Principi programskih jezikov

Rešitve

V teh vajah obravnavamo kodo, ki sestoji *samo* iz majhnega delčka jave:

- osnovnih aritmetičnih in boolovih izrazov
- ukaza `print(e)`, ki je okrajšava za `System.out.print`
- prireditvenega stavka `x = e`
- zaporedja ukazov `A ; B`
- pogojnega stavka `if (P) { A } else { B }`
- in zanke `while (P) { A }` ter ukaza `break`

Pretvorba zanke `for` v zanko `while`

Lahko bi dodali zanko `for`, a je pravzaprav ne potrebujemo. Kako bi zanko

```
for (A; P; C) {
    B;
}
```

predelali v ekvivalentno kodo, ki uporablja samo `while`?

Rešitev

```
A;
while (P) {
    B;
    C;
 }
```

Pretvorba zanke `do` v zanko `while`

V `while` predelajte še zanko `do`:

```
do {
  A;
} while (p);
```

Rešitev

```
A;
while (P) {
    A;
}
```

Odstranjevanje mrtve kode

Mrtva koda (angl. *dead code*) je del kode, za katerega zagotovo vemo, da se ne bo izvedel. Tako kodo lahko brez škode odstranimo. Nekaj primerov odstranjevanja mrtve kode:

- Če vemo, da pogoj `p` v `if (p) { A } else { B }` zagotovo velja, potem se `B` ne izvede, zato lahko pogojni stavek poenostavimo v `p ; A`. Če nadalje vemo še, da `p` ne bo sprožil izjeme (zaradi deljenja z `0`), potem lahko poenostavimo kar v `A`.
- Če se v `A ; B` ukaz `A` nikoli ne konča, se `B` ne bo izvedel in ga lahko odstranimo.
- Če ob vstopu v `while (p) { A }` pogoj `p` *ne* velja, se zanka sploh ne izvede in jo lahko odstranimo.

Primer 1

Odstranite mrtvo kodo v programu

```
print("Kdor to bere je ");
a = 3;
if (a * a < 10) {
    print("mula");
} else {
    print("osel");
}
```

Rešitev

```
print("Kdor to bere je ");
print("mula");
```

Primer 2

Odstranite mrtvo kodo v programu, kjer ima `n` neznano celoštevilsko vrednost:

```
k = n + (n - 1);
if (k % 2 == 0) {
    print("foo");
} else {
    print("bar");
}
m = k * (-k);
while (m > 0) {
    m = m - 1;
    print(m);
}
```

Rešitev

```
k = n + (n - 1); // se zagotovo izvede
// opazimo, da je n + (n - 1) vedno liho število, torej se bo izvedel else
print("bar");
m = k * (-k); // se zagotovo izvede
// ker je k liho število, je n * (-n) negativno število,
// zanka while se ne izvede in jo lahko odstranimo
```

Primer 3

Odstranite mrtvo kodo v programu:

```
i = 0;
while (i < 100) {
    if (i % 2 == 0) {
        p = i * (i - 1) * (i - 2);
    } else {
        p = i * (i + 1) * (i + 2);
    }

    if (p % 3 == 0) {
        break;
    } else {
    }
    i = i + 1;
}

if (i >= 100) {
    print("ni zanimivih števil");
} else {
    print("našel sem zanimivo število " + i);
}
```

Rešitev

```
i = 0;
while (i < 100) {
    if (i % 2 == 0) {
        p = i * (i - 1) * (i - 2);
    } else {
        p = i * (i + 1) * (i + 2);
    }
    // p je zmnožek treh zaporednih števil, zato je deljiv s 3
    // zagotovo se izvede break
    break;
}
// i je enak 0
print("našel sem zanimivo šetvilo " + i);
```

Ta program bi lahko še *optimizirali*, ker se zanka `while` izvede samo enkrat. A to ne bi bilo več le odstranjevanje mrtve kode:

```
i = 0;
p = i * (i - 1) * (i - 2);
print("našel sem zanimivo število " + i);
```

Primer 4

Odstranite mrtvo kodo v programu:

```
s = 0;
i = 0;
while (i < 100) {
    s = s + i;
}
print(s);
```

Rešitev

```
s = 0;
i = 0;
while (i < 100) {
    s = s + i;
}
// zanka se nikoli ne konča, print se ne zgodi
```

Program, ki odstranja mrtvo kodo

V tej nalogi dodamo še uporabo tabel in funkcij. Dobri prevajalniki se trudijo odstraniti čim več mrtve kode.

*Ali lahko implementiramo program, ki vedno odstrani **vso** mrtvo kodo?*

Uporabi znanje iz teorije izračunljivosti in naslednjo opazko: če se v programu

```
A ;
B
```

ukaz `A` nikoli ne konča (se zacikla), potem je `B` mrtva koda.

Rešitev

Če bi imeli program `DEAD`, ki vedno odstrani vso mrtvo kodo, bi lahko implementirali program, ki ugotovi, ali se Turingov stroj ustavi. Za dani Turingov stroj `T` bi sestavili program:

```
simuliraj(T);
print("Ali se ustavi?")
```

nato pa bi s programom `DEAD` ugotovili, ali je stavek `print` mrtva koda. Če je, se stroj `T` ne ustavi, sicer se ustavi. Na ta način bi lahko algoritmično rešili problem zaustavitve (angl. *halting problem*), ki pa ni algoritmično rešljiv.

Odstranjevanje ukaza `break`

V tej nalogi bomo ugotovili, da se lahko ukaza `break` znebimo. Najprej naredimo nekaj primerov.

Primer 1

Naslednjo kodo predelajte v enakovredno kodo, ki ne uporablja ukaza `break`, pri čemer ima `n` neznano celoštevilsko vrednost:

```
while (n > 0) {
    digit = n % 10;
    if (digit == 7) {
        print(n + " vsebuje številko 7");
        break;
    } else { }
    n = n / 10;
}
```

Rešitev

```
stop = false;
while (!stop && n > 0) {
    digit = n % 10;
    if (digit == 7) {
        print(n + " vsebuje številko 7");
        stop = true;
    } else { }
    if (!stop) {
        n = n / 10;
    }
}
```

Primer 2

Naslednjo kodo predelajte v enakovredno kodo, ki ne uporablja ukaza `break`.

```
// najdi najmanjše popolno število
n = 1;
while (true) {
    d = 1;
    vsota = 0; // vsota deliteljev števila n
    while (d < n) {
        if (n % d == 0) {
            vsota = vsota + d;
        } else { }
        if (vsota > n) {
            break;
        } else { }
        d = d + 1;
    }
    if (vsota == n) {
        break;
    } else { }
    print(n + " ni popolno število");
    n = n + 1;
}
print("našel popolno število: " + n);
```

Rešitev

```
// najdi najmanjše popolno število
n = 1;
stop = false;
while (!stop) {
    d = 1;
    vsota = 0;
    stop = false;
    while (!stop && d < n) {
        if (i % d == 0) {
            vsota = vsota + d;
        } else { }
        if (vsota > n) {
            stop = true;
        } else { }
        if (!stop) {
            d = d + 1;
        }
    }
    stop = false;
    if (vsota == n) {
        stop = true;
    } else { }
    if (!stop) {
        print(n + " ni popolno število");
        n = n + 1;
    }
}
print("našel popolno število: " + n);
```

Splošni postopek

Opišite splošni postopek za odstranjevanje ukaza `break`. Na papirju definirajte funkcijo `UNBREAK`, ki sprejme program in vrne enakovredni program brez ukaza `break`. Funkcija je rekurzivna.

Rešitev

Uvedemo novo boolovo spremenljivko `stop` z začetno vrednostjo `false`. Če se `stop` že pojavlja v kodi, izberemo drugo ime, ki se še ne pojavi.

```
UNBREAK(x = e) :=
   x = e

UNBREAK(print(e)) :=
   print(e)

UNBREAK(break) :=
   stop = true

UNBREAK(A ; B) :=
   UNBREAK(A); if (!stop) { UNBREAK(B) } else { }

UNBREAK(if (p) { A } else { B }) :=
   if (p) { UNBREAK(A); } else { UNBREAK(B); }

UNBREAK(while (p) { A }) :=
   stop = false;
   while (!stop && p) {
     UNBREAK(A);
   }
   stop = false;
```

Izboljšava splošnega postopka

Izboljšajte `UNBREAK` tako, da ne bo po nepotrebnem spreminjal kode.

Rešitev

Podobno kot prejšnja rešitev, le da kode ne spremenimo, če ne vsebuje `break`:

```
UNBREAK(c) := c    če se break ne pojavi v c (ta primer zaobjema print in x = e)
```

sicer:

```
UNBREAK(break) :=
   stop = true

UNBREAK(A ; B) :=
   UNBREAK(A); if (!stop) { UNBREAK(B) } else { }

UNBREAK(if (p) { A } else { B }) :=
   if (p) { UNBREAK(A); } else { UNBREAK(B); }

UNBREAK(while (p) { A }) :=
   stop = false;
   while (!stop && p) {
     UNBREAK(A);
   }
   stop = false;
```

Zadnja sprememba: torek, 13. marec 2018, 10:46

◄ Vaje: ukazni programski jezik

Skok na...

Zapiski ►