Konačno pridruživanje vrijednosti, zaključivanje tipova, lambda izrazi

Matej Mihelčić

Prirodoslovno-matematički fakultet, Sveučilište u Zagrebu

matmih@math.hr

15. studenoga, 2022.

Konačno pridruživanje vrijednosti varijabli

Svaka **lokalna varijabla** i svaki `final` **član** moraju imati **konačno pridijeljenu vrijednost** kada se dogodi **bilo kakav pristup njihovim vrijednostima**. Pristup vrijednosti je **navođenje imena varijabli** ili **imena elementa člana klase** u obliku `this.ime` koje se javlja u **proizvoljnom izrazu osim u izrazu pridruživanja kao lijevi operand** (onaj u koji se pridružuje vrijednost). Ukoliko prije pristupa lokalnoj varijabli ili `final` elementu članu klase **nije konačno pridružena vrijednost**, **dolazi do greške pri prevođenju**. `final` elementima se vrijednost smije pridružiti **samo jednom** ili dolazi do greške pri prevođenju (neposredno prije prvog i jedinog pridruživanja se mora dogoditi **konačno oddruživanje**).

- **Konačno pridruživanje:** pridruživanje vrijednosti lokalnoj varijabli ili `final` elementu članu klase (kojem još nije pridružena vrijednost) se mora dogoditi na svakom mogućem smjeru izvršavanja do pristupa vrijednosti.
- **Konačno oddruživanje:** niti jedno drugo pridruživanje se ne smije dogoditi `final` elementu članu klase (kojem još nije pridružena vrijednost) niti na jednom mogućem smjeru izvršavanja do naredbe pridruživanja.

Konačno pridruživanje vrijednosti varijabli

Pri analizi toka se uzimaju u obzir **struktura naredbi i izraza**, međutim ne uzimaju se u obzir **vrijednosti izraza**.

```
1 {
2     int k; //lokalna varijabla kojoj pristupamo
3     if (v > 0 && (k = System.in.read()) >= 0)
4         System.out.println(k); //pristup lokalnoj varijabli
5 }
```

Primjer analize toka.

U primjeru iznad, prevodioc može zaključiti da je vrijednost lokalne varijable k definitivno pridružena prije pristupa vrijednosti u liniji 4 zato što se prije pristupa vrijednosti nužno mora izvršiti naredba `(k = System.in.read()) >= 0`. Primjetite da ta činjenica **ne ovisi o vrijednostima varijable v ili učitane vrijednosti** već vrijedi uvijek (ili ne pristupamo vrijednosti od k ili se izvrši naredba pridruživanja vrijednosti varijabli k).

Konačno pridruživanje vrijednosti varijabli

```
1 {
2     int k; //lokalna varijabla
3     while (true) {
4         k = n; //pridruzivanje
5         if (k >= 5) break; //koristenje vrijednosti
6         n = 6;
7     }
8     System.out.println(k);
9 }
```

Primjer analize toka.

U gornjem primjeru, prevodioc zaključuje da će se naredba `k=n` uvijek izvršiti pošto je uvijet `while` petlje konstanta `true`. Stoga je vrijednost lokalne varijable k konačno pridružena prije korištenja vrijednosti u liniji 5.

Konačno pridruživanje vrijednosti varijabli

```
1 {
2     int k; //lokalna varijabla
3     while (n < 4) {
4         k = n;
5         if (k >= 5) break; //pristup vrijednosti
6         n = 6;
7     }
8     System.out.println(k); //pristup vrijednosti
9 }
```

Primjer analize toka.

U gornjem primjeru vrijednost **nije konačno pridružena** lokalnoj varijabli k prije pristupa njezinoj vrijednosti. **Postoji tok izvršavanja** u kojem se tijelo `while` petlje **ne izvrši** a pristup vrijednosti od k **se izvrši**. Takav tok se odvije ukoliko je $n \geq 4$ neposredno prije izvršavanja `while` petlje.

Konačno pridruživanje vrijednosti varijabli

```
1 {
2     int k;//lokalna varijabla
3     int n = 5;
4     if (n > 2)
5         k = 3;//pridruzivanje
6     System.out.println(k); //pristup vrijednosti
7 }
```

Primjer analize toka.

U gornjem primjeru vrijednost **nije konačno pridružena** lokalnoj varijabli *k* zbog toga što se pri analizi toka **ne promatraju vrijednosti izraza**. Stoga se **ne može utvrditi** da je **uvijek** `n>2` iako to uvijek vrijedi (bez mijenjanja izvornog koda) pošto imamo naredbu `n=5`.

Konačno pridruživanje vrijednosti varijabli

```
1 void f(boolean uvjet) {
2     int k; //lokalna varijabla
3     if (uvjet) k = 3; //pridruzivanje
4     else k = 4; //pridruzivanje
5     System.out.println(k); //pristupanje vrijednosti
6 }
```

Primjer analize toka.

U gornjem primjeru je vrijednost varijable *k* **konačno pridružena** zato što dolazi do pridruživanja **neovisno o vrijednosti** varijable `uvjet`.

```
1 void f(boolean uvjet) {
2     int k; //lokalna varijabla
3     if (uvjet) k = 3; //pridruzivanje
4     if (!uvjet) k = 4; //pridruzivanje
5     System.out.println(k); //pristupanje vrijednosti
6 }
```

Primjer analize toka.

U sličnom primjeru, previodioc **ne može zaključiti da uvijek dolazi do pridruživanja vrijednosti** prije pristupa u liniji 5.

Konačno pridruživanje vrijednosti varijabli

```
1  void g(boolean uvjet) {
2      final int k; //final varijabla
3      if (uvjet) {
4          k = 3; //pridruzivanje vrijednosti
5          System.out.println(k); //pristup vrijednosti
6      }
7      else {
8          k = 4; //pridruzivanje vrijednosti
9          System.out.println(k); //pristup vrijednosti
10     }
11 }
```

Primjer analize toka.

U gornjem primjeru se **može utvrditi** da je varijabla *k* **oddružena**, odnosno da **ne postoji tok kojim bi se dogodilo pridruživanje prije pridruživanja u linijama** 4 i 8, te da se pridruživanje izvršava **samo jednom** (neovisno o vrijednosti zastavice `uvjet`).

Konačno pridruživanje vrijednosti varijabli

```
1  void g(boolean uvjet) {
2      final int k; //final varijabla
3      if (uvjet) {
4          k = 3; //pridruzivanje vrijednosti
5          System.out.println(k); //pristup vrijednosti
6      }
7      if (!uvjet) {
8          k = 4; //pridruzivanje vrijednosti
9          System.out.println(k); //pristup vrijednosti
10     }
11 }
```

Primjer analize toka.

U sličnom primjeru, prevodioc **ne može zaključiti** da je varijabla k oddružena stoga **dolazi do greške pri prevođenju**.

Zaključivanje tipova

Zaključivanje tipova je potrebno prilikom **raznih analiza** tijekom prevođenja programa. Primarno, **testovi primijenjivosti generičkih metoda** i **zaključivanje tipa** kod **poziva generičkih metoda**.

Zaključivanje tipova se vrši kroz tri glavna postupka:
- **Redukcija:** **reducira formule uvjeta** (tvrdnje koje bi trebale vrijediti za kompatibilnost izraza i tipova) u **skupove ograda nad varijablama za koje zaključujemo tip**. Često se događa da se **formule uvjeta reduciraju u nove formule uvjeta**, pa se redukcija primjenjuje rekurzivno dok ne dođemo do **ograda**. Cilj je doći do **skupa ograda** koje **definiraju uvjete** koji **zadovoljavaju sve formule uvjeta**.
- **Uključivanje:** **održava skup ograda za varijable** za koje zaključujemo tip te **pazi da su nove ograde konzistentne sa postojećima**. **Neke ograde varijable** mogu utjecati na **izbor ograda** drugih (ovisnih) varijabli, pa se informacije o granicama **dijele** između ovisnih varijabli.
- **Rezolucija:** utvrđuje **konačan oblik varijable**, poštujući sve ograde iz skupa. Utvrđuje i **ispravan poredak izvrednjavanja** ovisnih varijabli.

Uvjetne formule i ograde

Uvjetne formule su tvrdnje o **kompatibilnosti ili relacijama podtipa** koje mogu uključivati varijable kojima zaključujemo tip. Uvjetne formule imaju jedan od oblika $<$ Izraz $\rightarrow T >$ (izraz je kompatibilan s T u kontekstu relaksiranog poziva), $< S \rightarrow T >$ (tip S je kompatibilan s T u kontekstu relaksiranog poziva), $< S <: T >$ (referencirani tip S je podtip referenciranog tipa T), $< S \leq T >$ (argument tipa S je sadržan argumentom tipa T), $< S = T >$ (tip T je identičan tipu S), $<$ LambdaIzraz $\rightarrow_{javlja} T >$ (iznimka koja se provjerava je prijavljena od lambda izraza i tipa je koji je izveden iz T), $<$ ReferencaMetode $\rightarrow_{javlja} T >$ (iznimka koja se provjerava je prijavljena od lambda izraza i tipa je koji je izveden iz T).

Ograde su **ograničenja** koja vrijede za varijable kojima zaključujemo tip. Primjeri: $S = T$, gdje zaključujemo tip ili varijabli S ili varijabli T, $S <: T$, `false` itd.

Redukcija i uključivanje

Redukcija je **postupak pretvorbe** skupa formula uvjeta u skup ograda. Formule uvjeta se **obrađuju redom**. Rezultat su **nove formula uvjeta ili skup ograda**. Formule uvjeta se **rekurzivno reduciraju** dok se ne dobije konačan skup ograda (niti jedna formula uvjeta se više **ne može reducirati**). Od tipova uvjeta imamo: **uvjete kompatibilnosti izraza**, **uvjete kompatibilnosti tipova**, **uvjete podtipiranja**, **uvjete jednakosti tipa** i **uvjete iznimaka koje se provjeravaju**.

Uključivanje je postupak dodavanja novih granica u skup granica. Uključivanje se može dogoditi ukoliko skup granica sadrži komplementarne formule ili ukoliko može doći do supstituiranja tipova kod granica (čime se mogu dobiti nove granice).

Proces uključivanja i redukcije je povezan, stvaranjem novih formula uvjeta, dolazi do redukcije i stvaranja novih granica, nakon njihovog uključivanja, može doći do stvaranja novih granica i formula uvjeta itd. Proces u nekom trenutku doseže fiksnu točku (nije moguće stvaranje novih granica).

Rezolucija

Kod skupa ograda koje ne sadrže ogradu `false`, za dio varijabli kojima se zaključuje tip **može doći do određivanja tipa**. U tom slučaju se **zaključeni tipovi mogu dodati u skup ograda**. To može dovesti do **novih formula, uvjeta i ograda** koje vode do **zaključivanja preostalih tipova**. Ovisnosti u skupu ograda mogu uvesti **nužni redoslijed određivanja tipova varijabli** ili **uzrokovati nužno zaključivanje tipova dodatnih varijabli**.

Lambda izrazi

Lambda izrazi su oblika LambdaParametri $\rightarrow$ TijeloLambdaIzraza. Imaju **sličnu funkcionalnost metodi**, tijelo lambda izraza je izraz ili blok izražen u ovisnosti o parametrima. Lambda izrazi su **uvijek ovisni o kontekstu** i **smiju se javljati samo u kontekstu pridruživanja, poziva metode ili pretvorbe**.

Izvrednjavanje lambda izraza **kreira instancu funkcijskog sučelja**, međutim ne uzrokuje izvršavanje tijela lambda izraza već se to može dogoditi i kasnije **kada se pozove odgovarajuća metoda funkcijskog sučelja**.

```
1 () -> {}  // bez parametara, rezultat je void
2 () -> 42  // bez parametara, tijelo je izraz
3 () -> null  // bez parametara, tijelo je izraz
4 () -> { return 42; }  // bez parametara, tijelo je blok sa
    return naredbom
5 () -> { System.gc(); }  // bez parametara, tijelo je void
    block
```

Primjer lambda izraza.

Lambda izrazi

```
() -> {     // Kompleksni blok sa return
  if (true) return 12;
  else {
    int result = 15;
    for (int i = 1; i < 10; i++)
      result *= i;
    return result;
  }
}
(int x) -> x+1   // Parametar deklariranog tipa
(int x) -> { return x+1; }  // Parametar deklariranog tipa
(x) -> x+1   // Parametar zakljucenog tipa
x -> x+1   // Zagrade su opcionalne za jedan parametar
    zakljucenog tipa
(String s) -> s.length() // Parametar deklariranog tipa
(Thread t) -> { t.start(); }  // Parametar deklariranog tipa
s -> s.length()   // Parametar zakljucenog tipa
t -> { t.start(); }  // Parametar zakljucenog tipa
```

Primjer lambda izraza.

Lambda izrazi

```
1 (int x, int y) -> x+y  // vise parametara deklariranog tipa
2 (x, y) -> x+y  // vise parametara zakljucenog tipa
3 (x, int y) -> x+y    // Nije dopusteno mijesati deklarirane
    i zakljucene tipove
4 (x, final y) -> x+y  // Nije dopusteno koristenje
    modifikatora (kao final) kod parametara sa tipom koji se
     odreduje zakljucivanjem
```

Primjer lambda izraza.

```
1 interface Sucelje1 { int f(int x); }
2 interface Sucelje2{ int f(); }
3
4 Sucelje1 s = (int x)->{return x;};
5      System.out.println(s.f(5)+3);  //8
6      s= (int x)->{
7          if(x>0) return x*x;
8          else return Math.abs((int)Math.pow(x, 3)); };
9      System.out.println(s.f(-10)); //1000
```

Korištenje lambda izraza.

Lambda izrazi

```
1  Sucelje2 s1 = ()->42;
2  System.out.println(s1.f()); // 42
3  s1 = ()->{return 54;};
4  System.out.println(s1.f()); // 54
5
6  interface Sucelje3{ double f(double x, int ...y);}
7
8  Sucelje3 s4= (double x, int ...y)->{
9          double res = 0.0;
10         for(int pot:y){
11             res+=Math.pow(x, pot);
12         }
13         return res;
14     };
15
16     System.out.println(s4.f(2.5, 1,2,3,4)); //63.4375 =
   2.5^1+2.5^2+2.5^3+2.5^4
```

Korištenje lambda izraza.

Kod lambda izraza **nije dozvoljeno** deklariranje parametara tipa.

Lambda izrazi

Dopušteno je korištenje **samo jednog parametra varijabilnog broja** koji se **mora nalaziti na zadnjem mjestu** u listi parametara.
Svaki parametar lambda izraza ima ili **zaključeni** ili **deklarirani** tip.
Parametar koji je deklariran ključnom riječi `var` ili imenom varijable ima **tip koji se zaključuje**. Tip se zaključuje iz **tipa deklariranog u funkcijskom sučelju** kojem se pridružuje lambda izraz. Lambda izrazi `(int... x)` $\rightarrow$ `TIJELO` i `(int[] x)` $\rightarrow$ `TIJELO` su identični.

Nije dopušteno korištenje `final` za više od jednog parametra ili dodijeljivanje vrijednosti `final` parametru unutar tijela lambda izraza. Nije dopušteno deklariranje dva parametra istog imena. Ukoliko je parametar tipa `var`, nije dopušteno korištenje `[]`.

Lambda izrazi

Tijelo lambda izraza je **jedan izraz ili blok**. Tijelo lambda izraza **naznačava kod koji će se izvršiti kada se dogodi poziv**. Tijelo lambda izraza može biti *void-kompatibilno* (svaka naredba `return` je prazna `return;`) ili *vrijednosno-kompatibilno* (svaka naredba `return` je oblika `return Izraz;` i `return` se uvijek pozove - blok naredbi **ne završava normalno**). Ukoliko tijelo lambda izraza nije niti *void-kompatibilno* niti *vrijednosno-kompatibilno* **dolazi do greške pri prevođenju**.

```
1 () -> {} //lambda izraz je void-kompatibilan
2 () -> { System.out.println("tekst"); } //lambda izraz je
    void-kompatibilan
3
4 () -> { return "IzlazniString"; } //lambda izraz je
    vrijednosno-kompatibilan
5 () -> { if (...) return 1; else return 0; } //lambda izraz
    je vrijednosno-kompatibilan
```

Tipovi tijela lambda izraza.

Lambda izrazi

```
1 () -> { throw new RuntimeException(); } //lambda izraz je i
    void-kompatibilan i vrijednosno-kompatiblian (
    specificnost prijavljivanja RuntimeException).
2
3 () -> { if (...) return "nesto"; System.out.println("nesto
    drugo"); } //lambda izraz nije niti void-kompatibilan
    niti vrijednosno-kompatibilan stoga dolazi do greske pri
     prevodenju.
```

Tipovi tijela lambda izraza.

- **Lokalna varijabla koja je inicijalizirana** pri deklaraciji je efektivno `final` ako vrijedi sve od navedenog: a) **nije** deklarirana kao `final`, b) **nikada se ne javlja kao lijevi operand izraza pridruživanja** (ovdje se ne ubraja inicijalizacija pri deklaraciji), c) **nikada se ne javlja kao operand prefiks ili postfiks inkrement/dekrement operatora**.

Lambda izrazi

- **Lokalna varijabla koja nije inicijalizirana** pri deklaraciji je efektivno `final` ako vrijedi sve od navedenog: a) **nije** deklarirana kao `final`, b) kada se javlja kao lijevi operand u izrazu pridruživanja, tada **mora biti konačno oddijeljena** prije i nakon pridjeljivanja vrijednosti te **nesmije biti konačno pridijeljena prije tog dodijeljivanja vrijednosti ili nakon**, c) **nikada se ne javlja kao operand prefiks ili postfiks inkrement/dekrement operatora**.
- Metoda, konstruktor, lambda izraz ili parametar iznimke se u svrhu određivanja svojstva efektivno `final` **tretiraju kao lokalne varijable čiji deklarator ima inicijalizator**.

Ukoliko je varijabla efektivno `final`, tada dodavanje modifikatora `final` **neće uvesti greške pri prevođenju**. Isto tako, u ispravno definiranom programu, varijabla koja je `final`, **postaje efektivno** `final` **ukoliko se ukloni modifikator** `final`.