

Generátorok előállítása

CPS-transzformációval Java nyelven

Bagossy Attila

Témavezetők: Dr. Battyányi Péter, Balla Tibor

2017. április 20.

Debreceni Egyetem, Informatikai Kar, Számítógéptudományi Tanszék

1. A generátor és rokonai
2. Generátorok ismert nyelvekben
3. Korábbi Java implementációk
4. Az eljárás egy példán keresztül

A generátor és rokonai

Szubrutin (*subroutine*)

Alkalmas kódrészletek kiemelésére, csökkentve a duplikációt.

Meghívása után a hívó kód végrehajtása felfüggesztésre kerül a visszatérésig.

Hívó

```
/*  
 * Kódrészlet  
*/  
add(2, 2);  
/*  
 * Kódrészlet  
*/
```

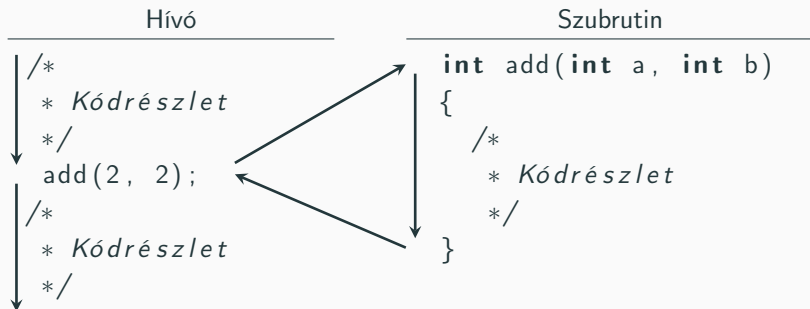
Szubrutin

```
int add(int a, int b)  
{  
    /*  
     * Kódrészlet  
     */  
}
```

Szubrutin (*subroutine*)

Alkalmas kódrészletek kiemelésére, csökkentve a duplikációt.

Meghívása után a hívó kód végrehajtása felfüggesztésre kerül a visszatérésig.



Korutin (*coroutine*)

A végrehajtás mindig ott folytatódik, ahol a legutóbbi hívás esetén abbamaradt.

A lokális változók értéke megőrződik a hívások között.

```
coroutine A()  
{  
    /* ... */  
    yield B;  
    /* ... */  
    yield B;  
}
```


```
coroutine B()  
{  
    /* ... */  
    yield A;  
    /* ... */  
}
```

Korutin (*coroutine*)

A végrehajtás mindig ott folytatódik, ahol a legutóbbi hívás esetén abbamaradt.

A lokális változók értéke megőrződik a hívások között.

```
coroutine A()  
{  
    /* ... */  
    yield B;  
    /* ... */  
    yield B;  
}  
  
coroutine B()  
{  
    /* ... */  
    yield A;  
    /* ... */  
}
```



Korutin (*coroutine*)

A végrehajtás mindig ott folytatódik, ahol a legutóbbi hívás esetén abbamaradt.

A lokális változók értéke megőrződik a hívások között.

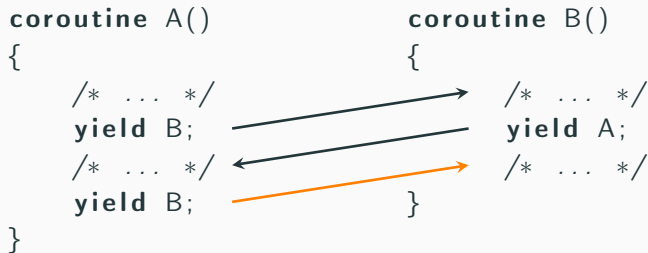
```
coroutine A()  
{  
    /* ... */  
    yield B;  
    /* ... */  
    yield B;  
}  
  
coroutine B()  
{  
    /* ... */  
    yield A;  
    /* ... */  
}
```

The diagram illustrates the execution flow between two coroutines, A and B. Coroutine A contains two yield statements, both labeled 'B'. Coroutine B contains one yield statement labeled 'A'. A black arrow points from the first 'yield B;' in A to the start of B, and an orange arrow points from the 'yield A;' in B back to the second 'yield B;' in A, showing the alternating execution between the two.

Korutin (*coroutine*)

A végrehajtás mindig ott folytatódik, ahol a legutóbbi hívás esetén abbamaradt.

A lokális változók értéke megőrződik a hívások között.



Generátor (*generator*)

Aszimmetrikus korutin, amely elemek sorozatát állítja elő.

Minden egyes hívás alkalmával a sorozat egy elemét képz.

```
for ch in alphabet()  
{  
    print ch  
}
```

```
generator alphabet()  
{  
    yield 'a';  
    yield 'b';  
    yield 'c';  
    /* ... */  
}
```

Generátorok ismert nyelvekben

Első, úttörő próbálkozások

IPL-V, Alphard

A `yield` kulcsszó bevezetése

CLU

TIOBE Top 10, 2017. március

- *C#*
- *Python*
- *Visual Basic .NET*
- *PHP*
- *JavaScript*

Végtelen sorozatok

- Fibonacci sorozat
- Prímszámok
- Véletlen értékek forrása

Véges sorozatok

- Reguláris kifejezésre illesztés eredményei
- Fájl beolvasása soronként/darabonként
- Paraméteres görbék pontjai

- Szimmetrikus korutinok modellezése
- `async/await` szimulálása (*JavaScript*)
- Mikroszálak létrehozása, ütemezése

Korábbi Java implementációk

Csak régebbi *Java* verziók támogatása (*Java SE* 6-7)

Csak régebbi *Java* verziók támogatása (*Java SE* 6-7)

Kényelmetlen interfész

Csak régebbi *Java* verziók támogatása (*Java SE* 6-7)

Kényelmetlen interfész

Rugalmatlan, kevésbé megbízható megvalósítás

```
1 @Continuable
2 public Iterable<Integer> power(int number, int exponent)
3 {
4     int counter = 0;
5     int result = 1;
6     while (counter++ < exponent) {
7         result = result * number;
8
9         System.out.print "[" + result + " ");
10
11         Yield.ret(result);
12     }
13     return Yield.done();
14 }
```

```
1 Generator<Integer> simpleGenerator =  
2   new Generator<Integer>() {  
3       public void run() throws InterruptedException {  
4           yield(1);  
5           /* ... */  
6           yield(2);  
7       }  
8   };
```

```
1 public Iterable<Integer>
2     power(final int number, final int exponent)
3 {
4     int counter = 0;
5     int result = 1;
6     while (counter++ < exponent) {
7         result = result * number;
8
9         System.out.print "[" + result + " ");
10
11         yield(result);
12     }
13 }
```

Az eljárás egy példán keresztül

A Pluggable Annotation Processing API-t használja

A Pluggable Annotation Processing API-t használja

A `return` utasítás jelképezi a `yield` utasítást

A Pluggable Annotation Processing API-t használja

A `return` utasítás jelképezi a `yield` utasítást

Continuation Passing Style-alapú megvalósítás

Prímek generálása

```
1 @Generator
2 private Stream<Integer> generatePrimes() {
3     LinkedList<Integer> primes = new LinkedList<>();
4     primes.add(2);
5     return 2;
6
7     int current = 1;
8
9     loop:
10    do {
11        current += 2;
12        for (int i : primes) {
13            if (current % i == 0) {
14                continue loop;
15            }
16        }
17
18        primes.add(current);
19
20        return current;
21    } while (true);
22 }
```

Prímek generálása

```
1 @Generator
2 private Stream<Integer> generatePrimes() {
3     LinkedList<Integer> primes = new LinkedList<>();
4     primes.add(2);
5     return 2;
6
7     int current = 1;
8
9     loop:
10    do {
11        current += 2;
12        for (int i : primes) {
13            if (current % i == 0) {
14                continue loop;
15            }
16        }
17
18        primes.add(current);
19
20        return current;
21    } while (true);
22 }
```

Prímek generálása

```
1 @Generator
2 private Stream<Integer> generatePrimes() {
3     LinkedList<Integer> primes = new LinkedList<>();
4     primes.add(2);
5     return 2;
6
7     int current = 1;
8
9     loop:
10    do {
11        current += 2;
12        for (int i : primes) {
13            if (current % i == 0) {
14                continue loop;
15            }
16        }
17
18        primes.add(current);
19
20        return current;
21    } while (true);
22 }
```

Prímek generálása – Vágási pontok

```
1 @Generator
2 private Stream<Integer> generatePrimes() {
3     LinkedList<Integer> primes = new LinkedList<>();
4     primes.add(2);
5     return 2;
6
7     int current = 1;
8
9     loop:
10    do {
11        current += 2;
12        for (int i : primes) {
13            if (current % i == 0) {
14                continue loop;
15            }
16        }
17
18        primes.add(current);
19
20        return current;
21    } while (true);
22 }
```

Prímek generálása – Vágási pontok

```
1 @Generator
2 private Stream<Integer> generatePrimes() {
3     LinkedList<Integer> primes = new LinkedList<>();
4     primes.add(2);
5     return 2;
6
7     int current = 1;
8
9     loop:
10    do {
11        current += 2;
12        for (int i : primes) {
13            if (current % i == 0) {
14                continue loop;
15            }
16        }
17
18        primes.add(current);
19
20        return current;
21    } while (true);
22 }
```

Prímek generálása – Vágási pontok

```
1 @Generator
2 private Stream<Integer> generatePrimes() {
3     LinkedList<Integer> primes = new LinkedList<>();
4     primes.add(2);
5     return 2;
6
7     int current = 1;
8
9     loop:
10    do {
11        current += 2;
12        for (int i : primes) {
13            if (current % i == 0) {
14                continue loop;
15            }
16        }
17
18        primes.add(current);
19
20        return current;
21    } while (true);
22 }
```

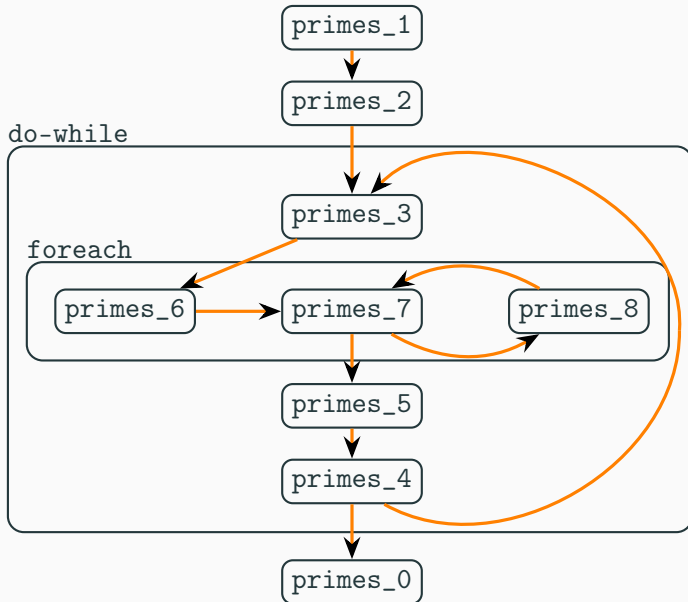
Prímek generálása – Vágási pontok

```
1 @Generator
2 private Stream<Integer> generatePrimes() {
3     LinkedList<Integer> primes = new LinkedList<>();
4     primes.add(2);
5     return 2;
6
7     int current = 1;
8
9     loop:
10    do {
11        current += 2;
12        for (int i : primes) {
13            if (current % i == 0) {
14                continue loop;
15            }
16        }
17
18        primes.add(current);
19
20        return current;
21    } while (true);
22 }
```


Prímek generálása – Vágási pontok

```
1 @Generator
2 private Stream<Integer> generatePrimes() {
3     LinkedList<Integer> primes = new LinkedList<>();
4     primes.add(2);
5     return 2;
6
7     int current = 1;
8
9     loop:
10    do {
11        current += 2;
12        for (int i : primes) {
13            if (current % i == 0) {
14                continue loop;
15            }
16        }
17
18        primes.add(current);
19
20        return current;
21    } while (true);
22 }
```

A vezérlés útja



A transzformáció lépései – 1

```
1 private Bounce<Integer> primes_0(GenState<Integer> k) {  
2     return Bounce.cont(()->k.apply(GenState.empty()));  
3 }
```

Eredeti kód

```
1  LinkedList<Integer> primes = new LinkedList<>();  
2  primes.add(2);  
3  return 2;
```

Transzformált kód

```
1  private Bounce<Integer> primes_1(GenState<Integer> k) {  
2      primes = new LinkedList<>();  
3      primes.add(2);  
4      return Bounce.cont(()->primes_2(k), 2);  
5  }
```

Eredeti kód

```
1  LinkedList<Integer> primes = new LinkedList<>();  
2  primes.add(2);  
3  return 2;
```

Transzformált kód

```
1  private Bounce<Integer> primes_1(GenState<Integer> k) {  
2      primes = new LinkedList<>();  
3      primes.add(2);  
4      return Bounce.cont(()->primes_2(k), 2);  
5  }
```

Eredeti kód

```
1  LinkedList<Integer> primes = new LinkedList<>();  
2  primes.add(2);  
3  return 2;
```

Transzformált kód

```
1  private Bounce<Integer> primes_1(GenState<Integer> k) {  
2      primes = new LinkedList<>();  
3      primes.add(2);  
4      return Bounce.cont(()->primes_2(k), 2);  
5  }
```

Eredeti kód

```
1 loop:
2 do {
3     /*
4     * Törzs
5     */
6 } while (true);
```

Transzformált kód

```
1 private Bounce<Integer> primes_4(GenState<Integer> k) {
2     if (true) {
3         return Bounce.cont(()->primes_3(k));
4     }
5     return Bounce.cont(()->primes_0(k));
6 }
```

Eredeti kód

```
1 loop:
2 do {
3     /*
4      * Törzs
5      */
6 } while (true);
```

Transzformált kód

```
1 private Bounce<Integer> primes_4(GenState<Integer> k) {
2     if (true) {
3         return Bounce.cont(()->primes_3(k));
4     }
5     return Bounce.cont(()->primes_0(k));
6 }
```


Eredeti kód

```
1 loop:
2 do {
3     /*
4     * Törzs
5     */
6 } while (true);
```

Transzformált kód

```
1 private Bounce<Integer> primes_4(GenState<Integer> k) {
2     if (true) {
3         return Bounce.cont(()->primes_3(k));
4     }
5     return Bounce.cont(()->primes_0(k));
6 }
```

Eredeti kód

```
1 loop:
2 do {
3     /*
4     * Törzs
5     */
6 } while (true);
```

Transzformált kód

```
1 private Bounce<Integer> primes_4(GenState<Integer> k) {
2     if (true) {
3         return Bounce.cont(()->primes_3(k));
4     }
5     return Bounce.cont(()->primes_0(k));
6 }
```

A transzformáció lépései – 4

Eredeti kód

```
1  current += 2;
2  for (int i : primes) {
3      if (current % i == 0) {
4          continue loop;
5      }
6  }
7
8  primes.add(current);
9
10 return current;
```

Transzformált kód

```
1  private Bounce<Integer> primes_3(GenState<Integer> k) {
2      current += 2;
3      return Bounce.cont(()->primes_6(k));
4  }
```

A transzformáció lépései – 4

Eredeti kód

```
1  current += 2;
2  for (int i : primes) {
3      if (current % i == 0) {
4          continue loop;
5      }
6  }
7
8  primes.add(current);
9
10 return current;
```

Transzformált kód

```
1  private Bounce<Integer> primes_5(GenState<Integer> k) {
2      primes.add(current);
3      return Bounce.cont(()->primes_4(k), current);
4  }
```

Eredeti kód

```
1  for (int i : primes) {  
2      if (current % i == 0) {  
3          continue loop;  
4      }  
5  }
```

Transzformált kód

```
1  private Bounce<Integer> primes_6(GeState<Integer> k) {  
2      iterator = CPSUtil.iterator(primes);  
3      return Bounce.cont(()->primes_7(k));  
4  }
```

A transzformáció lépései – 5

Eredeti kód

```
1  for (int i : primes) {  
2      if (current % i == 0) {  
3          continue loop;  
4      }  
5  }
```

Transzformált kód

```
1  private Bounce<Integer> primes_7(GenState<Integer> k) {  
2      if (iterator.hasNext()) {  
3          i = iterator.next();  
4          return Bounce.cont(()->primes_8(k));  
5      }  
6      return Bounce.cont(()->primes_5(k));  
7  }
```

A transzformáció lépései – 5

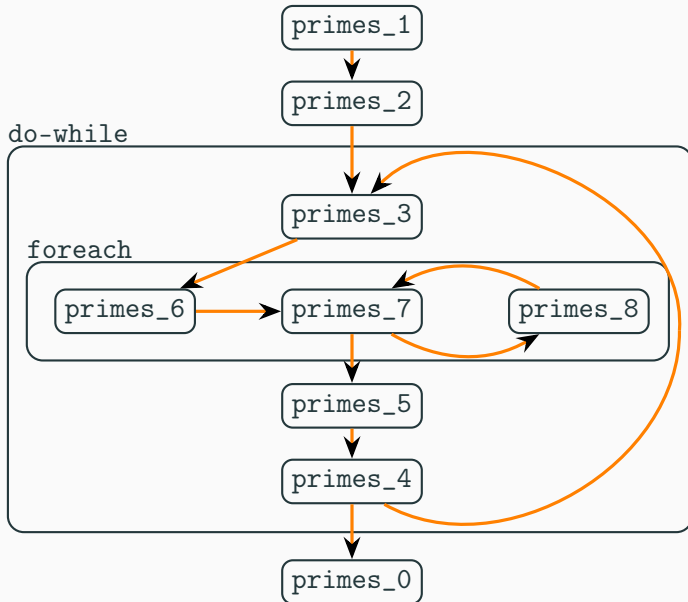
Eredeti kód

```
1  for (int i : primes) {  
2      if (current % i == 0) {  
3          continue loop;  
4      }  
5  }
```

Transzformált kód

```
1  private Bounce<Integer> primes_8(GenState<Integer> k) {  
2      if (current % i == 0) {  
3          return Bounce.cont(()->primes_4(k));  
4      }  
5      return Bounce.cont(()->primes_7(k));  
6  }
```

A vezérlés útja



A generátorok bevezetése *Javában* indokolt, hiszen rendkívül rugalmasan felhasználhatóak.

A bemutatott eljárás mindössze egy annotáció elhelyezését igényli a programozó részéről.

A háttérben a metódusok kisebb darabokra vágása történik meg, melyeket futásidőben egy *trampoline* vezérel.

A vezérlési szerkezeteket elágazások és megfelelő módon szervezett *continuation*ök modellezzik.

Köszönöm a figyelmet!