

Different implementations in Java of a nested loop and their proofs

Jan Dockx

Marko van Dooren

Eric Steegmans

Report CW 324, December 2001



Katholieke Universiteit Leuven
Department of Computer Science

Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Different implementations in Java of a nested loop and their proofs

Jan Dockx

Marko van Dooren

Eric Steegmans

Report CW 324, December 2001

Department of Computer Science, K.U.Leuven

Abstract

In this technical report, we prove the correctness of different implementations of a given specification. This way, we show that using internal iterators dramatically shortens the length of a proof of correctness, and thus the complexity of the code. A remarkable outcome is that introducing a helper method no longer shortens the proof when internal iterators are used, but makes it longer.

Keywords : proof, object-oriented, dijkstra.

CR Subject Classification : D.1.5, D.2.4, D.2.m, D.3.3

Different implementations in Java of a nested loop and their proofs

Jan Dockx

Marko van Dooren

Eric Steegmans

{Jan.Dockx, Marko.vanDooren, Eric.Steegmans}@cs.kuleuven.ac.be

December 2001 (version 1)

February 2002 (version 2)

Abstract

In this technical report, we prove the correctness of different implementations of a given specification. This way, we show that using internal iterators dramatically shortens the length of a proof of correctness, and thus the complexity of the code. A remarkable outcome is that introducing a helper method no longer shortens the proof when internal iterators are used, but makes it longer.

Contents

1	Introduction	3
2	Proofs of correctness	3
2.1	Proof of correctness for an implementation using for-loops. . .	3
2.2	Proof of correctness for an implementation using a helper method	50
2.3	Proof of correctness for a recursive implementation.	78
2.4	Proof of correctness for an implementation using the ForAll and Exists classes.	101
2.5	Proof of correctness for an implementation using the ForAll and Exists classes with a helper method.	109
3	Conclusion	119

1 Introduction

As explained in our paper which was submitted to ECOOP 2002 [2], iterations are the most complex construction in current object oriented software systems. We suggest internal iterators as defined in [3] as a solution for the problem. This technical report contains proofs of correctness for different implementations of a method that needs a nested iteration. As Dijkstra already pointed out in [1], the complexity of a piece of code is comparable to the length of its proof. Therefore, a construction with a much shorter proof should be considerably easier to use than a construction with a long proof.

The following implementations are proven:

1. nested for-loop
2. use of a helper method and *external* iterators
3. recursive implementation using a helper method
4. implementation using internal iterators
5. implementation using internal iterators and a helper method

2 Proofs of correctness

Aside from making iterations much easier to write and to read, the classes in the collections package provide another benefit: they make proving the correctness of code a lot easier. We will show this by proving the correctness of the method `allOneYearOlder()`. The specification of the method is shown in listing 1.

When two `Person` objects are equal, they will have the same `age()`. Also, the age is limited between 0 and 2000. This is illustrated by the class invariant of `Person` as shown in listing 2.

We assume we are not working in a concurrent environment, so the collections aren't changed during the execution of our code by outside sources. Furthermore, our code does not change the collections itself. This means no `ConcurrentModificationExceptions` can be thrown by the iterators.

2.1 Proof of correctness for an implementation using for-loops.

Listing 3 shows an implementation for the `allOneYearOlder()` method using for loops to iterate over the explicit collections. Iterators could also have been used, but using the arrays allows a more formal loop invariant. The implementation is not efficient, but is easier to prove than an efficient implementation.

Listing 1: Specification of the allOneYearOlder() method.

```
/*@
  @ public behavior
  @
  @ pre first != null;
  @ pre second != null;
  @ pre (\forall Object o; first .contains(o); o instanceof Person);
  @ pre (\forall Object o; second.contains(o); o instanceof Person);
  @ pre ! first .contains(null);
  @ pre ! second.contains(null);
  @
  @ post \result == (\forall Person p1; first .contains(p1);
    @ (\exists Person p2; second.contains(p2);
    @ p1.age() == p2.age() + 1));
  @*/
public boolean allOneYearOlder(final Collection first, final Collection
  second);
```

Listing 2: Class invariant of Person concerning the age() method.

```
/*@
  @ public invariant (\forall Person p1; p1 != null;
  @ (\forall Person p2; p2 != null;
  @ o1.equals(o2) ==> o1.age() == o2.age()));
  @ public invariant age() >= 0;
  @ public invariant age() <= 2000;
  @*/
```

Listing 3: Implementation of the allOneYearOlder method using for loops.

```

0 //{P0}
1 Person[] firstPersons = (Person[]) first.toArray(new Person[0]);
2 Person[] secondPersons = (Person[])second.toArray(new Person[0]);
3 boolean accOuter = true;
4 int i=0; //{P1}
5 while(i < firstPersons.length) { //{P2}
6     Person p1 = firstPersons[i];
7     boolean accInner = false;
8     int j=0; //{P3}
9     while(j < secondPersons.length) { //{P4}
10        Person p2 = secondPersons[j]; //{P5}
11        accInner = accInner || ( p1.age() == p2.age() + 1 ); //{P6}
12        j++; //{P7}
13    } //{P8}
14    accOuter = accOuter && accInner; //{P9}
15    i++; //{P10}
16 } //{P11}
17 return accOuter; //{Pn}

```

We assume that the toArray() returns the explicit collection of a collection (which it does, but which is not specified in the Java API). You really don't want to write such proofs for all your iterations.

Program point P₀

At P₀, we know that the preconditions of the method are true.

[0.1] first != null

[0.2] second != null

[0.3] (\forall Object o; first.contains(o); o instanceof Person);

[0.4] (\forall Object o; second.contains(o); o instanceof Person);

[0.5] ! first.contains(null);

[0.6] ! second.contains(null);

2.1.1 Program point P₁

preconditions

We have to prove that the calls made in line 1 and 2 will succeed. There is a possibility for NullPointerExceptions, ArrayStoreExceptions and Class-

CastExceptions. We believe the documentation of the `toArray(Object[] a)` method can be translated into JML as show in listing 4.

Listing 4: Contract of the `toArray(Object[] a)` method.

```

/*@
  @ public behavior
  @
  @ post \result != null;
  @ post \result.getClass() == a.getClass();
  @ post (a.length >= size()) ==> (\result == a);
  @ post (a.length < size()) ==> (\result.length == size());
  @ post (\forall int i; i >= 0 && i < size();
  @       \result[i] == toArray()[i]);
  @
  @ signals (NullPointerException)
  @       a == null;
  @ signals (ArrayStoreException)
  @       (\exists Object o; contains(o);
  @       (o != null) &&
  @       (! a.getClass().getComponentType().isInstance(o)));
  @*/
public Object[] toArray(Object[] a);

```

To prove that no exceptions will be thrown, we have to prove that:

- [1.a] `first != null`
- [1.b] `(! (\exists int i; i >= 0 && i < first.size();
 (first.toArray()[i] != null) &&
 (! new Person[0].getClass().isInstance(first.toArray()[i])));`
- [1.c] `Person[].class.isAssignableFrom(first.toArray(new Person[0]).getClass())` Per-
- [1.d] `second != null`
- [1.e] `(! (\exists int i; i >= 0 && i < second.size();
 (second.toArray()[i] != null) &&
 (! new Person[0].getClass().isInstance(second.toArray()[i])));`
- [1.f] `Person[].class.isAssignableFrom(second.toArray(new Person[0]).getClass())` Per-

Proofs

[1.a] \dashv [1.1]

QED

[1.b] \dashv (\forall Object o; first.contains(o);
(o == null) || (new
Person[0].getClass().getComponentType().isInstance(o)));
 \dashv (\forall Object o; first.contains(o);
(o == null) || (Person.class.isInstance(o)));
 \dashv (\forall Object o; first.contains(o); o instanceof Person);
 \dashv [0.3]

QED

[1.c] \dashv Person[].class.isAssignableFrom(first.toArray(new
Person[0]).getClass())
*substitution according to the second postcondition of the toArray
method*
 \dashv Person[].class.isAssignableFrom(new Person[0].getClass())
 \dashv Person[].class.isAssignableFrom(Person[].class)
 \dashv true

QED

[1.d] \dashv [1.2]

QED

[1.e] \dashv (\forall Object o; second.contains(o);
(o == null) || (new
Person[0].getClass().getComponentType().isInstance(o)));
 \dashv (\forall Object o; second.contains(o);
(o == null) || (Person.class.isInstance(o)));
 \dashv (\forall Object o; second.contains(o); o instanceof Person);
 \dashv [0.4]

QED

- [1.f] \vdash `Person[].class.isAssignableFrom(second.toArray(new Person[0]).getClass())`
substitution according to the second postcondition of the toArray method
- \vdash `Person[].class.isAssignableFrom(new Person[0].getClass())`
- \vdash `Person[].class.isAssignableFrom(Person[].class)`
- \vdash `true`

QED

Assertions

About the state P_1 , the following can be said:

- [1.1] `first != null`
- [1.2] `second != null`
- [1.3] $(\forall$ Object o ; `first.contains(o)`; o instanceof `Person)`
- [1.4] $(\forall$ Object o ; `second.contains(o)`; o instanceof `Person)`
- [1.5] `! first.contains(null)`
- [1.6] `! second.contains(null)`
- [1.7] `firstPersons == first.toArray(new Person[0])`
- [1.8] `firstPersons != null`
- [1.9] $(\forall$ int p ; $p \geq 0 \ \&\& \ p < \text{firstPersons.length}$; `firstPersons[p] != null`)
- [1.10] `secondPersons == second.toArray(new Person[0])`
- [1.11] `secondPersons != null`
- [1.12] $(\forall$ int q ; $q \geq 0 \ \&\& \ q < \text{secondPersons.length}$; `secondPersons[q] != null`)
- [1.13] `accOuter == true`
- [1.14] `i == 0`

Proofs

[1.1] \dashv [0.1]

QED

[1.2] \dashv [0.2]

QED

[1.3] \dashv [0.3]

QED

[1.4] \dashv [0.4]

QED

[1.5] \dashv [0.5]

QED

[1.6] \dashv [0.6]

QED

[1.7] \dashv line 1

QED

[1.8] \dashv line 1 $\&\&$ first postcondition of toArray

QED

[1.9] \dashv (\forall int p; p \geq 0 $\&\&$ p $<$ firstPersons.length; firstPersons[p] \neq null)
 \dashv (\forall int p; p \geq 0 $\&\&$ p $<$ first.toArray(new Person[0]).length; first.toArray(new Person[0])[p] \neq null)
 \dashv (\forall int p; p \geq 0 $\&\&$ p $<$ first.toArray().length; first.toArray()[p] \neq null)
 \dashv ! first.contains(null)
 \dashv [0.5]

QED

[1.10] \dashv line 2

QED

[1.11] \dashv line 2 $\&\&$ first postcondition of toArray

QED

[1.12] \dashv (\forall forall int p; p \geq 0 $\&\&$ p $<$ secondPersons.length; secondPersons[p] \neq null)
 \dashv (\forall forall int p; p \geq 0 $\&\&$ p $<$ second.toArray(new Person[0]).length; second.toArray(new Person[0])[p] \neq null)
 \dashv (\forall forall int p; p \geq 0 $\&\&$ p $<$ second.toArray().length; second.toArray()[p] \neq null)
 \dashv ! second.contains(null)
 \dashv [0.5]

QED

[1.13] \dashv line 3

QED

[1.14] \dashv line 4

QED

2.1.2 Program point P_2

Preconditions

To arrive safely at program point P_2 , the following precondition must be satisfied.

[2.a] firstPersons \neq null

Proof

[2.a] \dashv [1.8]

QED

Assertions

The assertions that are true at program point P_2 are the following. Assertion [LI.1] is the loop invariant which will be proven by induction.

[2.1] `first != null`

[2.2] `second != null`

[2.3] `(\forall \text{Object } o; \text{first.contains}(o); o \text{ instanceof } \text{Person})`

[2.4] `(\forall \text{Object } o; \text{second.contains}(o); o \text{ instanceof } \text{Person})`

[2.5] `! first.contains(null)`

[2.6] `! second.contains(null)`

[2.7] `firstPersons == first.toArray(new Person[0])`

[2.8] `firstPersons != null`

[2.9] `(\forall \text{int } p; p \geq 0 \ \&\& \ p < \text{firstPersons.length}; \text{firstPersons}[p] \neq \text{null})`

[2.10] `secondPersons == second.toArray(new Person[0])`

[2.11] `secondPersons != null`

[2.12] `(\forall \text{int } q; q \geq 0 \ \&\& \ q < \text{secondPersons.length}; \text{secondPersons}[q] \neq \text{null})`

[LI.1] `(i \geq 0) \ \&\& \`
`accOuter == (\forall \text{int } a; a \geq 0 \ \&\& \ a < i;`
`(\exists \text{int } b; b \geq 0 \ \&\& \ b < \text{secondPersons.length};`
`firstPersons[a].age() ==`
`secondPersons[b].age() + 1));`

[2.13] `i < firstPersons.length`

Proofs

[2.1] \dashv [1.1]

QED

[2.2] \dashv [1.2]

QED

$$[2.3] \dashv [1.3]$$

QED

$$[2.4] \dashv [1.4]$$

QED

$$[2.5] \dashv [1.5]$$

QED

$$[2.6] \dashv [1.6]$$

QED

$$[2.7] \dashv [1.7]$$

QED

$$[2.8] \dashv [1.8]$$

QED

$$[2.9] \dashv [1.9]$$

QED

$$[2.10] \dashv [1.10]$$

QED

$$[2.11] \dashv [1.11]$$

QED

$$[2.12] \dashv [1.12]$$

QED

The proof of [LI.1] consists of two parts, as is usual when using induction. The proof of the degenerate case (the initialization) is given here. The proof of the induction step is given at program point P₁₀.

We have to prove that:

[LI.1] \vdash P₁

[LI.1] \vdash P₁₀

The first part of the proof is given below.

```
[LI.1]  $\vdash$  (i >= 0)&&
    accOuter == (\forall int a; a >= 0 && a < i;
                (\exists int b; b >= 0 &&
                 firstPersons[a] ==
                 secondPersons[b].age() + 1));
    using [1.14]
 $\vdash$  (0 >= 0)&&
    accOuter == (\forall int a; a >= 0 && a < 0;
                (\exists int b; b >= 0 &&
                 firstPersons[a].age() ==
                 secondPersons[b].age() + 1));
 $\vdash$  accOuter == true
    using [1.13]
 $\vdash$  true
```

QED

[2.13] \vdash line 5

QED

2.1.3 Program point P₃

Preconditions

The preconditions to be satisfied are:

[3.a] firstPersons != null

[3.b] i >= 0

[3.c] i < firstPersons.length

Proofs

[3.a] \dashv [2.8]

QED

[3.b] \dashv [2.14]

QED

[3.c] \dashv [2.15]

QED

Assertions

The assertions that are true at program point P_3 are:

[3.1] `first != null`

[3.2] `second != null`

[3.3] $(\forall \text{Object } o; \text{first.contains}(o); o \text{ instanceof Person})$

[3.4] $(\forall \text{Object } o; \text{second.contains}(o); o \text{ instanceof Person})$

[3.5] `! first.contains(null)`

[3.6] `! second.contains(null)`

[3.7] `firstPersons == first.toArray(new Person[0])`

[3.8] `firstPersons != null`

[3.9] $(\forall \text{int } p; p \geq 0 \ \&\& \ p < \text{firstPersons.length}; \text{firstPersons}[p] \neq \text{null})$

[3.10] `secondPersons == second.toArray(new Person[0])`

[3.11] `secondPersons != null`

[3.12] $(\forall \text{int } q; q \geq 0 \ \&\& \ q < \text{secondPersons.length}; \text{secondPersons}[q] \neq \text{null})$

[3.13] $(i \geq 0) \ \&\& \ \text{accOuter} == (\forall \text{int } a; a \geq 0 \ \&\& \ a < i; (\exists \text{int } b; b \geq 0 \ \&\& \ b < \text{secondPersons.length}; \text{firstPersons}[a].\text{age}() == \text{secondPersons}[b].\text{age}() + 1));$

[3.14] $i < \text{firstPersons.length}$

[3.15] $p1 \neq \text{null}$

[3.16] $p1 == \text{firstPersons}[i]$

[3.17] $\text{accInner} == \text{false}$

[3.18] $j == 0$

Proofs

[3.1] $\vdash [2.1]$

QED

[3.2] $\vdash [2.2]$

QED

[3.3] $\vdash [2.3]$

QED

[3.4] $\vdash [2.4]$

QED

[3.5] $\vdash [2.5]$

QED

[3.6] $\vdash [2.6]$

QED

[3.7] $\vdash [2.7]$

QED

[3.8] $\vdash [2.8]$

QED

[3.9] ⊢ [2.9]	QED
[3.10] ⊢ [2.10]	QED
[3.11] ⊢ [2.11]	QED
[3.12] ⊢ [2.12]	QED
[3.13] ⊢ [LI.1]	QED
[3.14] ⊢ line 5	QED
[3.15] ⊢ p1 != null ⊢ firstPersons[i] != null ⊢ (∀forall int p; p >= 0 && p < firstPersons.length; firstPersons[p] != null) ⊢ [2.9]	QED
[3.16] ⊢ line 6	QED
[3.17] ⊢ line 7	QED
[3.18] ⊢ line 8	QED

2.1.4 Program point P₄

Preconditions

At program point P₄, we enter the second loop. The precondition is similar to that of P₂.

[4.a] firstPersons != null

Proof

[4.a] \dashv [3.11]

QED

Assertions

The assertions of program point P₄ are listed below. The loop invariant of the inner loop is given in [LI.2].

[4.1] first != null

[4.2] second != null

[4.3] (\forall Object o; first.contains(o); o instanceof Person)

[4.4] (\forall Object o; second.contains(o); o instanceof Person)

[4.5] ! first.contains(null)

[4.6] ! second.contains(null)

[4.7] firstPersons == first.toArray(new Person[0])

[4.8] firstPersons != null

[4.9] (\forall int p; p \geq 0 && p < firstPersons.length; firstPersons[p] != null)

[4.10] secondPersons == second.toArray(new Person[0])

[4.11] secondPersons != null

[4.12] (\forall int q; q \geq 0 && q < secondPersons.length; secondPersons[q] != null)

[4.13] (i \geq 0) &&
accOuter == (\forall int a; a \geq 0 && a < i;
 \exists int b; b \geq 0 && b < secondPersons.length;
firstPersons[a].age() ==
secondPersons[b].age() + 1);

[4.14] $i < \text{firstPersons.length}$

[4.15] $p1 \neq \text{null}$

[4.16] $p1 == \text{firstPersons}[i]$

[LI.2] $(j \geq 0) \&\&$
 $\text{accInner} == (\exists \text{int } b; b \geq 0 \ \&\& \ b < j;$
 $\text{firstPersons}[i].\text{age}() ==$
 $\text{secondPersons}[b].\text{age}() + 1);$

[4.17] $j < \text{secondPersons.length}$

Again, we only prove the initialization. The proof of the induction step will be done again at P_7 to ensure that loop invariant [LI.2] is always satisfied.

Proofs

[4.1] \dashv [3.1]

QED

[4.2] \dashv [3.2]

QED

[4.3] \dashv [3.3]

QED

[4.4] \dashv [3.4]

QED

[4.5] \dashv [3.5]

QED

[4.6] \dashv [3.6]

QED

[4.7] \dashv [3.7]

QED

$$[4.8] \dashv [3.8]$$

QED

$$[4.9] \dashv [3.9]$$

QED

$$[4.10] \dashv [3.10]$$

QED

$$[4.11] \dashv [3.11]$$

QED

$$[4.12] \dashv [3.12]$$

QED

$$[4.13] \dashv [3.13]$$

QED

$$[4.14] \dashv [3.14]$$

QED

$$[4.15] \dashv [3.15]$$

QED

$$[4.16] \dashv [3.16]$$

QED

To prove [LI.2], we have to prove that.

$$[LI.2] \dashv P_3$$

$$[LI.2] \dashv P_7$$

The first part of the proof is given below.

```

[LI.2] ⊢ (j>=0)&&
      accInner == (j>=0)&&
                (\exists int b; b >= 0 && b < j;
                firstPersons[i].age() ==
                secondPersons[b].age() + 1);

      using [3.18]
⊢ (0>=0)&&
  accInner == (0>=0)&&
              (\exists int b; b >= 0 && b < 0;
              firstPersons[i].age() ==
              secondPersons[b].age() + 1);

      using [3.17]
⊢ accInner == false
⊢ true

```

QED

[4.17] ⊢ [line 9]

QED

2.1.5 Program point P_5

Preconditions

The preconditions for program point P_5 are similar to those of P_3 . They ensure that the access

[5.a] `secondPersons != null`

[5.b] `j >= 0`

[5.c] `j < secondPersons.length`

Proof

[5.a] ⊢ [4.8]

QED

[5.b] ⊢ [LI.2]

QED

[5.c] ⊢ [4.17]

QED

Assertions

The assertions at P_5 are:

- [5.1] `first != null`
- [5.2] `second != null`
- [5.3] `(\forall \text{Object } o; \text{first.contains}(o); o \text{ instanceof } \text{Person})`
- [5.4] `(\forall \text{Object } o; \text{second.contains}(o); o \text{ instanceof } \text{Person})`
- [5.5] `! first.contains(null)`
- [5.6] `! second.contains(null)`
- [5.7] `firstPersons == first.toArray(new Person[0])`
- [5.8] `firstPersons != null`
- [5.9] `(\forall \text{int } p; p >= 0 \ \&\& \ p < \text{firstPersons.length}; \text{firstPersons}[p] != \text{null})`
- [5.10] `secondPersons == second.toArray(new Person[0])`
- [5.11] `secondPersons != null`
- [5.12] `(\forall \text{int } q; q >= 0 \ \&\& \ q < \text{secondPersons.length}; \text{secondPersons}[q] != \text{null})`
- [5.13] `(i >= 0)\&\& \ \text{accOuter} == (\forall \text{int } a; a >= 0 \ \&\& \ a < i; \ (\exists \text{int } b; b >= 0 \ \&\& \ b < \text{secondPersons.length}; \ \text{firstPersons}[a].\text{age}() == \text{secondPersons}[b].\text{age}() + 1));`
- [5.14] `i < firstPersons.length`
- [5.15] `p1 != null`
- [5.16] `p1 == firstPersons[i]`
- [5.17] `(j >= 0)\&\& \ \text{accInner} == (\exists \text{int } b; b >= 0 \ \&\& \ b < j; \ \text{firstPersons}[i].\text{age}() == \text{secondPersons}[b].\text{age}() + 1);`
- [5.18] `j < secondPersons.length`
- [5.19] `p2 != null`
- [5.20] `p2 == secondPersons[j]`

Proofs

$$[5.1] \dashv [4.1]$$

QED

$$[5.2] \dashv [4.2]$$

QED

$$[5.3] \dashv [4.3]$$

QED

$$[5.4] \dashv [4.4]$$

QED

$$[5.5] \dashv [4.5]$$

QED

$$[5.6] \dashv [4.6]$$

QED

$$[5.7] \dashv [4.7]$$

QED

$$[5.8] \dashv [4.8]$$

QED

$$[5.9] \dashv [4.9]$$

QED

$$[5.10] \dashv [4.10]$$

QED

[5.11] \vdash [4.11]

QED

[5.12] \vdash [4.12]

QED

[5.13] \vdash [4.13]

QED

[5.14] \vdash [4.14]

QED

[5.15] \vdash [4.15]

QED

[5.16] \vdash [4.16]

QED

[5.17] \vdash [LI.2]

QED

[5.18] \vdash [4.17]

QED

[5.19] \vdash p2 != null
 \vdash secondPersons[j] != null
 \vdash (\forall int q; q \geq 0 && q < secondPersons.length; secondPersons[p] != null)
 \vdash [4.12]

QED

[5.20] \vdash line 10

QED

2.1.6 Program point P_6

Probleem: JML gebruikt java +, en die kan overflowen zodat een verkeerd resultaat bekomen wordt. Verhelpt `age() != 0` dit ?

Preconditions

The preconditions to reach P_6 are:

[6.a] `p1 != null`

[6.b] `p2 != null`

[6.c] `p2.age() < Integer.MAX_VALUE`

Proof

[6.a] \vdash [5.15]

QED

[6.b] \vdash [5.19]

QED

[6.c] *Third invariant of Person.*

\vdash `p2.age() <= 2000`

\vdash `p2.age() < Integer.MAX_VALUE`

\vdash `true`

QED

Assertions

The assertions at P_6 are:

[6.1] `first != null`

[6.2] `second != null`

[6.3] $(\forall \text{Object } o; \text{first.contains}(o); o \text{ instanceof Person})$

[6.4] $(\forall \text{Object } o; \text{second.contains}(o); o \text{ instanceof Person})$

[6.5] $! \text{first.contains}(\text{null})$

[6.6] $! \text{second.contains}(\text{null})$

- [6.7] `firstPersons == first.toArray(new Person[0])`
- [6.8] `firstPersons != null`
- [6.9] `(\forall int p; p >= 0 && p < firstPersons.length; firstPersons[p] != null)`
- [6.10] `secondPersons == second.toArray(new Person[0])`
- [6.11] `secondPersons != null`
- [6.12] `(\forall int q; q >= 0 && q < secondPersons.length; secondPersons[q] != null)`
- [6.13] `(i >= 0)&&`
`accOuter == (\forall int a; a >= 0 && a < i;`
`(\exists int b; b >= 0 && b < secondPersons.length;`
`firstPersons[a].age() ==`
`secondPersons[b].age() + 1));`
- [6.14] `i < firstPersons.length`
- [6.15] `p1 != null`
- [6.16] `p1 == firstPersons[i]`
- [6.17] `(j >= 0)&&`
`accInner == (\exists int b; b >= 0 && b < j + 1;`
`firstPersons[i].age() ==`
`secondPersons[b].age() + 1);`
- [6.18] `j < secondPersons.length`
- [6.19] `p2 != null`
- [6.20] `p2 == secondPersons[j]`

Proofs

[6.1] \dashv [5.1]

QED

[6.2] \dashv [5.2]

QED

[6.3] \dashv [5.3]

QED

[6.4] \dashv [5.4]

QED

[6.5] \dashv [5.5]

QED

[6.6] \dashv [5.6]

QED

[6.7] \dashv [5.7]

QED

[6.8] \dashv [5.8]

QED

[6.9] \dashv [5.9]

QED

[6.10] \dashv [5.10]

QED

[6.11] \dashv [5.11]

QED

[6.12] \dashv [5.12]

QED

[6.13] \dashv [5.13]

QED

[6.14] \vdash [5.14]

QED

[6.15] \vdash [5.15]

QED

[6.16] \vdash [5.16]

QED

We must now prove that the case “j” can be added to the forall predicate of the loop invariant (resulting in the “ $< j + 1$ ”). The proof consists of two steps: proving that this is true in case the comparison of both ages returns true and in case it returns false.

A) $p1.age() == p2.age() + 1$

```
accInner == \old(accInner) || (p1.age() == p2.age() + 1)  $\vdash$  true
      accInner == accInner@P5 || true  $\vdash$  true
      accInner == true  $\vdash$  true
```

```
[6.17]  $\vdash$  (j >= 0)&&
      accInner == (\exists int b; b >= 0 && b < j + 1;
      firstPersons[i].age() ==
      secondPersons[b].age() + 1);
      using [5.17]
 $\vdash$  accInner == (\exists int b; b >= 0 && b < j + 1;
      firstPersons[i].age() ==
      secondPersons[b].age() + 1);
      Expression is true if it is true for j
 $\vdash$  accInner == firstPersons[i].age() ==
      secondPersons[j].age() + 1);
      using [5.16] and [5.20]
 $\vdash$  accInner == p1.age() ==
      p2.age() + 1);
 $\vdash$  accInner == true
 $\vdash$  true
```

QED

B) $p1.age() \neq p2.age() + 1$

$accInner == \text{old}(accInner) \parallel (p1.age() == p2.age() + 1) \dashv \text{true}$
 $accInner == accInner@P_5 \parallel \text{false} \dashv \text{true}$
 $accInner == accInner@P_5 \dashv \text{true}$

$p1.age() \neq p2.age() + 1 \dashv \text{true}$
 $firstPersons[i].age() \neq secondPersons[j].age() + 1 \dashv \text{true}$

[6.17] $\dashv (j \geq 0) \&\&$
 $accInner == (\exists \text{ int } b; b \geq 0 \&\& b < j + 1;$
 $firstPersons[i].age() ==$
 $secondPersons[b].age() + 1);$
using [5.17]
 $\dashv accInner == (\exists \text{ int } b; b \geq 0 \&\& b < j + 1;$
 $firstPersons[i].age() ==$
 $secondPersons[b].age() + 1);$
The case for $j+1$ is false, so it can be remove from the range.
 $\dashv accInner == (\exists \text{ int } b; b \geq 0 \&\& b < j;$
 $firstPersons[i].age() ==$
 $secondPersons[b].age() + 1);$
 $\dashv accInner@P_5 == (\exists \text{ int } b; b \geq 0 \&\& b < j;$
 $firstPersons[i].age() ==$
 $secondPersons[b].age() + 1);$
using [5.17]
 $\dashv \text{true}$

QED

[6.18] \dashv [5.18]

QED

[6.19] \dashv [5.19]

QED

[6.20] \dashv [5.20]

QED

2.1.7 Program point P₇

Preconditions

The only precondition for P₇ is that `j++` does not cause an overflow. Java won't signal an error if it does, but it's clear that it should not happen.

[7.a] `j ≤ Integer.MAX_VALUE`

Proof

[7.a] $\vdash j \leq \text{Integer.MAX_VALUE}$

The maximum length of an array is Integer.MAX_VALUE

$\vdash j \leq \text{secondPersons.length}$

$\vdash [6.18]$

QED

Assertions

The assertions in program point P₇ are:

[7.1] `first != null`

[7.2] `second != null`

[7.3] $(\forall \text{Object } o; \text{first.contains}(o); o \text{ instanceof Person})$

[7.4] $(\forall \text{Object } o; \text{second.contains}(o); o \text{ instanceof Person})$

[7.5] `! first.contains(null)`

[7.6] `! second.contains(null)`

[7.7] `firstPersons == first.toArray(new Person[0])`

[7.8] `firstPersons != null`

[7.9] $(\forall \text{int } p; p \geq 0 \ \&\& \ p < \text{firstPersons.length}; \text{firstPersons}[p] \neq \text{null})$

[7.10] `secondPersons == second.toArray(new Person[0])`

[7.11] `secondPersons != null`

[7.12] $(\forall \text{int } q; q \geq 0 \ \&\& \ q < \text{secondPersons.length}; \text{secondPersons}[q] \neq \text{null})$

[7.13] $(i \geq 0) \wedge \wedge$
accOuter == (\forall forall int a; $a \geq 0 \wedge \wedge a < i$;
(\exists exists int b; $b \geq 0 \wedge \wedge b < \text{secondPersons.length}$;
firstPersons[a].age() ==
secondPersons[b].age() + 1));

[7.14] $i < \text{firstPersons.length}$

[7.15] $p1 \neq \text{null}$

[7.16] $p1 == \text{firstPersons}[i]$

[LI.2] $(j \geq 0) \wedge \wedge$
accInner == (\exists exists int b; $b \geq 0 \wedge \wedge b < j$;
firstPersons[i].age() ==
secondPersons[b].age() + 1);

[7.17] $j - 1 < \text{secondPersons.length}$

[7.18] $p2 \neq \text{null}$

[7.19] $p2 == \text{secondPersons}[j - 1]$

Proofs

[7.1] \dashv [6.1]

QED

[7.2] \dashv [6.2]

QED

[7.3] \dashv [6.3]

QED

[7.4] \dashv [6.4]

QED

[7.5] \dashv [6.5]

QED

[7.6] \dashv [6.6]

QED

[7.7] \dashv [6.7]

QED

[7.8] \dashv [6.8]

QED

[7.9] \dashv [6.9]

QED

[7.10] \dashv [6.10]

QED

[7.11] \dashv [6.11]

QED

[7.12] \dashv [6.12]

QED

[7.13] \dashv [6.13]

QED

[7.14] \dashv [6.14]

QED

[7.15] \dashv [6.15]

QED

[7.16] \dashv [6.16]

QED

We now prove that loop invariant [LI.2] is satisfied at the end of the loop. This proves the induction step for the inner loop.

```
[LI.2]  ⊢ (j >= 0)&&
        accInner == (\exists int b; b >= 0 && b < j;
                    firstPersons[i].age() ==
                    secondPersons[b].age() + 1);
        j == j@P6 + 1
⊢ (j >= 0)&&
  accInner == (\exists int b; b >= 0 && b < j@P6 + 1;
              firstPersons[i].age() ==
              secondPersons[b].age() + 1);
⊢ [6.17]
```

QED

```
[7.17] ⊢ j - 1 < secondPersons.length
        j == j@P6 + 1
⊢ j@P6 < secondPersons.length
⊢ [6.18]
```

QED

```
[7.18] ⊢ [6.19]
```

QED

```
[7.19] ⊢ p2 == secondPersons[j - 1]
        j == j@P6 + 1
⊢ p2 == secondPersons[j@P6]
⊢ [6.20]
```

QED

2.1.8 Program point P₈

Preconditions

The precondition is the same as for P₄.

```
[8.a] secondPersons != null
```

Proof

[8.a] \dashv 7.11

QED

Assertions

Assertion [8.18] states that we have left the loop. Together with [8.17] it will fix the value of j , which will allow us to prove [LI.1] later on.

[8.1] `first != null`

[8.2] `second != null`

[8.3] `(\forall \text{Object } o; \text{first.contains}(o); o \text{ instanceof } \text{Person})`

[8.4] `(\forall \text{Object } o; \text{second.contains}(o); o \text{ instanceof } \text{Person})`

[8.5] `! first.contains(null)`

[8.6] `! second.contains(null)`

[8.7] `firstPersons == first.toArray(new Person[0])`

[8.8] `firstPersons != null`

[8.9] `(\forall \text{int } p; p \ge 0 \ \&\& \ p < \text{firstPersons.length}; \text{firstPersons}[p] != \text{null})`

[8.10] `secondPersons == second.toArray(new Person[0])`

[8.11] `secondPersons != null`

[8.12] `(\forall \text{int } q; q \ge 0 \ \&\& \ q < \text{secondPersons.length}; \text{secondPersons}[q] != \text{null})`

[8.13] `(i \ge 0) \ \&\& \`
`accOuter == (\forall \text{int } a; a \ge 0 \ \&\& \ a < i;`
`(\exists \text{int } b; b \ge 0 \ \&\& \ b < \text{secondPersons.length};`
`firstPersons[a].age() ==`
`secondPersons[b].age() + 1));`

[8.14] `i < firstPersons.length`

[8.15] `p1 != null`

[8.16] `p1 == firstPersons[i]`

```
[LI.2] (j >= 0)&&  
      accInner == (\exists int b; b >= 0 && b < j;  
                  firstPersons[i].age() ==  
                  secondPersons[b].age() + 1);
```

[8.17] $j - 1 < \text{secondPersons.length}$

[8.18] $j \geq \text{secondPersons.length}$

[8.19] $p2 \neq \text{null}$

[8.20] $p2 == \text{secondPersons}[j - 1]$

Proofs

[8.1] \dashv [7.1]

QED

[8.2] \dashv [7.2]

QED

[8.3] \dashv [7.3]

QED

[8.4] \dashv [7.4]

QED

[8.5] \dashv [7.5]

QED

[8.6] \dashv [7.6]

QED

[8.7] \dashv [7.7]

QED

$$[8.8] \quad \dashv \quad [7.8]$$

QED

$$[8.9] \quad \dashv \quad [7.9]$$

QED

$$[8.10] \quad \dashv \quad [7.10]$$

QED

$$[8.11] \quad \dashv \quad [7.11]$$

QED

$$[8.12] \quad \dashv \quad [7.12]$$

QED

$$[8.13] \quad \dashv \quad [7.13]$$

QED

$$[8.14] \quad \dashv \quad [7.14]$$

QED

$$[8.15] \quad \dashv \quad [7.15]$$

QED

$$[8.16] \quad \dashv \quad [7.16]$$

QED

[LI.2] \dashv [LI.2]

QED

[8.17] \dashv [7.17]

QED

[8.18] \dashv line 9

QED

[8.19] \dashv [7.18]

QED

[8.20] \dashv [7.19]

QED

2.1.9 Program point P_9

Preconditions

There are no preconditions for P_9 .

Assertions

The assertions for program point P_9 are:

[9.1] `first != null`

[9.2] `second != null`

[9.3] `(\forall \text{Object } o; \text{first.contains}(o); o \text{ instanceof } \text{Person})`

[9.4] `(\forall \text{Object } o; \text{second.contains}(o); o \text{ instanceof } \text{Person})`

[9.5] `! first.contains(null)`

[9.6] `! second.contains(null)`

[9.7] `firstPersons == first.toArray(new Person[0])`

[9.8] `firstPersons != null`

[9.9] (\forall int p; p \geq 0 $\&\&$ p $<$ firstPersons.length; firstPersons[p] \neq null)

[9.10] secondPersons == second.toArray(new Person[0])

[9.11] secondPersons \neq null

[9.12] (\forall int q; q \geq 0 $\&\&$ q $<$ secondPersons.length; secondPersons[q] \neq null)

[9.13] (i \geq 0) $\&\&$
accOuter == (\forall int a; a \geq 0 $\&\&$ a $<$ i + 1;
(\exists int b; b \geq 0 $\&\&$ b $<$ secondPersons.length;
firstPersons[a].age() ==
secondPersons[b].age() + 1));

[9.14] i $<$ firstPersons.length

[9.15] p1 \neq null

[9.16] p1 == firstPersons[i]

[9.17] (j \geq 0) $\&\&$
accInner == (\exists int b; b \geq 0 $\&\&$ b $<$ secondPersons.length;
firstPersons[i].age() ==
secondPersons[b].age() + 1);

[9.18] j == secondPersons.length

[9.19] p2 \neq null

[9.20] p2 == secondPersons[j - 1]

Proofs

[9.1] \dashv [8.1]

QED

[9.2] \dashv [8.2]

QED

[9.3] \dashv [8.3]

QED

[9.4] \dashv [8.4]

QED

[9.5] \dashv [8.5]

QED

[9.6] \dashv [8.6]

QED

[9.7] \dashv [8.7]

QED

[9.8] \dashv [8.8]

QED

[9.9] \dashv [8.9]

QED

[9.10] \dashv [8.10]

QED

[9.11] \dashv [8.11]

QED

[9.12] \dashv [8.12]

QED

To prove [9.13], we must consider two cases again: `accInner` is true, or `accInner` is false.

A) `accInner == true`

`accOuter == accOuter@P8 && accInner` \dashv true

`accOuter == accOuter@P8 && true` \dashv true

`accOuter == accOuter@P8` \dashv true

[9.13] $\vdash (i \geq 0) \&\&$
 $\text{accOuter} == (\forall \text{int } a; a \geq 0 \ \&\& \ a < i + 1;$
 $(\exists \text{int } b; b \geq 0 \ \&\& \ b < \text{secondPer}$
 $\text{sons.length};$
 $\text{firstPersons}[a].\text{age}() ==$
 $\text{secondPersons}[b] + 1));$
using [8.13]
 $\vdash \text{accOuter} == (\forall \text{int } a; a \geq 0 \ \&\& \ a < i + 1;$
 $(\exists \text{int } b; b \geq 0 \ \&\& \ b < \text{secondPer}$
 $\text{sons.length};$
 $\text{firstPersons}[a].\text{age}() ==$
 $\text{secondPersons}[b].\text{age}() + 1));$
using [LI.2] and $\text{accInner} == \text{true}$, we can see that the \exists predicate
results in true for $a=i$, so that value can be removed from the range.
 $\vdash \text{accOuter} == (\forall \text{int } a; a \geq 0 \ \&\& \ a < i;$
 $(\exists \text{int } b; b \geq 0 \ \&\& \ b < \text{secondPer}$
 $\text{sons.length};$
 $\text{firstPersons}[a].\text{age}() ==$
 $\text{secondPersons}[b].\text{age}() + 1));$
 $\text{accOuter} == \text{accOuter}@P_8$
 $\vdash \text{accOuter}@P_8 == (\forall \text{int } a; a \geq 0 \ \&\& \ a < i;$
 $(\exists \text{int } b; b \geq 0 \ \&\& \ b < \text{secondPer}$
 $\text{sons.length};$
 $\text{firstPersons}[a].\text{age}() ==$
 $\text{secondPersons}[b].\text{age}() + 1));$
using [8.13] again
 $\vdash \text{true}$

QED

B) $\text{accInner} == \text{false}$

$\text{accOuter} == \text{accOuter}@P_8 \ \&\& \ \text{accInner} \ \vdash \ \text{true}$
 $\text{accOuter} == \text{accOuter}@P_8 \ \&\& \ \text{false} \ \vdash \ \text{true}$
 $\text{accOuter} == \text{false} \ \vdash \ \text{true}$

[9.13] $\vdash (i \geq 0) \&\&$
 $\text{accOuter} == (\forall \text{int } a; a \geq 0 \ \&\& \ a < i + 1;$
 $(\exists \text{int } b; b \geq 0 \ \&\& \ b < \text{secondPer}$
 $\text{sons.length};$
 $\text{firstPersons}[a].\text{age}() ==$
 $\text{secondPersons}[b].\text{age}() + 1));$

using [8.13]

```
⊢ accOuter == (∀ int a; a ≥ 0 && a < i + 1;
               (∃ int b; b ≥ 0 && b < secondPer-
                sons.length;
                firstPersons[a].age() ==
                secondPersons[b].age() + 1));
```

using [LI.2] and accInner == false, we can see that the ∃ predicate results in false for a=i, so the entire ∀ predicate is false.

```
⊢ accOuter == false
⊢ true
```

QED

[9.14] ⊢ [8.14]

QED

[9.15] ⊢ [8.15]

QED

[9.16] ⊢ [8.16]

QED

```
[9.17] ⊢ (j ≥ 0) &&
accInner == (∃ int b; b ≥ 0 && b < secondPer-
             sons.length;
             firstPersons[i].age() ==
             secondPersons[b].age() + 1);
```

using [9.18]

```
⊢ (j ≥ 0) &&
accInner == (∃ int b; b ≥ 0 && b < j;
             firstPersons[i].age() ==
             secondPersons[b].age() + 1);
```

```
⊢ [LI.2]
```

QED

[9.18] $\vdash j == \text{secondPersons.length}$
 $\vdash j \geq \text{secondPersons.length} \ \&\&$
 $\quad j - 1 < \text{secondPersons.length}$
 $\vdash [8.17] \ \&\& [8.18]$

QED

[9.19] $\vdash [8.19]$

QED

[9.20] $\vdash [8.20]$

QED

2.1.10 Program point P₁₀

Preconditions

As for program point P₇ the only precondition says the counter may not overflow.

[10.a] $i < \text{Integer.MAX_VALUE}$

Proof

[10.a] $\vdash i < \text{Integer.MAX_VALUE}$

The maximum length of an array is Integer.MAX_VALUE

$\vdash i < \text{firstPersons.length}$

$\vdash [9.14]$

QED

Assertions

[10.1] $\text{first} \neq \text{null}$

[10.2] $\text{second} \neq \text{null}$

[10.3] $(\forall \text{Object } o; \text{first.contains}(o); o \text{ instanceof Person})$

[10.4] $(\forall \text{Object } o; \text{second.contains}(o); o \text{ instanceof Person})$

[10.5] ! first.contains(null)

[10.6] ! second.contains(null)

[10.7] firstPersons == first.toArray(new Person[0])

[10.8] firstPersons != null

[10.9] (\forall int p; p >= 0 && p < firstPersons.length; firstPersons[p] != null)

[10.10] secondPersons == second.toArray(new Person[0])

[10.11] secondPersons != null

[10.12] (\forall int q; q >= 0 && q < secondPersons.length; secondPersons[q] != null)

[LI.1] (i >= 0)&&
accOuter == (\forall int a; a >= 0 && a < i;
(\exists int b; b >= 0 && b < secondPersons.length;
firstPersons[a].age() ==
secondPersons[b].age() + 1));

[10.13] i - 1 < firstPersons.length

[10.14] p1 != null

[10.15] p1 == firstPersons[i - 1]

[10.16] (j >= 0)&&
accInner == (\exists int b; b >= 0 && b < secondPersons.length;
firstPersons[i].age() ==
secondPersons[b].age() + 1);

[10.17] j == secondPersons.length

[10.18] p2 != null

[10.19] p2 == secondPersons[j - 1]

Proofs

[10.1] \dashv [9.1]

QED

[10.2] \dashv [9.2]

QED

[10.3] \dashv [9.3]

QED

[10.4] \dashv [9.4]

QED

[10.5] \dashv [9.5]

QED

[10.6] \dashv [9.6]

QED

[10.7] \dashv [9.7]

QED

[10.8] \dashv [9.8]

QED

[10.9] \dashv [9.9]

QED

[10.10] \dashv [9.10]

QED

[10.11] \dashv [9.11]

QED

[10.12] \dashv [9.12]

QED

[LI.1] \vdash $(i \geq 0) \&\&$
 $\text{accOuter} == (\forall \text{int } a; a \geq 0 \ \&\& \ a < i;$
 $\quad (\exists \text{int } b; b \geq 0 \ \&\& \ b < \text{secondPersons.length};$
 $\quad \text{firstPersons}[a].\text{age}() ==$
 $\quad \text{secondPersons}[b].\text{age}() + 1));$
according to line 15: $i == i@P_9 + 1$.
accOuter hasn't changed, so $\text{accOuter} == \text{accOuter}@P_9$.
 \vdash $(i \geq 0) \&\&$
 $\text{accOuter}@P_9 == (\forall \text{int } a; a \geq 0 \ \&\& \ a < i@P_9 + 1;$
 $\quad (\exists \text{int } b; b \geq 0 \ \&\& \ b < \text{secondPersons.length};$
 $\quad \text{firstPersons}[a].\text{age}() ==$
 $\quad \text{secondPersons}[b].\text{age}() + 1));$
using [9.13]
 \vdash true

QED

[10.13] \vdash $i - 1 < \text{firstPersons.length}$
according to line 15: $i == i@P_9 + 1$.
 \vdash $i@P_9 < \text{firstPersons.length}$
 \vdash [9.14]

QED

[10.14] \vdash [9.15]

QED

[10.15] \vdash $p1 == \text{firstPersons}[i - 1]$
according to line 15: $i == i@P_9 + 1$.
 \vdash $p1 == \text{firstPersons}[i@P_9]$
 \vdash [9.16]

QED

[10.16] \rightarrow [9.17]

QED

[10.17] \rightarrow [9.18]

QED

[10.18] \rightarrow [9.19]

QED

[10.19] \rightarrow [9.20]

QED

2.1.11 Program point P₁₁

Preconditions

As for program point P₂, there is only one precondition:

[11.a] firstPersons != null

Proof

[11.a] \rightarrow [10.8]

QED

Assertions

The assertions for program point P₁₁ are:

[11.1] first != null

[11.2] second != null

[11.3] (\forall Object o; first.contains(o); o instanceof Person)

[11.4] (\forall Object o; second.contains(o); o instanceof Person)

[11.5] ! first.contains(null)

[11.6] ! second.contains(null)

[11.7] `firstPersons == first.toArray(new Person[0])`

[11.8] `firstPersons != null`

[11.9] `(\forall int p; p >= 0 && p < firstPersons.length; firstPersons[p] != null)`

[11.10] `secondPersons == second.toArray(new Person[0])`

[11.11] `secondPersons != null`

[11.12] `(\forall int q; q >= 0 && q < secondPersons.length; secondPersons[q] != null)`

[LI.1] `(i >= 0)&&`
`accOuter == (\forall int a; a >= 0 && a < i;`
`(\exists int b; b >= 0 && b < secondPersons.length;`
`firstPersons[a].age() ==`
`secondPersons[b].age() + 1));`

[11.13] `i - 1 < firstPersons.length`

[11.14] `i >= firstPersons.length`

[11.15] `p1 != null`

[11.16] `p1 == firstPersons[i - 1]`

[11.17] `(j >= 0)&&`
`accInner == (\exists int b; b >= 0 && b < secondPersons.length;`
`firstPersons[i].age() ==`
`secondPersons[b].age() + 1);`

[11.18] `j == secondPersons.length`

[11.19] `p2 != null`

[11.20] `p2 == secondPersons[j - 1]`

Proofs

[11.1] \dashv [10.1]

QED

[11.2] \dashv [10.2]

QED

$$[11.3] \dashv [10.3]$$

QED

$$[11.4] \dashv [10.4]$$

QED

$$[11.5] \dashv [10.5]$$

QED

$$[11.6] \dashv [10.6]$$

QED

$$[11.7] \dashv [10.7]$$

QED

$$[11.8] \dashv [10.8]$$

QED

$$[11.9] \dashv [10.9]$$

QED

$$[11.10] \dashv [10.10]$$

QED

$$[11.11] \dashv [10.11]$$

QED

$$[11.12] \dashv [10.12]$$

QED

[LI.1] \dashv [LI.1]

QED

[11.13] \dashv [10.13]

QED

[11.14] \dashv line 5

QED

[11.15] \dashv [10.14]

QED

[11.16] \dashv [10.15]

QED

[11.17] \dashv [10.16]

QED

[11.18] \dashv [10.17]

QED

[11.19] \dashv [10.18]

QED

[11.20] \dashv [10.19]

QED

2.1.12 Program point P_n

Preconditions

Program point P_n has no preconditions.

Assertions

There is only one assertion needed in P_n : the return value must satisfy to postconditions of the method.

```
[n.1] \result == (\forall Person p1; first.contains(p1); (\exists
      Person p2; second.contains(p2); p1.age() ==
      p2.age() + 1));
```

Proof

```
i == firstPersons.length  + i >= firstPersons.length &&
                          i - 1 < firstPersons.length
                          + [11.13] && [11.14]
                          + true
```

```
[n.1] + \result == (\forall Person p1; first.contains(p1);
      (\exists Person p2; second.contains(p2);
      p1.age() == p2.age() + 1));
```

Knowing that age() returns the same for all equal persons (class invariant 1 of Person), we can use the explicit collections of first and second.

```
+ \result == (\forall int a; a >= 0 && a <
      first.toArray().length);
      (\exists int b; b >= 0 && b <
      second.toArray().length;
      ((Person)first.toArray()[a]).age() ==
      ((Person)second.toArray()[b]).age() + 1));
```

$\text{\result} == \text{accOuter} == \text{accOuter}@P_{11}$

```
+ accOuter@P11 == (\forall int a; a >= 0 && a <
      first.toArray().length);
      (\exists int b; b >= 0 && b <
      second.toArray().length;
      ((Person)first.toArray()[a]).age() == ==
      ((Person)second.toArray()[b]).age() + 1));
```

using postconditions of toArray(Array a) and the preconditions of allOneYearOlder.

```
+ accOuter@P11 == (\forall int a; a >= 0 && a <
      first.toArray(new Person[0]).length);
      (\exists int b; b >= 0 && b <
      second.toArray(new Person[0]).length;
      ((Person)first.toArray(new Person[0])[a]) ==
      ((Person)second.toArray(new
      Person[0])[b]).age() + 1));
```

```

    using [11.7] and [11.11].
  † accOuter@P11 == (\forall int a; a >= 0 && a <
                      firstPersons.length);
                      (\exists int b; b >= 0 && b <
                      secondPersons.length;
                      firstPersons[a].age() ==
                      secondPersons[b].age() + 1));
    i == firstPersons.length
  † accOuter@P11 == (\forall int a; a >= 0 && a < i);
                      (\exists int b; b >= 0 && b <
                      secondPersons.length;
                      firstPersons[a].age() ==
                      secondPersons[b].age() + 1));
  † [LI.1]
  † true

```

QED

Quod Erat Demonstrandum

2.2 Proof of correctness for an implementation using a helper method

Listing 5 shows an implementation using a helper method. The specification and implementation of the helper method are shown in listing 6. Again, no efficient implementation is used in order to make the code a bit easier to prove.

Listing 5: Implementation of the allOneYearOlder method using a helper method and iterators.

```

1  public boolean allOneYearOlder(
2      final Collection first , final Collection second) { //P0
3      Iterator iter = first.iterator() ;
4      boolean acc = true; //P1
5      while(iter.hasNext()) { //P2
6          Person person = (Person)iter.next();
7          acc = acc && oneYearOlderThanSome(person, second); //P3
8      } //P4
9      return acc; //Pn
10 }

```

Listing 6: Implementation of the oneYearOlderThanSome method.

```
1  /*@
2   @ public behavior
3   @
4   @ pre person != null;
5   @ pre collection != null;
6   @ pre (\forall Object o; collection .contains(o);
7   @     o instanceof Person);
8   @ pre ! collection .contains(null);
9   @
10  @ post \result == (\exists Person p2;
11  @     collection .contains(p2);
12  @ person.age() == p2.age() + 1));
13  @*/
14  public boolean oneYearOlderThanSome(
15      Person person, Collection collection) { //P0
16      Iterator iter = collection .iterator ();
17      boolean acc = false; //P1
18      while(iter.hasNext()) { //P2
19          Person otherPerson = (Person)iter.next();
20          acc = acc || ( person.age() == otherPerson.age() + 1); //P3
21      } //P4
22      return acc; //Pn
```

Proof of the allOneYearOlder() method

2.2.1 Program point P₀

For program point P₀, there are no preconditions.

Assertions

The assertions in program point P₀ are the preconditions of the allOneYearOlder() method. We may assume they are satisfied.

[0.1] first != null

[0.2] (\forall Object o; first.contains(o); o instanceof Person)

[0.3] ! first.contains(null)

[0.4] second != null

[0.5] (\forall Object o; second.contains(o); o instanceof Person)

[0.6] ! second.contains(null)

2.2.2 Program point P₁

Preconditions

There is one precondition for program point P₁:

[1.a] first != null

Proof

[1.a] \rightarrow [0.1]

QED

Assertions

[1.1] first != null

[1.2] (\forall Object o; first.contains(o); o instanceof Person)

[1.3] ! first.contains(null)

[1.4] second != null

[1.5] (\forall Object o; second.contains(o); o instanceof Person)

[1.6] ! second.contains(null)

[1.7] iter != null

[1.8] iter.collection == first

[1.9] iter.previousElements.size() == 0

[1.10] acc == true

Proofs

[1.1] \dashv [0.1]

QED

[1.2] \dashv [0.2]

QED

[1.3] \dashv [0.3]

QED

[1.4] \dashv [0.4]

QED

[1.5] \dashv [0.5]

QED

[1.6] \dashv [0.6]

QED

[1.7] \dashv postcondition of Collection.iterator()

QED

[1.8] \dashv postcondition of Collection.iterator()

QED

[1.9] \dashv postcondition of Collection.iterator()

QED

[1.10] \dashv line 4

QED

2.2.3 Program point P_2

Preconditions

The only precondition for program point P_2 is:

[2.a] `iter != null`

Proof

[2.a] \dashv [1.7]

QED

Assertions

The new assertions for program point P_2 are the negation of the stop criterion, and the loop invariant.

[2.1] `first != null`

[2.2] $(\forall \text{Object } o; \text{first.contains}(o); o \text{ instanceof Person})$

[2.3] $! \text{first.contains}(\text{null})$

[2.4] `second != null`

[2.5] $(\forall \text{Object } o; \text{second.contains}(o); o \text{ instanceof Person})$

[2.6] $! \text{second.contains}(\text{null})$

[2.7] `iter != null`

[2.8] `iter.collection == first`

[2.9] `iter.hasNext()`

[LI.1] `acc == (\forall \text{Person } p;`
 `iter.previousElements.has(p);`
 $(\exists \text{Person } p2;$
 `second.contains(p2);`
 $p.\text{age}() == p2.\text{age}() + 1));$

Proofs

[2.1] \dashv [1.1]

QED

[2.2] \dashv [1.2]

QED

[2.3] \dashv [1.3]

QED

[2.4] \dashv [1.4]

QED

[2.5] \dashv [1.5]

QED

[2.6] \dashv [1.6]

QED

[2.7] \dashv [1.7]

QED

[2.8] \dashv [1.8]

QED

[2.9] \dashv line 5

QED

The loop invariant will now be proven for the initialization of the loop. The induction step is trivial since [LI.1] will appear again at program point P₃.

```
[LI.1] ⊢ acc == (∀ Person p;
                iter.previousElements.has(p);
                (∃ Person p2;
                 second.contains(p2);
                 p.age() == p2.age() + 1));
        using the postcondition of JMLObjectSequence.has()
⊢ acc == (∀ Person p;
          (∃ int i;
           0 ≤ i && i < iter.previousElement.length();
           iter.previousElement.itemAt(i) == item);
          (∃ Person p2; second.contains(p2); p.age()
           == p2.age() + 1));
        using the postcondition of JMLObjectSequence.length()
⊢ acc == (∀ Person p;
          (∃ int i;
           0 ≤ i && i < iter.previousElement.size();
           iter.previousElement.itemAt(i) == item);
          (∃ Person p2; second.contains(p2); p.age()
           == p2.age() + 1));
        using [1.9]
⊢ acc == (∀ Person p;
          (∃ int i;
           0 ≤ i && i < 0;
           iter.previousElement.itemAt(i) == item);
          (∃ Person p2; second.contains(p2); p.age()
           == p2.age() + 1));
        Since no object can satisfy the range condition, the ∀ forall clause is
        true.
⊢ acc == true
⊢ [1.10]
```

QED

2.2.4 Program point P₃

Preconditions

The preconditions for program point P₃ are:

[3.a] iter != null

[3.b] `iter.hasNext()`

[3.c] `iter.next() instanceof Person`

[3.d] `person != null`

[3.e] `second != null`

[3.f] $(\forall \text{Object } o; \text{second.contains } o; o \text{ instanceof Person})$

[3.g] $\neg \text{second.contains}(\text{null})$

Proofs

[3.a] \dashv [2.7]

QED

[3.b] \dashv [2.9]

QED

[3.c] \dashv `iter.next() instanceof Person`

\dashv `iter.nextElement().first() instanceof Person`

For the last line to be true, `iter.nextElement()` may not be empty. We know this from precondition [3.b].

`iter.hasNext()` \dashv true

`iter.nextElement().size() != 0` \dashv true

`iter.nextElement().size() > 0` \dashv true

In this case, the precondition of the `first()` method says that `itemAt(0)` will be returned.

[3.c] \dashv `iter.nextElement().itemAt(0) instanceof Person`

Of course, the number of times `itemAt(0)` is present in `nextElements` is greater than 0 since it is already present at index 0.

`iter.nextElement().count(iter.nextElement().itemAt(0)) > 0` \dashv true

A type invariant of Iterator states that the count of an element in nextElements and previousElements equals the number of explicit occurrences of that element in the collection of the iterator:

```
(\forall Object o; ;
nextElements.count(o) + previousElements.count(o) ==
collection.nbExplicitOccurrences(o));
```

From that, we can conclude that the collection must contain itemAt(0) at least once explicitly, and thus also at least once implicitly.

```
first.nbExplicitOccurrences(iter.nextElements.itemAt(0)) > 0  ⇐ true
    from the postcondition of nbOccurrences()
first.nbOccurrences(iter.nextElements.itemAt(0)) > 0  ⇐ true
    from the postcondition of contains()
first.contains(iter.nextElements.itemAt(0))  ⇐ true
    from [2.3].
iter.nextElements.itemAt(0) instance of Person  ⇐ true
```

This is what we needed to finish the proof.

```
[3.c]  ⇐ true
```

QED

```
[3.d]  ⇐ person != null
    ⇐ iter.next() != null
```

From the previous proof, we know that iter.next() is iter.nextElements.itemAt(0) and < first > contains it.

```
first.contains(iter.nextElements.itemAt(0))  ⇐ true
first.contains(iter.nextElements.first())  ⇐ true
first.contains(iter.next())  ⇐ true
first.contains(person)  ⇐ true
```

From [2.3], we can conclude that person is not null.

```
person != null  ⇐ true
[3.d]  ⇐ true
```

QED

[3.e] \dashv [2.4]

QED

[3.f] \dashv [2.5]

QED

[3.g] \dashv [2.6]

QED

Assertions

[3.1] first != null

[3.2] (\forall Object o; first.contains(o); o instanceof Person)

[3.3] ! first.contains(null)

[3.4] second != null

[3.5] (\forall Object o; second.contains(o); o instanceof Person)

[3.6] ! second.contains(null)

[3.7] iter != null

[3.8] iter.collection == first

[LI.1] acc == (\forall Person p;
iter.previousElements.has(p);
 \exists Person p2;
second.contains(p2);
p.age() == p2.age() + 1));

Proofs

[3.1] \dashv [2.1]

QED

[3.2] \dashv [2.2]

QED

[3.3] \dashv [2.3]

QED

[3.4] \vdash [2.4]

QED

[3.5] \vdash [2.5]

QED

[3.6] \vdash [2.6]

QED

[3.7] \vdash [2.7]

QED

[3.8] \vdash [2.8]

QED

To prove that the loop invariant holds, we must consider two cases.

A) $oneYearOlderThanSome(person) == false$

$acc == acc@P_2 \ \&\& \ false \ \vdash \ true$
 $acc == false \ \vdash \ true$

[LI.1] $\vdash acc == (\text{forall Person } p;$
 $\quad iter.previousElements.has(p);$
 $\quad (\text{exists Person } p2;$
 $\quad \quad second.contains(p2);$
 $\quad \quad p.age() == p2.age() + 1));$
 $\vdash false == (\text{forall Person } p; iter.previousElements.has(p);$
 $\quad (\text{exists Person } p2;$
 $\quad \quad second.contains(p2); p.age() == p2.age() + 1));$
The forall clause is false if some element of iter.previousElements
makes the expression false. For example, the last one.
 $\vdash false == (\text{exists Person } p2;$
 $\quad second.contains(p2); iter.previousElements.last().age() == p2.age() +$
 $\quad 1);$
using the postcondition of next() and line 6

\vdash false == (\exists Person p2;
 second.contains(p2); person.age() == p2.age() + 1);
using line 7 and the postcondition of oneYearOlderThanSome()
 \vdash false == oneYearOlderThanSome(person,second)
We assumed oneYearOlderThanSome(person) == false.
 \vdash false == false
 \vdash true

QED

B) oneYearOlderThanSome(person) == true

acc == acc@P₂ && true \vdash true
 acc == acc@P₂ \vdash true

[LI.1] \vdash acc == (\forall Person p;
 iter.previousElements.has(p);
 (\exists Person p2;
 second.contains(p2);
 p.age() == p2.age() + 1));
 \vdash acc@P₂ == (\forall Person p; iter.previousElements.has(p);
 (\exists Person p2;
 second.contains(p2); p.age() == p2.age() + 1);
*Since for p == iter.previousElements.last(), the \exists clause is true
 (using the assumption made, and the postcondition of
 oneYearOlderThanSome()), it may be removed from the range.*
 \vdash acc@P₂ == (\forall Person p;
 (iter.previousElements.has(p))@P₂;
 (\exists Person p2;
 second.contains(p2); p.age() == p2.age() + 1);
 \vdash [LI.1]

QED

2.2.5 Program point P₄

Preconditions

The precondition is the same as for program point P₂.

[4.a] iter != null

Proof

[4.a] \vdash [3.7]

QED

Assertions

[4.1] first != null

[4.2] (\forall Object o; first.contains(o); o instanceof Person)

[4.3] ! first.contains(null)

[4.4] second != null

[4.5] (\forall Object o; second.contains(o); o instanceof Person)

[4.6] ! second.contains(null)

[4.7] iter != null

[4.8] iter.collection == first

[LI.1] acc == (\forall Person p;
iter.previousElements.has(p);
(\exists Person p2;
second.contains(p2);
p.age() == p2.age() + 1));

[4.9] ! iter.hasNext()

Proofs

[4.1] \dashv [3.1]

QED

[4.2] \dashv [3.2]

QED

[4.3] \dashv [3.3]

QED

[4.4] \dashv [3.4]

QED

[4.5] \dashv [3.5]

QED

[4.6] \dashv [3.6]

QED

[4.7] \dashv [3.7]

QED

[4.8] \dashv [3.8]

QED

[LI.1] \dashv [LI.1]

QED

[4.9] \dashv line 5

QED

2.2.6 Program point P_n

Preconditions

For program point P_n , there are no preconditions.

Assertions

[n.1] first != null

[n.2] (\forall Object o; first.contains(o); o instanceof Person)

[n.3] ! first.contains(null)

[n.4] second != null

[n.5] (\forall Object o; second.contains(o); o instanceof Person)

[n.6] ! second.contains(null)

[n.7] iter != null

[n.8] iter.collection == first

[n.9] \result == (\forall Person p1;
 first.contains(p1);
 (\exists Person p2;
 second.contains(p2);
 p1.age() == p2.age() + 1);

[n.10] ! iter.hasNext()

Proofs

[n.1] \dashv [4.1]

QED

[n.2] \dashv [4.2]

QED

[n.3] \dashv [4.3]

QED

[n.4] \dashv [4.4]

QED

[n.5] \dashv [4.5]

QED

[n.6] \dashv [4.6]

QED

[n.7] \dashv [4.7]

QED

[n.8] \dashv [4.8]

QED

From [4.9] and the postcondition of `hasNext()`, we can conclude that `iter.nextElement` is empty.

`iter.nextElement.size() == 0` \dashv `true`

From the type invariant of Iterator, we know:

```

(\forall Object o;;                                      $\dashv$  true
iter.previousElements.count(o) +
iter.nextElement.count(o) ==
iter.collection.nbExplicitOccurrences(o))
(\forall Object o;;                                      $\dashv$  true
iter.previousElements.count(o) +
(\num.of int i; i >= 0 && i <
iter.nextElement.length() &&
iter.nextElement.itemAt(i) == o;1) ==
iter.collection.nbExplicitOccurrences(o))
    since iter.nextElement.size() == 0
(\forall Object o;;                                      $\dashv$  true
iter.previousElements.count(o) ==
iter.collection.nbExplicitOccurrences(o))
(\forall Object o;;                                      $\dashv$  true
iter.previousElements.count(o) > 0  $\vdash$ 
iter.collection.nbExplicitOccurrences(o)
> 0
(\forall Object o;;                                      $\dashv$  true
iter.previousElements.has(o)  $\vdash$ 
iter.collection.nbExplicitOccurrences(o)
> 0
(\forall Object o;;                                      $\dashv$  true
iter.previousElements.has(o)  $\vdash$ 
iter.collection.nbOccurrences(o) > 0
(\forall Object o;;                                      $\dashv$  true
iter.previousElements.has(o)  $\vdash$ 
iter.collection.contains(o))

```

```

[n.9]  $\dashv$  \result == (\forall Person p1;
    first.contains(p1);
    (\exists Person p2;
    second.contains(p2);
    p1.age() == p2.age() + 1);
    using line 22 and [4.8]
 $\dashv$  acc == (\forall Person p1;
    iter.collection.contains(p1);
    (\exists Person p2;
    second.contains(p2);
    p1.age() == p2.age() + 1);

```

\vdash acc == (\forall Person p1;
 iter.previousElements.has(p1);
 \exists Person p2;
 second.contains(p2);
 p1.age() == p2.age() + 1);
 \vdash [LI.1]

QED

[n.10] \vdash [4.9]

QED

Quod Erat Demonstrandum

Proof of the oneYearOlderThanSome() method

2.2.7 Program point P₀

For program point P₀, there are no preconditions.

Assertions

The assertions in program point P₀ are the preconditions of the oneYearOlderThanSome() method. We may assume they are satisfied.

[0.1] person != null

[0.2] collection != null

[0.3] (\forall Object o; collection.contains(o); o instanceof Person)

[0.4] ! collection.contains(null)

2.2.8 Program point P₁

Preconditions

There is one precondition for program point P₁:

[1.a] collection != null

Proof

[1.a] \vdash [0.2]

QED

Assertions

[1.1] `person != null`

[1.2] `collection != null`

[1.3] `(\forall \text{Object } o; \text{collection.contains}(o); o \text{ instanceof } \text{Person})`

[1.4] `! collection.contains(null)`

[1.5] `iter != null`

[1.6] `iter.collection == collection`

[1.7] `iter.previousElements.size() == 0`

[1.8] `acc == false`

Proofs

[1.1] \dashv [0.1]

QED

[1.2] \dashv [0.2]

QED

[1.3] \dashv [0.3]

QED

[1.4] \dashv [0.4]

QED

[1.5] \dashv postcondition of `Collection.iterator()`

QED

[1.6] \dashv postcondition of `Collection.iterator()`

QED

[1.7] \dashv postcondition of `Collection.iterator()`

QED

[1.8] \dashv line 17

QED

2.2.9 Program point P_2

Preconditions

The only precondition for program point P_2 is:

[2.a] `iter != null`

Proof

[2.a] \dashv [1.5]

QED

Assertions

The new assertions for program point P_2 are the negation of the stop criterion, and the loop invariant.

[2.1] `person != null`

[2.2] `collection != null`

[2.3] $(\forall \text{Object } o; \text{collection.contains}(o); o \text{ instanceof Person})$

[2.4] `! collection.contains(null)`

[2.5] `iter != null`

[2.6] `iter.collection == collection`

[2.7] `iter.hasNext()`

[LI.1] `acc == (\exists \text{Person } p;`
 `iter.previousElements.has(p);`
 `person.age() == p.age() + 1);`

Proofs

[2.1] \dashv [1.1]

QED

[2.2] \dashv [1.2]

QED

[2.3] \dashv [1.3]

QED

[2.4] \dashv [1.4]

QED

[2.5] \dashv [1.5]

QED

[2.6] \dashv [1.6]

QED

[2.7] \dashv line 18

QED

The loop invariant will now be proven for the initialization of the loop. The induction step is trivial since [LI.1] will appear again at program point P_3 .

```
[LI.1]  $\dashv$  acc == (\exists Person p;
                iter.previousElements.has(p);
                person.age() == p.age() + 1);
        using the postcondition of JMLObjectSequence.has()
 $\dashv$  acc == (\exists Person p;
            (\exists int i;
            0 <= i && i < iter.previousElement.length();
            iter.previousElement.itemAt(i) == item);
            person.age() == p.age() + 1);
        using the postcondition of JMLObjectSequence.length()
 $\dashv$  acc == (\exists Person p;
            (\exists int i;
            0 <= i && i < iter.previousElement.size();
            iter.previousElement.itemAt(i) == item);
            person.age() == p.age() + 1);
        using [1.7]
 $\dashv$  acc == (\exists Person p;
            (\exists int i;
            0 <= i && i < 0;
            iter.previousElement.itemAt(i) == item);
            person.age() == p.age() + 1);
 $\dashv$  acc == false
 $\dashv$  [1.8]
```

QED

2.2.10 Program point P₃

Preconditions

The preconditions for program point P₃ are:

[3.a] `iter != null`

[3.b] `iter.hasNext()`

[3.c] `iter.next() instanceof Person`

[3.d] `otherPerson != null`

[3.e] `otherPerson.age() <= Integer.MAX_VALUE - 1`

Proofs

[3.a] \dashv [2.5]

QED

[3.b] \dashv [2.7]

QED

[3.c] \dashv `iter.next() instanceof Person`
 \dashv `iter.nextElement().first() instanceof Person`

For the last line to be true, `iter.nextElement()` may not be empty. We know this from precondition [3.b].

`iter.hasNext() \dashv true`
`iter.nextElement().size() != 0 \dashv true`
`iter.nextElement().size() > 0 \dashv true`

In this case, the precondition of the `first()` method says that `itemAt(0)` will be returned.

[3.c] \dashv `iter.nextElement().itemAt(0) instanceof Person`

Of course, the number of times `itemAt(0)` is present in `nextElements` is greater than 0 since it is already present at index 0.

`iter.nextElement().count(iter.nextElement().itemAt(0)) > 0 \dashv true`

A type invariant of Iterator states that the count of an element in nextElements and previousElements equals the number of explicit occurrences of that element in the collection of the iterator:

```
(\forall Object o; ;
nextElements.count(o) + previousElements.count(o) ==
collection.nbExplicitOccurrences(o));
```

From that, we can conclude that the collection must contain itemAt(0) at least once explicitly, and thus also at least once implicitly.

```
collection.nbExplicitOccurrences(iter.nextElements.itemAt(0)) > 0  + true
      from the postcondition of nbOccurrences()
collection.nbOccurrences(iter.nextElements.itemAt(0)) > 0  + true
      from the postcondition of contains()
collection.contains(iter.nextElements.itemAt(0))  + true
      from [2.3].
iter.nextElements.itemAt(0) instance of Person  + true
```

This is what we needed to finish the proof.

```
[3.c]  + true
```

QED

```
[3.d]  + otherPerson != null
      + iter.next() != null
```

From the previous proof, we know that iter.next() is iter.nextElements.itemAt(0) and <collection> contains it.

```
collection.contains(iter.nextElements.itemAt(0))  + true
collection.contains(iter.nextElements.first())    + true
collection.contains(iter.next())                  + true
collection.contains(otherPerson)                  + true
```

From [2.4], we can conclude that otherPerson is not null.

```
otherPerson != null  + true
[3.d]                + true
```

QED

[3.e] \vdash otherPerson.age() \leq Integer.MAX_VALUE - 1
 \vdash otherPerson.age() \leq 2000
We know this is true because of the last type invariant of Person.
 \vdash true

QED

Assertions

[3.1] person \neq null

[3.2] collection \neq null

[3.3] (\forall Object o; collection.contains(o); o instanceof Person)

[3.4] ! collection.contains(null)

[3.5] iter \neq null

[3.6] iter.collection == collection

[LI.1] acc == (\exists Person p;
iter.previousElements.has(p);
person.age() == p.age() + 1);

Proofs

[3.1] \vdash [2.1]

QED

[3.2] \vdash [2.2]

QED

[3.3] \vdash [2.3]

QED

[3.4] \vdash [2.4]

QED

[3.5] \vdash [2.5]

QED

[3.6] \dashv [2.6]

QED

To prove that the loop invariant holds, we must consider two cases.

A) $person.age() == otherPerson.age() + 1$

$acc == acc@P_2 \parallel true \dashv true$
 $acc == true \dashv true$

[LI.1] $\dashv acc == (\exists Person\ p;$
 $iter.previousElements.has(p);$
 $person.age() == p.age() + 1);$
 $\dashv true == (\exists Person\ p;$
 $iter.previousElements.has(p);$
 $person.age() == p.age() + 1);$

The \exists clause is true if some element of $iter.previousElements$ makes the expression true. For example, the last one.

$\dashv true ==$
 $(person.age() == ((Person)iter.previousElements.last()).age() + 1)$
using the postcondition of $next()$ and line 19

$\dashv true ==$
 $(person.age() == (((Person)iter.next())@P_2).age() + 1)$
using line 19

$\dashv true == (person.age() == otherPerson.age() + 1)$
We assumed $person.age() == otherPerson.age() + 1$.

$\dashv true == true$

$\dashv true$

QED

B) $person.age() != otherPerson.age() + 1$

$acc == acc@P_2 \parallel false \dashv true$
 $acc == acc@P_2 \dashv true$

[LI.1] $\dashv acc == (\exists Person\ p;$
 $iter.previousElements.has(p);$
 $person.age() == p.age() + 1);$

\vdash $\text{acc}@P_2 == (\exists \text{ Person } p;$
 $\quad \text{iter.previousElements.has}(p);$
 $\quad \text{person.age()} == p.\text{age()} + 1);$
Since for $p == \text{iter.previousElements.last}()$, the expression is false, it
may be removed from the range.
 \vdash $\text{acc}@P_2 == (\exists \text{ Person } p;$
 $\quad (\text{iter.previousElements.has}(p))@P_2;$
 $\quad \text{person.age()} == p.\text{age()} + 1);$
 \vdash [LI.1]

QED

2.2.11 Program point P_4

Preconditions

The precondition is the same as for program point P_2 .

[4.a] $\text{iter} \neq \text{null}$

Proof

[4.a] \vdash [3.5]

QED

Assertions

[4.1] $\text{person} \neq \text{null}$

[4.2] $\text{collection} \neq \text{null}$

[4.3] $(\forall \text{ Object } o; \text{collection.contains}(o); o \text{ instanceof Person})$

[4.4] $! \text{collection.contains}(\text{null})$

[4.5] $\text{iter} \neq \text{null}$

[4.6] $\text{iter.collection} == \text{collection}$

[LI.1] $\text{acc} == (\exists \text{ Person } p;$
 $\quad \text{iter.previousElements.has}(p);$
 $\quad \text{person.age()} == p.\text{age()} + 1);$

[4.7] $! \text{iter.hasNext}()$

Proofs

[4.1] \dashv [3.1]

QED

[4.2] \dashv [3.2]

QED

[4.3] \dashv [3.3]

QED

[4.4] \dashv [3.4]

QED

[4.5] \dashv [3.5]

QED

[4.6] \dashv [3.6]

QED

[LI.1] \dashv [LI.1]

QED

[4.7] \dashv line 18

QED

2.2.12 Program point P_n

Preconditions

For program point P_n , there are no preconditions.

Assertions

[n.1] person != null

[n.2] collection != null

[n.3] (\forall Object o; collection.contains(o); o instanceof Person)

[n.4] ! collection.contains(null)

[n.5] iter != null

[n.6] iter.collection == collection

[n.7] \backslash result == (\exists Person p;
collection.contains(p);
person.age() == p.age() + 1);

[n.8] ! iter.hasNext()

Proofs

[n.1] \dashv [4.1]

QED

[n.2] \dashv [4.2]

QED

[n.3] \dashv [4.3]

QED

[n.4] \dashv [4.4]

QED

[n.5] \dashv [4.5]

QED

[n.6] \dashv [4.6]

QED

From [4.7] and the postcondition of hasNext(), we can conclude that iter.nextElement() is empty.

iter.nextElement().size() == 0 \vdash true

From the type invariant of Iterator, we know:

(\forall Object o;; \vdash true
iter.previousElements.count(o) +
iter.nextElement().count(o) ==
iter.collection.nbExplicitOccurrences(o))

(\forall Object o;; \vdash true
iter.previousElements.count(o) +
(\forall num_of int i; i >= 0 && i <
iter.nextElement().length() &&
iter.nextElement().itemAt(i) == o;1) ==
iter.collection.nbExplicitOccurrences(o))
since iter.nextElement().size() == 0

(\forall Object o;; \vdash true
iter.previousElements.count(o) ==
iter.collection.nbExplicitOccurrences(o))

(\forall Object o;; \vdash true
iter.previousElements.count(o) > 0 \vdash
iter.collection.nbExplicitOccurrences(o)
> 0

(\forall Object o;; \vdash true
iter.previousElements.has(o) \vdash
iter.collection.nbExplicitOccurrences(o)
> 0

(\forall Object o;; \vdash true
iter.previousElements.has(o) \vdash
iter.collection.nbOccurrences(o) > 0

(\forall Object o;; \vdash true
iter.previousElements.has(o) \vdash
iter.collection.contains(o))

[n.7] \vdash \result == (\exists Person p;
collection.contains(p);
person.age() ==
p.age() + 1);

using line 22

\vdash acc == (\exists Person p;
iter.collection.contains(p);
person.age() ==
p.age() + 1);

```

+ acc == (\exists Person p;
          iter.previousElements.has(p);
          person.age()      ==
          p.age() + 1);
+ [LI.1]

```

QED

[n.8] + [4.7]

QED

Quod Erat Demonstrandum

2.3 Proof of correctness for a recursive implementation.

We will now see and prove a recursive version of the `allOneYearOlder` method. The recursive version will also be split into two methods. Listing 7 shows the implementation of the `allOneYearOlder` method. Listing 8 shows the `oneYearOlderThanSome` method. Once again we've used an inefficient implementation to make the proof a bit easier.

Listing 7: Implementation of the `oneYearOlderThanSome` method.

```

1 public boolean allOneYearOlder(Collection first,
2                               Collection second) {           //P0
3     boolean result = true;
4     if ( first . size () != 0 ) {
5         Person person = (Person)first.toArray() [0];
6         Collection newFirst = new Vector(first);           //P1
7         newFirst.remove(other);                             //P2
8         result = ((oneYearOlderThanSome(person, second)) &&
9                 (allOneYearOlder(newFirst,second)));       //P3
10    }                                                         //P4
11    return result;                                           //Pn
12 }

```

Proof of correctness of the `allOneYearOlder()` method.

2.3.1 Program point P_0

Preconditions

Again, the assertions at program point P_0 are the preconditions of the method.

Listing 8: Implementation of the oneYearOlderThanSome method.

```

1  /*@
2   @ pre person != null;
3   @ pre collection != null;
4   @ pre ! collection .contains(null);
5   @ pre (\forall Object o; collection .contains(o);
6   @     o instanceof Person);
7   @
8   @ post \result == (\exists Person p2; collection .contains(p2);
9   @     person.age() == p2.age() +1);
10  @*/
11  public boolean oneYearOlderThanSome(Person person,
12     Collection collection) { //P0
13      boolean result = false;
14      if( collection . size () != 0) {
15          Person other = (Person)collection.toArray() [0];
16          Collection newCollection = new Vector(collection); //P1
17          newCollection.remove(other); //P2
18          result = ((person.age() == other.age() + 1) ||
19              (oneYearOlderThanSome(person, newCollection))); //P3
20      } //P4
21      return result; //Pn
22  }

```

- [0.1] first != null
- [0.2] second != null
- [0.3] (\forall Object o; first.contains(o); o instanceof Person)
- [0.4] (\forall Object o; first.contains(o); o instanceof Person)
- [0.5] ! first.contains(null)
- [0.6] ! second.contains(null)

2.3.2 Program point P₁

Preconditions

The preconditions for program point P₁ are:

- [1.a] first != null
- [1.b] first.toArray().length > 0

Proofs

[1.a] \dashv [0.2]

QED

[1.b] \dashv first.toArray().length > 0
postcondition of toArray()
 \dashv first.size() > 0
 \dashv first.size() >= 0 && first.size != 0
invariant of Collection
 \dashv first.size() != 0
 \dashv line 14

QED

Assertions

The assertion for program point P₁ are:

- [1.1] first != null
- [1.2] second != null
- [1.3] (\forall Object o; first.contains(o); o instanceof Person)

[1.4] (\forall Object o; first.contains(o); o instanceof Person)

[1.5] ! first.contains(null)

[1.6] ! second.contains(null)

[1.7] first.size() > 0

[1.8] newFirst != null

[1.9] (\forall Object o; ;first.contains(o) == newFirst.contains(o))

[1.10] ! newFirst.contains(null)

[1.11] (\forall Object o; newFirst.contains(o); o instanceof Person)

[1.12] first.contains(person)

[1.13] newFirst.size() == first.size()

Proofs

[1.1] \rightarrow [0.1]

QED

[1.2] \rightarrow [0.2]

QED

[1.3] \rightarrow [0.3]

QED

[1.4] \rightarrow [0.4]

QED

[1.5] \rightarrow [0.5]

QED

[1.6] \rightarrow [0.6]

QED

[1.7] *line 13, line 14 and a class invariant of Collection stating the the size() is zero or positive.*

⊢ true

QED

[1.8] ⊢ newFirst != null

From [1.a], we know that the constructor can no throw an exception.

⊢ line 17

QED

[1.9] ⊢ (\forall Object o; first.contains(o) == newFirst.contains(o))

This is a postcondition of the constructor of Vector with a Collection as argument.

⊢ true

QED

[1.10] ⊢ ! newFirst.contains(null)

[1.9] && [1.5]

⊢ true

QED

[1.11] ⊢ (\forall Object o; newFirst.contains(o); o instanceof Person)

using [1.9]

⊢ (\forall Object o; first.contains(o); o instanceof Person)

⊢ [1.5]

QED

From the postcondition of the toArray() method of Collection, we know:

```

(\forall Object o;                                     + true
 first.nbExplicitOccurrences(o) ==
 (\num_of int i; i >= 0 && i < size();
  first.toArray()[i] == o) )
first.nbExplicitOccurrences(first.toArray()[0]) ==   + true
(\num_of int i; i >= 0 && i < size();
 first.toArray()[i] == first.toArray()[0]) )
                                                    + true

The num_of expression is of course positive.
first.nbExplicitOccurrences(first.toArray()[0]) > 0 + true

```

QED

```

[1.12] + first.contains(person)
        + first.nbOccurrences(person) > 0
        + first.nbExplicitOccurrences(first.toArray()[0]) > 0
        + true

```

QED

[1.13] *This is a postcondition of the constructor of Vector.*

QED

2.3.3 Program point P₂

Preconditions

The preconditions for program point P₂ are:

[2.a] newFirst != null

Proof

[2.a] + [1.8]

QED

Since we use a Vector, the UnsupportedOperationException will not be thrown. The remove() method is supported for a Vector.

Assertions

The assertions for program point P_2 are:

[2.1] $\text{first} \neq \text{null}$

[2.2] $\text{second} \neq \text{null}$

[2.3] $(\forall \text{Object } o; \text{first.contains}(o); o \text{ instanceof Person})$

[2.4] $(\forall \text{Object } o; \text{first.contains}(o); o \text{ instanceof Person})$

[2.5] $! \text{first.contains}(\text{null})$

[2.6] $! \text{second.contains}(\text{null})$

[2.7] $\text{first.size}() > 0$

[2.8] $\text{newFirst} \neq \text{null}$

[2.9] $\text{first.contains}(\text{person})$

[2.10] $(\forall \text{Object } o; ;$
 $! \text{person.equals}(o) \Rightarrow$
 $\text{first.contains}(o) == \text{newFirst.contains}(o))$

[2.11] $! \text{newFirst.contains}(\text{null})$

[2.12] $(\forall \text{Object } o; \text{newFirst.contains}(o); o \text{ instanceof Person})$

[2.13] $(\forall \text{Object } o; \text{newFirst.contains}(o); \text{first.contains}(o))$

[2.14] $\text{newFirst.size}() == \text{first.size}() - 1$

Proofs

[2.1] \dashv [1.1]

QED

[2.2] \dashv [1.2]

QED

[2.3] \dashv [1.3]

QED

[2.4] \dashv [1.4]

QED

[2.5] \dashv [1.5]

QED

[2.6] \dashv [1.6]

QED

[2.7] \dashv [1.7]

QED

[2.8] \dashv [1.8]

QED

[2.9] \dashv [1.12]

QED

[2.10] \dashv (\forall forall Object o ;
! person.equals(o) \Rightarrow
first.contains(o) == newFirst.contains(o))

From [2.9], we know that person is in first, so according to [2.5], other can not be null. Therefore no NullPointerException can be thrown.

From the postcondition of the remove() method, we see that only nbOccurrences(person) can change. So if o does not equal person, nbOccurrences(o) does not change, and thus contains(o) does not change. For these objects, [1.9] is still valid.

\dashv true

QED

[2.11] \dashv ! newFirst.contains(null)

Since person can not be null, first.nbOccurrences(null) does not change.

\dashv ! first@P₂.contains(null)

\dashv true

QED

[2.12] \vdash (\forall Object o; newFirst.contains(o); o instanceof Person)
Since only the collection has changed, and only elements have been removed, this follows from [1.11].
 \vdash true

QED

[2.14] \vdash newFirst.size() == first.size() - 1
from [1.12] and the postcondition of remove(), we know the size of newFirst has decreased by 1.
 \vdash newFirst.size()@P₂.size() - 1 == first.size() - 1
 \vdash newFirst.size()@P₂.size() == first.size()
 \vdash [1.13]

QED

[2.13] \vdash (\forall Object o; newFirst.contains(o); first.contains(o))
 \vdash (\forall Object o; newFirst.nbOccurrences(o) > 0;
first.nbOccurrences(o) > 0)
 \vdash (\forall Object o; newFirst.nbOccurrences(o)@P₂ > 0;
first.nbOccurrences(o)@P₁ > 0)
 \vdash (\forall Object o; newFirst.nbOccurrences(o)@P₂ > 0;
first.nbOccurrences(o)@P₁ > 0)
 \vdash From a redundant postcondition of the remove() method we know that nbOccurrences(o) can only decrease.
 \vdash (\forall Object o; newFirst.nbOccurrences(o)@P₁ > 0;
first.nbOccurrences(o)@P₁ > 0)
 \vdash (\forall Object o; newFirst.contains(o)@P₁; first.contains(o)@P₁)
 \vdash [1.9]

QED

2.3.4 Program point P₃

Preconditions

The preconditions for program point P₃ are:

[3.a] p erson != null

[3.b] s econd != null

[3.c] newFirst != null

[3.d] (\forall \text{Object } o; \text{second.contains}(o); o \text{ instanceof Person});

[3.e] (\forall \text{Object } o; \text{newFirst.contains}(o); o \text{ instanceof Person});

[3.f] ! second.contains(null)

[3.g] ! newFirst.contains(null)

[3.h] other.age() <= Integer.MAX_VALUE - 1

Proofs

[3.a] \dashv [2.9] and [2.5]

QED

[3.b] \dashv [2.2]

QED

[3.c] \dashv [2.8]

QED

[3.d] \dashv [2.4]

QED

[3.e] \dashv [2.12]

QED

[3.f] \dashv [2.6]

QED

[3.g] \dashv [2.11]

QED

[3.h] \dashv other.age() <= Integer.MAX_VALUE - 1

\dashv other.age() <= 2000

From the third type invariant of Person, we know this is true.

\dashv true

QED

Assertions

The assertions for program point P_3 are:

[3.1] $\text{first.size()} > 0 \Rightarrow \text{result} ==$
 $(\forall \text{ Person } p1; \text{first.contains}(p1);$
 $(\exists \text{ Person } p2; \text{second.contains}(p2);$
 $p1.\text{age}() == p2.\text{age}() + 1))$

Proof

Since a boolean expression is either true or false, and $\text{person} \neq \text{null}$ because it is an element of collection, we know:

$(\forall \text{ Object } o; ! \text{person.equals}(o) \parallel \text{person.equals}(o)) \dashv \vdash \text{true}$

[3.1] $\dashv \vdash \text{first.size()} > 0 \Rightarrow \text{result} ==$
 $(\forall \text{ Person } p1; \text{first.contains}(p1);$
 $(\exists \text{ Person } p2; \text{second.contains}(p2);$
 $p1.\text{age}() == p2.\text{age}() + 1))$
 Using the previous assertion, we can split this in the cases
 !person.equals(o) and person.equals(o)
 $\dashv \vdash \text{first.size()} > 0 \Rightarrow \text{result} ==$
 $(\forall \text{ Person } p1; \text{first.contains}(p1) \ \&\& \ !\text{person.equals}(p1);$
 $(\exists \text{ Person } p2; \text{second.contains}(p2);$
 $p1.\text{age}() == p2.\text{age}() + 1)) \ \&\&$
 $(\forall \text{ Person } p1; \text{first.contains}(p1) \ \&\& \ \text{person.equals}(p1);$
 $(\exists \text{ Person } p2; \text{second.contains}(p2);$
 $p1.\text{age}() == p2.\text{age}() + 1))$
 using the postconditions en invariants of Collection, we can change this
 into:
 $\dashv \vdash \text{first.size()} > 0 \Rightarrow \text{result} ==$
 $(\forall \text{ Person } p1; \text{first.contains}(p1) \ \&\& \ !\text{person.equals}(p1);$
 $(\exists \text{ Person } p2; \text{second.contains}(p2);$
 $p1.\text{age}() == p2.\text{age}() + 1)) \ \&\&$
 $(\forall \text{ Person } p1; \text{first.contains}(\text{person}) \ \&\& \ \text{per-}$
 $\text{son.equals}(p1);$
 $(\exists \text{ Person } p2; \text{second.contains}(p2);$
 $p1.\text{age}() == p2.\text{age}() + 1))$
 we know from [2.9] that first.contains(person) is true.

- \dashv `first.size() > 0 \Rightarrow result ==`
`(\forall Person p1; first.contains(p1) && !person.equals(p1);`
`(\exists Person p2; second.contains(p2);`
`p1.age() == p2.age() + 1)) &&`
`(\forall Person p1; person.equals(p1);`
`(\exists Person p2; second.contains(p2);`
`p1.age() == p2.age() + 1))`
from the invariants of class Person, we know that since person and p2 are equal, their age must be the same.
- \dashv `first.size() > 0 \Rightarrow result ==`
`(\forall Person p1; first.contains(p1) && !person.equals(p1);`
`(\exists Person p2; second.contains(p2);`
`p1.age() == p2.age() + 1)) &&`
`(\forall Person p1; person.equals(p1);`
`(\exists Person p2; second.contains(p2);`
`person.age() == p2.age() + 1))`
since p1 does not occur anymore in the assertio of the second forall quantifier, that quantifier can be removed.
- \dashv `first.size() > 0 \Rightarrow result ==`
`(\forall Person p1; first.contains(p1) && !person.equals(p1);`
`(\exists Person p2; second.contains(p2);`
`p1.age() == p2.age() + 1)) &&`
`(\exists Person p2; second.contains(p2);`
`person.age() == p2.age() + 1)`
From [2.13], we know that !person.equals(p1) \Rightarrow first.contains(p1) == newFirst.contains(p1).
- \dashv `first.size() > 0 \Rightarrow result ==`
`(\forall Person p1; newFirst.contains(p1) && !per-`
`son.equals(p1);`
`(\exists Person p2; second.contains(p2);`
`p1.age() == p2.age() + 1)) &&`
`(\exists Person p2; second.contains(p2);`
`person.age() == p2.age() + 1)`
Because we may weaken the condition for the forall quantifier, we can say:
- \dashv `first.size() > 0 \Rightarrow result ==`
`(\forall Person p1; newFirst.contains(p1);`
`(\exists Person p2; second.contains(p2);`
`p1.age() == p2.age() + 1)) &&`
`(\exists Person p2; second.contains(p2);`
`person.age() == p2.age() + 1)`
The forall quantifier is the result of the allOneYearOlder method applied to newFirst and second. The last exists quantifier is the result of the oneYearOlderThanSome method applied to person and second.

⊢ line 14, line 18 and line 19

QED

2.3.5 Program point P_n

Assertions

The assertion for program point P_n is:

[n.1] $\backslash\text{result} == (\backslash\text{forall Person } p1; \text{first.contains}(p1);$
 $(\backslash\text{exists Person } p2; \text{second.contains}(p2);$
 $p1.\text{age}() == p2.\text{age}() + 1))$

Proof

[n.1] ⊢ $\backslash\text{result} == (\backslash\text{forall Person } p1; \text{first.contains}(p1);$
 $(\backslash\text{exists Person } p2; \text{second.contains}(p2);$
 $p1.\text{age}() == p2.\text{age}() + 1))$

From line 21 we know that result is returned.

⊢ $\text{result} == (\backslash\text{forall Person } p1; \text{first.contains}(p1);$
 $(\backslash\text{exists Person } p2; \text{second.contains}(p2);$
 $p1.\text{age}() == p2.\text{age}() + 1))$

⊢ $((\text{first.size}() > 0) \&\&$
 $\text{result} == (\backslash\text{forall Person } p1; \text{first.contains}(p1);$
 $(\backslash\text{exists Person } p2; \text{second.contains}(p2);$
 $p1.\text{age}() == p2.\text{age}() + 1)) \&\&$
 $((\text{first.size}() == 0) \&\&$
 $\text{result} == (\backslash\text{forall Person } p1; \text{first.contains}(p1);$
 $(\backslash\text{exists Person } p2; \text{second.contains}(p2);$
 $p1.\text{age}() == p2.\text{age}() + 1))$

⊢ [3.1] $\&\&$
 $((\text{first.size}() == 0) \&\&$
 $\text{result} == (\backslash\text{forall Person } p1; \text{first.contains}(p1);$
 $(\backslash\text{exists Person } p2; \text{second.contains}(p2);$
 $p1.\text{age}() == p2.\text{age}() + 1))$

If first is empty, first.contains(p1) will always be false.

⊢ [3.1] $\&\&$
 $((\text{first.size}() == 0) \&\&$
 $\text{result} == (\backslash\text{forall Person } p1; \text{false};$
 $(\backslash\text{exists Person } p2; \text{second.contains}(p2);$
 $p1.\text{age}() == p2.\text{age}() + 1))$

This means the forall quantifier is true.

\vdash [3.1] $\&\&$
 $((\text{first.size()} == 0) \&\& \text{result} == \text{true})$
 \vdash [3.1] \parallel
line 13, line 14 and line 21
 \vdash true

QED

To prove that the method will end, we note that if $\text{first.size()} == 0$, no method invocation or loops occur, so that case ends in a finite amount of time. If $\text{first.size()} > 0$, we see that the size of `newFirst` is one less than the size of `first`. Because `newFirst` is given as `first` parameter to the recursive method invocation, the method will always come to the point where $\text{first.size()} == 0$, and so the method will end.

Quod erat demonstrandum

Proof of correctness of the `oneYearOlderThanSome()` method.

2.3.6 Program point P_0

Preconditions

Again, the assertions at program point P_0 are the preconditions of the method.

[0.1] `person != null`

[0.2] `collection != null`

[0.3] `! collection.contains(null)`

[0.4] $(\forall \text{Object } o; \text{collection.contains}(o); o \text{ instanceof Person})$

2.3.7 Program point P_1

Preconditions

The preconditions for program point P_1 are:

[1.a] `collection != null`

[1.b] `collection.toArray().length > 0`

Proofs

[1.a] \vdash [0.2]

QED

- [1.b] \vdash `collection.toArray().length > 0`
postcondition of toArray()
- \vdash `collection.size() > 0`
- \vdash `collection.size() >= 0 && collection.size != 0`
invariant of Collection
- \vdash `collection.size() != 0`
- \vdash line 14

QED

Assertions

The assertion for program point P_1 are:

- [1.1] `person != null`
- [1.2] `collection != null`
- [1.3] `! collection.contains(null)`
- [1.4] `(\forallall Object o; collection.contains(o); o instanceof Person)`
- [1.5] `collection.size() > 0`
- [1.6] `newCollection != null`
- [1.7] `(\forallall Object o; ;collection.contains(o) == newCollection.contains(o))`

- [1.8] `! newCollection.contains(null)`
- [1.9] `(\forallall Object o; newCollection.contains(o); o instanceof Person)`
- [1.10] `collection.contains(other)`
- [1.11] `newCollection.size() == collection.size()`

Proofs

- [1.1] \vdash [0.1]

QED

- [1.2] \vdash [0.2]

QED

[1.3] \dashv [0.3]

QED

[1.4] \dashv [0.4]

QED

[1.5] *line 13, line 14 and a class invariant of Collection stating the the size() is zero or positive.*

\dashv true

QED

[1.6] \dashv newCollection != null

\dashv line 17

QED

[1.7] \dashv (\forall Object o; ;collection.contains(o) == newCollection.contains(o))

This is a postcondition of the constructor of Vector with a Collection as argument.

\dashv true

QED

[1.8] \dashv ! newCollection.contains(null)

[1.7] && [1.3]

\dashv true

QED

[1.9] \dashv (\forall Object o; newCollection.contains(o); o instanceof Person)
using [1.7]

\dashv (\forall Object o; collection.contains(o); o instanceof Person)

\dashv [1.4]

QED

From the postcondition of the `toArray()` method of `Collection`, we know:

$$\begin{aligned}
 & (\text{forall Object } o; \quad \quad \quad \vdash \text{ true} \\
 & \text{collection.nbExplicitOccurrences}(o) \\
 & == \\
 & (\text{num_of int } i; i \geq 0 \ \&\& \ i < \text{size}(); \\
 & \text{collection.toArray}()[i] == o)) \\
 & \text{collection.nbExplicitOccurrences}(\text{collection.toArray}()[0]) \vdash \text{ true} \\
 & == \\
 & (\text{num_of int } i; i \geq 0 \ \&\& \ i < \text{size}(); \\
 & \text{collection.toArray}()[i] == \text{collection.toArray}()[0])) \\
 & \quad \quad \quad \vdash \text{ true}
 \end{aligned}$$

The num_of expression is of course positive.

$$\text{collection.nbExplicitOccurrences}(\text{collection.toArray}()[0]) > 0 \quad \vdash \text{ true}$$

QED

$$\begin{aligned}
 [1.10] \quad & \vdash \text{collection.contains}(\text{other}) \\
 & \vdash \text{collection.nbOccurrences}(\text{other}) > 0 \\
 & \vdash \text{collection.nbExplicitOccurrences}(\text{collection.toArray}()[0]) > 0 \\
 & \vdash \text{true}
 \end{aligned}$$

QED

$$[1.11] \quad \text{this is a postcondition of the constructor of Vector.}$$

QED

2.3.8 Program point P₂

Preconditions

The preconditions for program point P₂ are:

$$[2.a] \quad \text{newCollection} \neq \text{null}$$

Proof

$$[2.a] \quad \vdash [1.6]$$

QED

Since we use a `Vector`, the `UnsupportedOperationException` will not be thrown. The `remove()` method is supported for a `Vector`.

Assertions

The assertions for program point P_2 are:

[2.1] `person != null`

[2.2] `collection != null`

[2.3] `! collection.contains(null)`

[2.4] `(\forall \text{Object } o; \text{collection.contains}(o); o \text{ instanceof } \text{Person})`

[2.5] `collection.size() > 0`

[2.6] `newCollection != null`

[2.7] `collection.contains(other)`

[2.8] `(\forall \text{Object } o; ;`
`! other.equals(o) \Rightarrow`
`collection.contains(o) == newCollection.contains(o))`

[2.9] `! newCollection.contains(null)`

[2.10] `(\forall \text{Object } o; newCollection.contains(o); o \text{ instanceof } \text{Person})`

[2.11] `(\forall \text{Object } o; newCollection.contains(o); collection.contains(o))`

[2.12] `newCollection.size() == collection.size() - 1`

Proofs

[2.1] \dashv [1.1]

QED

[2.2] \dashv [1.2]

QED

[2.3] \dashv [1.3]

QED

[2.4] \dashv [1.4]

QED

[2.5] \dashv [1.5]

QED

[2.6] \dashv [1.6]

QED

[2.7] \dashv [1.10]

QED

[2.8] \dashv (\forall Object o ; ;
! other.equals(o) \Rightarrow
collection.contains(o) == newCollection.contains(o)
From [2.7], we know that other is in collection, so according to [2.3] other can not be null. Therefore no NullPointerException can be thrown.
From the postcondition of the remove method, we see that only nbOccurrences(other) can change. So if o does not equal other, nbOccurrences(o) does not change, and thus contains(o) does not change. For these objects, [1.7] is still valid.
 \dashv true

QED

[2.9] \dashv ! newCollection.contains(null)
Since other can not be null, collection.nbOccurrences(null) does not change.
 \dashv ! collection@ P_2 .contains(null)
 \dashv true

QED

[2.10] \dashv (\forall Object o ; newCollection.contains(o); o instanceof Person)
Since only the collection has changed, and only elements have been removed, this follows from [1.9].
 \dashv true

QED

[2.12] \vdash `newCollection.size() == collection.size() - 1`
from [1.10] and the postcondition of `remove()`, we know the size of `newCollection` has decreased by 1.
 \vdash `newCollection.size()@P2.size() - 1 == collection.size() - 1`
 \vdash `newCollection.size()@P2.size() == collection.size()`
 \vdash [1.11]

QED

[2.11] \vdash (\forall `Object o`; `newCollection.contains(o)`; `collection.contains(o)`)
 \vdash (\forall `Object o`; `newCollection.nbOccurrences(o) > 0`;
`collection.nbOccurrences(o) > 0`)
 \vdash (\forall `Object o`; `newCollection.nbOccurrences(o)@P2 > 0`;
`collection.nbOccurrences(o)@P1 > 0`)
 \vdash (\forall `Object o`; `newCollection.nbOccurrences(o)@P2 > 0`;
`collection.nbOccurrences(o)@P1 > 0`)
 \vdash From a redundant postcondition of the `remove()` method we know that
`nbOccurrences(o)` can only decrease.
 \vdash (\forall `Object o`; `newCollection.nbOccurrences(o)@P1 > 0`;
`collection.nbOccurrences(o)@P1 > 0`)
 \vdash (\forall `Object o`; `newCollection.contains(o)@P1`;
`collection.contains(o)@P1`)
 \vdash [1.7]

QED

2.3.9 Program point P₃

Preconditions

The preconditions for program point P₃ are:

[3.a] `p erson != null`

[3.b] `o ther != null`

[3.c] `n ewCollection != null`

[3.d] `! newCollection.contains(null)`

[3.e] (\forall `Object o`; `newCollection.contains(o)`; `o instanceof Person`);

Proofs

[3.a] \dashv [2.1]

QED

[3.b] \dashv [2.4] && [2.7]

QED

[3.c] \dashv [2.6]

QED

[3.d] \dashv [2.9]

QED

[3.e] \dashv [2.10]

QED

Assertions

The assertions for program point P_3 are:

[3.1] $\text{collection.size()} > 0 \Rightarrow \text{result} ==$
 $(\exists \text{ Person } p2; \text{collection.contains}(p2);$
 $\text{person.age()} == p2.age() + 1)$

Proof

Since a boolean expression is either true or false, and $\text{other} \neq \text{null}$ because it is an element of collection , we know:

$(\forall \text{ Object } o; ! \text{other.equals}(o) \parallel \text{other.equals}(o)) \dashv \text{true}$

[3.1] \dashv $\text{collection.size()} > 0 \Rightarrow \text{result} ==$
 $(\exists \text{ Person } p2; \text{collection.contains}(p2);$
 $\text{person.age()} == p2.age() + 1)$
Using the previous assertion, we can split this in the cases
!other.equals(o) and other.equals(o)

- † `collection.size() > 0 ⇒ result ==`
`(\exists Person p2;`
`collection.contains(p2) && !other.equals(p2);`
`person.age() == p2.age() + 1) ||`
`(\exists Person p2;`
`collection.contains(p2) && other.equals(p2);`
`person.age() == p2.age() + 1)`
using the postconditions en invariants of Collection, we can change this into:
- † `collection.size() > 0 ⇒ result ==`
`(\exists Person p2;`
`collection.contains(p2) && !other.equals(p2);`
`person.age() == p2.age() + 1) ||`
`(\exists Person p2;`
`collection.contains(other) && other.equals(p2);`
`person.age() == p2.age() + 1)`
we know from [2.7] that `collection.contains(other)` is true.
- † `collection.size() > 0 ⇒ result ==`
`(\exists Person p2;`
`collection.contains(p2) && !other.equals(p2);`
`person.age() == p2.age() + 1) ||`
`(\exists Person p2;`
`other.equals(p2);`
`person.age() == p2.age() + 1)`
from the invariants of class Person, we know that since other and p2 are equal, their age must be the same.
- † `collection.size() > 0 ⇒ result ==`
`(\exists Person p2;`
`collection.contains(p2) && !other.equals(p2);`
`person.age() == p2.age() + 1) ||`
`(\exists Person p2;`
`other.equals(p2);`
`person.age() == other.age() + 1)`
since `other.equals(other)`, the exists quantifier can be removed.
- † `collection.size() > 0 ⇒ result ==`
`(\exists Person p2;`
`collection.contains(p2) && !other.equals(p2);`
`person.age() == p2.age() + 1) ||`
`person.age() == other.age() + 1)`
We now add the exists clause we've transformed. This is valid since $p || p == p$

- † collection.size() > 0 ⇒ result ==
 (∃ Person p2;
 collection.contains(p2) && !other.equals(p2);
 person.age() == p2.age() + 1) ||
 (∃ Person p2;
 collection.contains(p2) && other.equals(p2);
 person.age() == p2.age() + 1) ||
 person.age() == other.age() + 1)
Like we've split the initial exists quantifier into two quantifiers, we now combine the two quantifiers back to the original one.
- † collection.size() > 0 ⇒ result ==
 (∃ Person p2;
 collection.contains(p2);
 person.age() == p2.age() + 1) ||
 (person.age() == other.age() + 1)
Because we may strengthen the condition for the exists quantifier, using [2.11], we can say:
- † collection.size() > 0 ⇒ result ==
 (∃ Person p2;
 newCollection.contains(p2);
 person.age() == p2.age() + 1) ||
 (person.age() == other.age() + 1)
The quantifier is the result of the oneYearOlderThanSome method applied to person and newCollection.
- † line 14, line 18 and line 19

QED

2.3.10 Program point P_n

Assertions

The assertion for program point P_n is:

[n.1] \backslash result == (∃ Person p2; collection.contains(p2);
 person.age() == p2.age() + 1)

Proof

- [n.1] † \backslash result == (∃ Person p2; collection.contains(p2);
 person.age() == p2.age() + 1)
From line 21 we know that result is returned.
- † result == (∃ Person p2; collection.contains(p2);
 person.age() == p2.age() + 1)

```

+ ((collection.size() > 0) &&
  result ==(\exists Person p2; collection.contains(p2);
            person.age() == p2.age() + 1)) ||
  ((collection.size() == 0) &&
  result ==(\exists Person p2; collection.contains(p2);
            person.age() == p2.age() + 1)) &&
+ [3.1] ||
  ((collection.size() == 0) &&
  result ==(\exists Person p2; collection.contains(p2);
            person.age() == p2.age() + 1)) &&
  If collection is empty, collection.contains(p2) will always be false.
+ [3.1] ||
  ((collection.size() == 0) &&
  result ==(\exists Person p2; false;
            person.age() == p2.age() + 1)) &&
  Therefore, the exists quantifier will always be false.
+ [3.1] ||
  ((collection.size() == 0) && result ==false
+ [3.1] ||
  line 13, line 14 and line 21
+ true

```

QED

To prove that the method will end, we note that if `second.size() == 0`, no method invocation or loops occur, so that case ends in a finite amount of time. If `collection.size() > 0`, we see that the size of `newCollection` is one less than the size of `collection`. Because `newCollection` is given as first parameter to the recursive method invocation, the method will always come to the point where `collection.size() == 0`, and so the method will end.

Quod erat demonstrandum

2.4 Proof of correctness for an implementation using the ForAll and Exists classes.

Listing 9 shows an implementation of the method using the ForAll and Exists classes. We'll first prove the correctness of the `allOneYearOlder` method. Then we'll prove the correctness of both the criterion methods.

Proof of correctness of the `allOneYearOlder()` method

2.4.1 Program point P_0

The assertions known to be true at program point P_0 are the preconditions of the `allOneYearOlder()` method.

Listing 9: Implementation of the oneYearOlderThanSome method.

```

1  public boolean allOneYearOlder(
2      final Collection first , final Collection second) { //P0
3      return new ForAll() { //A
4          /*@
5              @ also public behavior
6              @
7              @ post \result == first.contains(element);
8              @
9              @ public model boolean isValidElement(Object element);
10             @*/
11
12             /*@
13                 @ also public behavior
14                 @
15                 @ post \result == (\exists Person p2;
16                 @                 second.contains(p2);
17                 @                 ((Person)element1).age() ==
18                 @                 p2.age() + 1));
19                 @*/
20             public boolean criterion(final Object element1) { //P1
21                 return new Exists() { //E
22                     /*@
23                         @ also public behavior
24                         @
25                         @ post \result == second.contains(element);
26                         @
27                         @ public model boolean isValidElement(Object element);
28                         @*/
29
30                     /*@
31                         @ also public behavior
32                         @
33                         @ post \result ==
34                         @ (((Person)element1).age() ==
35                         @ ((Person)element2).age() + 1);
36                         @*/
37                     public boolean criterion(final Object element2) { //P3
38                         return ((Person)element1).age() ==
39                         ((Person)element2).age() + 1; //P4
40                     }
41                 }.in(second); //P2
42             }
43         }.in( first ); //Pn
44     }

```

[0.1] first != null

[0.2] (\forall Object o; first.contains(o); o instanceof Person)

[0.3] ! first.contains(null)

[0.4] second != null

[0.5] (\forall Object o; second.contains(o); o instanceof Person)

[0.6] ! second.contains(null)

2.4.2 Program point P_n

Preconditions

The preconditions for program point P_n are (a is the ForAll object):

[n.a] first != null

[n.b] (\forall Object o, first.contains(o); isValidElement(o))

Proof

[n.a] \dashv [0.1]

QED

[n.b] \dashv (\forall Object o, first.contains(o); isValidElement(o))

\dashv (\forall Object o; first.contains(o); first.contains(o))

\dashv true

QED

Assertions

The assertions for program point P_n are:

[n.1] first != null

[n.2] (\forall Object o; first.contains(o); o instanceof Person)

[n.3] ! first.contains(null)

[n.4] second != null

[n.5] (\forall Object o; second.contains(o); o instanceof Person)

[n.6] ! second.contains(null)

[n.7] \backslash result == (\forall Person p1; first.contains(p1);
 \exists Person p2; second.contains(p2);
p1.age() == p2.age() + 1))

Proofs

[n.1] \vdash [0.1]

QED

[n.2] \vdash [0.2]

QED

[n.3] \vdash [0.3]

QED

[n.4] \vdash [0.4]

QED

[n.5] \vdash [0.5]

QED

[n.6] \vdash [0.6]

QED

[n.7] \vdash \backslash result == (\backslash forall Person p1; first.contains(p1);
 (\backslash exists Person p2; second.contains(p2);
 p1.age() == p2.age() + 1))
 using [0.2].
 \vdash \backslash result == (\backslash forall Object o1; first.contains(o1);
 (\backslash exists Person p2; second.contains(p2);
 ((Person)o1).age() == p2.age() + 1))
 using the postcondition of the criterion() method of class A
 \vdash \backslash result == (\backslash forall Object o1; first.contains(o1); a.criterion(o))
 using the postcondition of the in() method of class ForAll
 \vdash true

QED

Quod erat demonstrandum

Proof of correctness of the criterion() method of class A

2.4.3 Program point P₁

Since class A and the allOneYearOlder method are pure, the assertions at program point P₀ are still true.

Assertions

[1.1] first != null

[1.2] (\forall Object o; first.contains(o); o instanceof Person)

[1.3] ! first.contains(null)

[1.4] second != null

[1.5] (\forall Object o; second.contains(o); o instanceof Person)

[1.6] ! second.contains(null)

[1.7] first.contains(element1)

Proofs

[1.7] *this is a postcondition of isValidElement at the level of A, which is the precondition for A.criterion().*

⊢ true

QED

2.4.4 Program point P₂

Preconditions

The preconditions for program point P₂ are (e is the Exists object):

[2.a] (\forall Object o; second.contains(o); isValidElement(o))

Proof

[2.a] ⊢ (\forall Object o; second.contains(o); isValidElement(o))

⊢ (\forall Object o; second.contains(o); second.contains(o))

⊢ true

QED

Assertions

[2.1] first.contains(element1)

[2.2] \result == (\exists Person p2; second.contains(p2);
((Person)element1).age() == p2.age() + 1))

[2.1] \vdash [1.7]

QED

[2.2] \vdash \result == (\exists Person p2; second.contains(p2);
((Person)element1).age() == p2.age() + 1))
using [1.5]

\vdash \result == (\exists Object o2; second.contains(o2);
((Person)element1).age() == ((Person)o2).age() + 1))
using the postcondition of the criterion() method of class E.

\vdash \result == (\exists Object o2; second.contains(o2); e.criterion(o2)
using the postcondition of the in() method of class Exists.

\vdash true

QED

Quod erat demonstrandum

Proof of correctness of the criterion() method of class E

2.4.5 Program point P₃

The assertions are the preconditions of the criterion() method of class E and the assertions from the context (E,A and allOneYearOlder are pure). Note that program point P₃ is reached before program point P₂.

Assertions

[3.1] first != null

[3.2] (\forall Object o; first.contains(o); o instanceof Person)

[3.3] ! first.contains(null)

[3.4] second != null

[3.5] (\forall Object o; second.contains(o); o instanceof Person)

[3.6] ! second.contains(null)

[3.7] first.contains(element1)

[3.8] second.contains(element2)

Proofs

[3.7] *This is an invariant from the context.*

⊢ true

QED

[3.8] *this is a postcondition of isValidElement at the level of E, which is the precondition for criterion().*

⊢ true

QED

2.4.6 Program point P₄

Preconditions

[4.a] element1 != null

[4.b] element2 != null

[4.c] element1 instanceof Person

[4.d] element2 instanceof Person

[4.e] ((Person)element2).age() <= Integer.MAX_VALUE - 1

Proofs

[4.a] ⊢ element1 != null
⊢ first.contains(element1)
⊢ [3.7]

QED

[4.b] ⊢ element2 != null
⊢ second.contains(element2)
⊢ [3.8]

QED

[4.c] \vdash element1 instanceof Person
 \vdash first.contains(element1)
 \vdash [3.7]

QED

[4.d] \vdash element2 instanceof Person
 \vdash second.contains(element2)
 \vdash [3.8]

QED

[4.e] \vdash ((Person)element2).age() \leq Integer.MAX_VALUE - 1
 \vdash ((Person)element2).age() \leq 2000
from the third type invariant of Person we know this is true.
 \vdash true

QED

Assertions

[4.1] \backslash result == (((Person)element1).age() == ((Person)element2).age() + 1)

[4.1] \vdash lines 38 and 39

QED

Containment of the inner class.

Because the anonymous inner classes use the preconditions of the method they are defined in, they will not function correctly outside the scope of these methods. Therefore, we must ensure that no objects of these classes will be reachable from outside the methods they are defined in. Since both anonymous inner classes are pure, and their instances aren't assigned to any variable, we are sure they will not escape.

Quod erat demonstrandum

2.5 Proof of correctness for an implementation using the ForAll and Exists classes with a helper method.

Listing 10 shows an implementation of the method using the ForAll classes and a helper method. Listing 11 shows the implementation of the helper method using the Exists class. We'll first prove the correctness of the allOneYearOlder method. Then we'll prove the correctness the helper method.

Listing 10: Implementation of the allOneYearOlder method.

```
1 public boolean allOneYearOlder(  
2     final Collection first , final Collection second) { //P0  
3     return new ForAll() { //A  
4         /*@  
5         @ also public behavior  
6         @  
7         @ post \result == first.contains(element);  
8         @  
9         @ public model boolean isValidElement(Object element);  
10        @*/  
11  
12        /*@  
13        @ also public behavior  
14        @  
15        @ post \result == (\exists Person p2;  
16        @         second.contains(p2);  
17        @         ((Person)element1).age() ==  
18        @         p2.age() + 1));  
19        @*/  
20        public boolean criterion(final Object element1) { //P1  
21            return oneYearOlderThanSome(element1,second); //P2  
22        }.in( first ); //Pn  
23    }
```

Proof of correctness of the allOneYearOlder() method

2.5.1 Program point P₀

The assertions known to be true at program point P₀ are the preconditions of the allOneYearOlder() method.

[0.1] first != null

[0.2] (\forall Object o; first.contains(o); o instanceof Person)

[0.3] ! first.contains(null)

Listing 11: Implementation of the oneYearOlderThanSome method.

```

1  /*@
2   @ public behavior
3   @
4   @ pre person != null;
5   @ pre collection != null;
6   @ pre (\forall Object o; collection .contains(o);
7   @       o instanceof Person);
8   @ pre ! collection .contains(null);
9   @
10  @ post \result == (\exists Person p2;
11  @           collection .contains(p2);
12  @           person.age() == p2.age() + 1));
13  @*/
14  public boolean oneYearOlderThanSome(
15      Person person, final Collection collection) { //P0
16      return new Exists() { //E
17          /*@
18           @ also public behavior
19           @
20           @ post \result == collection.contains(element);
21           @
22           @ public model boolean isValidElement(Object element);
23           @*/
24
25          /*@
26           @ also public behavior
27           @
28           @ post \result ==
29           @ (person.age() ==
30           @ ((Person)element2).age() + 1);
31           @*/
32          public boolean criterion(final Object element2) { //P1
33              return person.age() ==
34              ((Person)element2).age() + 1; //P2
35          }
36      }.in( collection ); //Pn
37  }

```

[0.4] second != null

[0.5] (\forall Object o; second.contains(o); o instanceof Person)

[0.6] ! second.contains(null)

2.5.2 Program point P_n

Preconditions

The preconditions for program point P_n are (a is the ForAll object):

[n.a] first != null

[n.b] (\forall Object o, first.contains(o); isValidElement(o))

Proof

[n.a] \vdash [0.1]

QED

[n.b] \vdash (\forall Object o, first.contains(o); first.contains(o))
 \vdash true

QED

Assertions

The assertions for program point P_n are:

[n.1] first != null

[n.2] (\forall Object o; first.contains(o); o instanceof Person)

[n.3] ! first.contains(null)

[n.4] second != null

[n.5] (\forall Object o; second.contains(o); o instanceof Person)

[n.6] ! second.contains(null)

[n.7] \backslash result == (\forall Person p1; first.contains(p1);
 \backslash exists Person p2; second.contains(p2);
p1.age() == p2.age() + 1))

Proofs

[n.1] \vdash [0.1]

QED

[n.2] \vdash [0.2]

QED

[n.3] \vdash [0.3]

QED

[n.4] \vdash [0.4]

QED

[n.5] \vdash [0.5]

QED

[n.6] \vdash [0.6]

QED

[n.7] \vdash \backslash result == (\backslash forall Person p1; first.contains(p1);
 (\backslash exists Person p2; second.contains(p2);
 p1.age() == p2.age() + 1))
 using [0.2].
 \vdash \backslash result == (\backslash forall Object o1; first.contains(o1);
 (\backslash exists Person p2; second.contains(p2);
 ((Person)o1).age() == p2.age() + 1))
 using the postcondition of the criterion() method of class A
 \vdash \backslash result == (\backslash forall Object o1; first.contains(o1); a.criterion(o1))
 using the postcondition of the in() method of class ForAll
 \vdash true

QED

Quod erat demonstrandum

Proof of correctness of the criterion() method of class A

2.5.3 Program point P₁

Since class A and the allOneYearOlder() method are pure, the assertions at program point P₀ are still true.

Assertions

[1.1] first != null

[1.2] (\forall Object o; first.contains(o); o instanceof Person)

[1.3] ! first.contains(null)

[1.4] second != null

[1.5] (\forall Object o; second.contains(o); o instanceof Person)

[1.6] ! second.contains(null)

[1.7] first.contains(element1)

Proof

[1.7] *this is a postcondition of isValidElement at the level of A, which is the precondition for A.criterion().*

⊢ true

QED

2.5.4 Program point P₂

Preconditions

The preconditions for program point P₂ are:

[2.a] element1 != null

[2.b] element1 instanceof Person

[2.c] second != null

[2.d] (\forall Object o; second.contains(o); o instanceof Person)

[2.e] ! second.contains(null)

Proof

[2.a] \vdash element1 != null
 using [1.3]
 \vdash first.contains(element1)
 \vdash [1.7]

QED

[2.b] \vdash element1 instanceof Person
 using [1.2]
 \vdash first.contains(element1)
 \vdash [1.7]

QED

[2.c] \vdash [1.4]

QED

[2.d] \vdash [1.5]

QED

[2.e] \vdash [1.6]

QED

Assertions

[2.1] first.contains(element1)

[2.2] \backslash result == (\backslash exists Person p2; second.contains(p2);
 ((Person)element1).age() == p2.age() + 1))

[2.1] \vdash [1.7]

QED

[2.2] \vdash \backslash result == (\backslash exists Person p2; second.contains(p2);
 ((Person)element1).age() == p2.age() + 1))
 this is the postcondition of oneYearOlderThanSome()
 \vdash line 22

QED

Quod erat demonstrandum

Proof of correctness of the oneYearOlderThanSome() method.

2.5.5 Program point P_0

The assertions known to be true at program point P_0 are the preconditions of the oneYearOlderThanSome() method.

[0.1] person != null

[0.2] collection != null

[0.3] (\forall Object o; collection.contains(o); o instanceof Person)

[0.4] ! collection.contains(null)

2.5.6 Program point P_n

Preconditions

The preconditions for program point P_n are (e is the Exists object):

[n.a] collection != null

[n.b] (\forall Object o, collection.contains(o); isValidElement(o))

Proof

[n.a] \vdash [0.2]

QED

[n.b] \vdash (\forall Object o, collection.contains(o); isValidElement(o))

\vdash postcondition of isValidElement at the level of E.

\vdash (\forall Object o; collection.contains(o); collection.contains(o))

\vdash true

QED

Assertions

The assertions for program point P_n are:

[n.1] person != null

[n.2] collection != null

[n.3] (\forall Object o; collection.contains(o); o instanceof Person)

[n.4] ! collection.contains(null)

[n.5] \backslash result == (\exists Person p2; collection.contains(p2);
person.age() == p2.age() + 1)

Proofs

[n.1] \vdash [0.1]

QED

[n.2] \vdash [0.2]

QED

[n.3] \vdash [0.3]

QED

[n.4] \vdash [0.4]

QED

[n.5] \vdash \backslash result == (\backslash exists Person p2; collection.contains(p2);
person.age() == p2.age() + 1))

Using [0.3].

\vdash \backslash result == (\backslash exists Object o2; collection.contains(o2);
person.age() == ((Person)o2).age() + 1))

Using the postcondition of the criterion() method of class E.

\vdash \backslash result == (\backslash forall Object o1; first.contains(o1); e.criterion(o))

Using the postcondition of the in() method of class Exists.

\vdash true

QED

Quod erat demonstrandum

Proof of correctness of the criterion() method of class E

2.5.7 Program point P₁

Since class E and the oneYearOlderThanSome() method are pure, the assertions at program point P₀ are still true.

Assertions

[1.1] person != null

[1.2] collection != null

[1.3] (\backslash forall Object o; collection.contains(o); o instanceof Person)

[1.4] ! collection.contains(null)

[1.5] collection.contains(element2)

Proof

[1.5] *this is a postcondition of isValidElement at the level of E, which is the precondition for E.criterion().*
 \vdash true

QED

2.5.8 Program point P₂

Preconditions

The preconditions for program point P₂ are:

[2.a] person != null

[2.b] element2 != null

[2.c] element2 instanceof Person

[2.d] ((Person)element2).age() <= Integer.MAXVALUE - 1

Proof

[2.a] \vdash [1.1]

QED

[2.b] \vdash element2 != null
 using [1.4]
 \vdash collection.contains(element2)
 \vdash [1.5]

QED

[2.c] \vdash element2 instanceof Person
 using [1.3]
 \vdash collection.contains(element2)
 \vdash [1.5]

QED

[2.d] \vdash ((Person)element2).age() <= Integer.MAXVALUE - 1
 \vdash ((Person)element2).age() <= 2000
 This is the third type invariant of Person.
 \vdash true

QED

Assertions

[2.1] `collection.contains(element2)`

[2.2] `\result == (person.age() == ((Person)element2).age() + 1)`

[2.1] \vdash [1.5]

QED

[2.2] \vdash `\result == (person.age() == ((Person)element2).age() + 1)`
 \vdash lines 33 && 34

QED

Quod erat demonstrandum

3 Conclusion

As a conclusion, we'll look at the lengths of the different proofs:

nested for-loop :	45 pages
use of a helper method and <i>external</i> iterators	26 pages
recursive implementation using a helper method	22 pages
implementation using internal iterators	6 pages
implementation using internal iterators and a helper method	8 pages

It is interesting to see that with internal iterators, using a helper method didn't shorten the length of proof. The overhead of the method call (preconditions need to be verified) didn't outweigh the extra information that has to be taken along when not using a helper method. On the other hand, the version with the helper method has an extra method (the inner iteration) which might be useful.

It is clear that internal iterators shorten the proof dramatically. The proof for the implementation with ForAll and Exists in a single method is 3 times shorter than the shortest proof without internal iterators.

References

- [1] E. W. DIJKSTRA, *Notes on structured programming*, tech. rep., Technological University Eindhoven, 1970.
- [2] J. DOCKX, M. VAN DOOREN, AND E. STEEGSMANS, *Dijkstra's dream. internal iterators as software theorems*. submitted to ECOOP 2002.
- [3] E. GAMMA, R. HELM, R. JOHNSON, AND J. VLISSIDES, *Design patterns*, Addison-Wesley, 1999.