostih interakcija
- Pošto su sve kolekcije u programiranju funktori, održavaju strukturu pri transformaciji

Zaključak

- **Zašto možemo da uradimo ovakvu optimizaciju?**

- Kompozicije osiguravaju da strelice postoje i da se dijagrami grade iz prostih interakcija
- Pošto su sve kolekcije u programiranju funktori, održavaju strukturu pri transformaciji
- Komutirajući dijagrami nude priliku da prenesemo strukturu iz tipova u kompleksnost

Zaključak

- Da li se ovo može uraditi van funkcionalnog programiranja?

Zaključak

- Da li se ovo može uraditi van funkcionalnog programiranja?
- Odgovor je **DA**!

Zaključak

- Funkcije moraju biti čiste
  - nema sporednih efekata - mutacija stanja, globalne promenjive

Zaključak

- Funkcije moraju biti čiste
  - nema sporednih efekata - mutacija stanja, globalne promenjive
- Tipski sistem mora biti strog

Zaključak

- Funkcije moraju biti čiste
  - nema sporednih efekata - mutacija stanja, globalne promenjive
- Tipski sistem mora biti strog
- Olakšava ako se funkcije mogu lako kompozirati
  - kao sto u Haskell-u možemo da kažemo f = g . h , bez da navodimo parametre i da te funkcije pozivamo

Ostali jezici

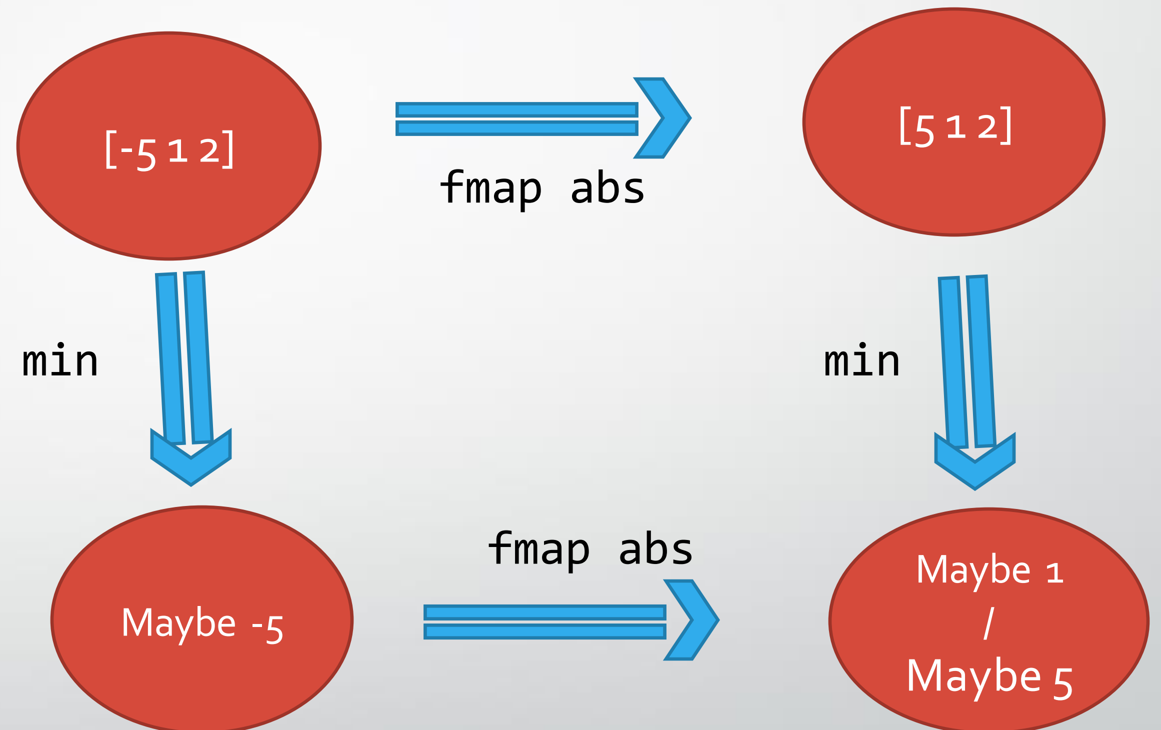
- C++ jezik kroz standarde teži ka tome
- Uvedeno dosta funktora
  - std::transform -> fmap u Haskell-u
  - std::optional -> Maybe u Haskell-u
- Ova optimizacija će se videti uskoro vremenom i u drugim jezicima

Pitanja?

Hvala na pažnji!

Dodatak 1

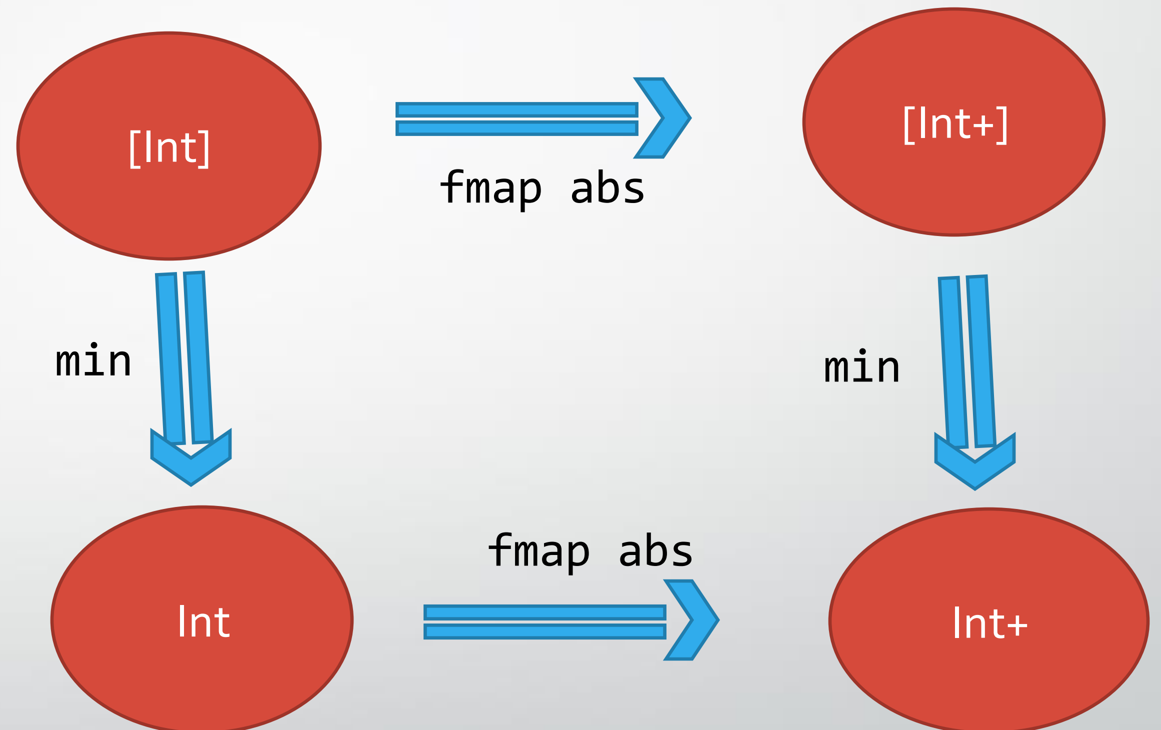
- Ulaz: [Int]
- Izlaz: Maybe Int
  - Najmanji element po apsolutnoj vredonsti?
  - Absolutna vrednost najmanjeg elementa?



Dodatak 1

- Ulaz: [Int]
- Izlaz: Maybe Int
  - Najmanji element po apsolutnoj vrednosti?
  - Apsolutna vrednost najmanjeg elementa?

- Nemamo optimizaciju zbog različitih domena među-rezultata!



Literatura

- Category Theory For Programmers – Bartosz Milewski

- Video materijali:
  - Category Theory in Life - Eugenia Cheng
  - Category theory playlist - Bartosz Milewski

Diagram