Dafny klasar og fastayrðingar gagna

Snorri Agnarsson

IntQueue – Skilgreining

```
trait IntQueue
{
    ghost var ghostseq: seq<int>;
    ghost var Repr: set<object>;

    predicate Valid()
        reads this, Repr;

    predicate method IsEmpty()
        reads this, Repr;
        requires Valid();
        ensures IsEmpty() <==> ghostseq==[];

    method Put( x: int )
        modifies this, Repr;
        requires Valid();
        ensures Valid() && fresh(Repr-old(Repr));
        ensures ghostseq == old(ghostseq)+[x];

    method Get() returns ( x: int )
        modifies this, Repr;
        requires Valid();
        requires ghostseq != [];
        ensures Valid() && fresh(Repr-old(Repr));
        ensures ghostseq == old(ghostseq[1..]);
        ensures x == old(ghostseq[0]);
}
```

IntQueueArray – Útfærsla (1/2)

```
class IntQueueArray extends IntQueue
{
    var a: array<int>;
    var front: int;
    var size: int;

    predicate Valid()
        reads this, Repr;
    {
        Repr == {this,a} &&
        a.Length > 0 &&
        0 <= size <= a.Length &&
        0 <= front < a.Length &&
        |ghostseq| == size &&
        if front+size <= a.Length then
            ghostseq == a[front..front+size]
        else
            ghostseq == a[front..]+a[..front+size-a.Length]
    }
```

```
    constructor()
        ensures Valid() && fresh(Repr-{this});
        ensures ghostseq == [];
    {
        a := new int[1];
        front := 0;
        size := 0;
        Repr := {this,a};
        ghostseq := [];
    }

    predicate method IsEmpty()
        reads this, Repr;
        requires Valid();
        ensures IsEmpty() <==> ghostseq==[];
    {
        size == 0
    }
```

IntQueueArray – Útfærsla (2/2)

```
method Put( x: int )
    modifies this, Repr;
    requires Valid();
    ensures Valid() && fresh(Repr-old(Repr));
    ensures ghostseq == old(ghostseq)+[x];
{
    if size == a.Length
    {
        var newa := new int[2*a.Length];
        var i := 0;
        var j := front;
        while i < size && j != a.Length
            decreases size-i;
            invariant 0 <= j <= a.Length;
            invariant 0 <= i <= |ghostseq| == size <= a.Length < newa.Length;
            invariant j == front+i;
            invariant newa[..i] == ghostseq[..i];
            invariant ghostseq == old(ghostseq);
            invariant Valid();
        {
            newa[i] := a[j];
            i := i+1;
            j := j+1;
        }
        if j == a.Length
        {
            j := 0;
            assert j == front+i-a.Length;
            while i < size
                decreases size-i;
                invariant 0 <= j <= size <= a.Length;
                invariant 0 <= i <= |ghostseq| == size <= a.Length < newa.Length;
                invariant j == front+i-a.Length;
                invariant newa[..i] == ghostseq[..i];
                invariant ghostseq == old(ghostseq);
                invariant Valid();
            {
                newa[i] := a[j];
                i := i+1;
                j := j+1;
            }
        }
        a := newa;
        Repr := {this,a};
        front := 0;
    }
    if front+size >= a.Length
    {
        a[front+size-a.Length] := x;
    }
    else
    {
        a[front+size] := x;
    }
    size := size+1;
    ghostseq := ghostseq+[x];
  }
}
```

IntMultiset – Skilgreining

```
trait IntMultiset
{
    ghost var ghostbag: multiset<int>;
    ghost var Repr: set<object>;

    predicate Valid()
        reads this, Repr;

    method Add( x: int )
        modifies this, Repr;
        requires Valid();
        ensures Valid();
        ensures ghostbag == old(ghostbag)+multiset{x};
        ensures fresh(Repr-old(Repr));

    method Remove() returns( x: int )
        modifies this, Repr;
        requires Valid();
        requires |ghostbag| != 0;
        ensures Valid();
        ensures x in old(ghostbag);
        ensures ghostbag == old(ghostbag)-multiset{x};
        ensures fresh(Repr-old(Repr));

    predicate method IsEmpty()
        reads this, Repr;
        requires Valid();
        ensures IsEmpty() <==> |ghostbag| == 0;
}
```

IntMultisetArray – Útfærsla (1/2)

```
class IntMultisetArray extends IntMultiset
{
    var a: array<int>;
    var size: int;

    predicate Valid()
        reads this, Repr;
    {
        Repr == {this,a} &&
        0 <= size <= a.Length &&
        multiset(a[..size]) == ghostbag &&
        a.Length > 0
    }
```

```
    constructor()
        ensures Valid() && fresh(Repr-{this});
        ensures |ghostbag| == 0;
    {
        a := new int[100];
        size := 0;
        Repr := {this,a};
        ghostbag := multiset{};
    }
```

IntMultisetArray – Útfærsla (2/2)

```
predicate method IsEmpty()
    reads this, Repr;
    requires Valid();
    ensures IsEmpty() <==> |ghostbag| == 0;
{
    size == 0
}
```

```
method Add( x: int )
    modifies this, Repr;
    requires Valid();
    ensures Valid();
    ensures fresh(Repr-old(Repr));
    ensures ghostbag == old(ghostbag)+multiset{x};
{
    if size == a.Length
    {
        var newa := new int[2*a.Length];
        assert newa.Length > a.Length;
        var i := 0;
        while i != size
            decreases size-i;
            invariant 0 <= i <= size <= a.Length < newa.Length;
            invariant newa[..i] == a[..i];
            invariant ghostbag == old(ghostbag);
            invariant Valid();
        {
            newa[i] := a[i];
            i := i+1;
        }
        a := newa;
        Repr := {this,a};
    }
    a[size] := x;
    size := size+1;
    ghostbag := ghostbag+multiset{x};
}
```

IntMinPriQueue – Skilgreining

```
trait IntMinPriQueue
{
    ghost var ghostbag: multiset<int>;
    ghost var Repr: set<object>;

    predicate Valid()
        reads this, Repr;

    predicate method IsEmpty()
        reads this, Repr;
        requires Valid();
        ensures IsEmpty() <==> |ghostbag| == 0;

    function Contents(): multiset<int>
        reads this;
    {
        ghostbag
    }

    method Add( x: int )
        modifies this, Repr;
        requires Valid();
        ensures Valid() && fresh(Repr-old(Repr));
        ensures ghostbag == old(ghostbag)+multiset{x};

    method RemoveMin() returns ( x: int )
        modifies this, Repr;
        requires Valid();
        requires |ghostbag| != 0;
        ensures Valid() && fresh(Repr-old(Repr));
        ensures x in old(ghostbag);
        ensures ghostbag == old(ghostbag)-multiset{x};
        ensures forall z | z in ghostbag :: x <= z;
}
```

IntMinPriQueueHeap – Útfærsla (1/2)

```
class IntMinPriQueueHeap extends IntMinPriQueue
{
    var a: array<int>;
    var n: int;

    predicate Valid()
        reads this, Repr;
    {
        Repr == {a,this} &&
        a.Length >= 2 &&
        0 <= n <= a.Length &&
        IsMinHeap(a[..],0,n) &&
        multiset(a[..n]) == ghostbag
    }
```

```
    constructor()
        ensures Valid() && fresh(Repr-{this});
        ensures |ghostbag| == 0;
    {
        n := 0;
        ghostbag := multiset{};
        a := new int[2];
        Repr := {a,this};
    }

    predicate method IsEmpty()
        reads this, Repr;
        requires Valid();
        ensures IsEmpty() <==> |ghostbag| == 0;
    {
        n == 0
    }
```

IntMinPriQueueHeap – Útfærsla (2/2)

```
method Add( x: int )
    modifies this, Repr;
    requires Valid();
    ensures Valid() && fresh(Repr-old(Repr));
    ensures ghostbag == old(ghostbag)+multiset{x};
{
    if n == a.Length
    {
        var newa := new int[2*a.Length];
        var i := 0;
        while i < n
            decreases n-i;
            invariant 0 <= i <= |ghostbag| ==
                  n <= a.Length < newa.Length;
            invariant a == old(a);
            invariant a[..] == old(a[..]);
            invariant newa[..i] == a[..i];
            invariant ghostbag == old(ghostbag);
            invariant Valid();
        {
            newa[i] := a[i];
            i := i+1;
        }
        a := newa;
        Repr := {a,this};
    }
    a[n] := x;
    RollUpMinHeap(a,n);
    ghostbag := ghostbag+multiset{x};
    n := n+1;
}
```

```
    method RemoveMin() returns ( x: int )
        modifies this, Repr;
        requires Valid();
        requires |ghostbag| != 0;
        ensures Valid() && fresh(Repr-old(Repr));
        ensures x in old(ghostbag);
        ensures ghostbag == old(ghostbag)-multiset{x};
        ensures forall z | z in ghostbag :: x <= z;
    {
        ZeroHasMin(a[..],n);
        x := a[0];
        n := n-1;
        ghostbag := ghostbag-multiset{x};
        if n == 0 { return; }
        a[0] := a[n];
        RollDownMinHeap(a,0,n);
    }
}
```

Dafny classes and data invariants

Snorri Agnarsson

IntQueue – Definition

```
trait IntQueue
{
    ghost var ghostseq: seq<int>;
    ghost var Repr: set<object>;

    predicate Valid()
        reads this, Repr;

    predicate method IsEmpty()
        reads this, Repr;
        requires Valid();
        ensures IsEmpty() <==> ghostseq==[];

    method Put( x: int )
        modifies this, Repr;
        requires Valid();
        ensures Valid() && fresh(Repr-old(Repr));
        ensures ghostseq == old(ghostseq)+[x];

    method Get() returns ( x: int )
        modifies this, Repr;
        requires Valid();
        requires ghostseq != [];
        ensures Valid() && fresh(Repr-old(Repr));
        ensures ghostseq == old(ghostseq[1..]);
        ensures x == old(ghostseq[0]);
}
```

IntQueueArray – Implementation (1/2)

```
class IntQueueArray extends IntQueue
{
    var a: array<int>;
    var front: int;
    var size: int;

    predicate Valid()
        reads this, Repr;
    {
        Repr == {this,a} &&
        a.Length > 0 &&
        0 <= size <= a.Length &&
        0 <= front < a.Length &&
        |ghostseq| == size &&
        if front+size <= a.Length then
            ghostseq == a[front..front+size]
        else
            ghostseq == a[front..]+a[..front+size-a.Length]
    }

    constructor()
        ensures Valid() && fresh(Repr-{this});
        ensures ghostseq == [];
    {
        a := new int[1];
        front := 0;
        size := 0;
        Repr := {this,a};
        ghostseq := [];
    }

    predicate method IsEmpty()
        reads this, Repr;
        requires Valid();
        ensures IsEmpty() <==> ghostseq==[];
    {
        size == 0
    }

    method Get() returns ( x: int )
        modifies this, Repr;
        requires Valid();
        requires ghostseq != [];
        ensures Valid() && fresh(Repr-old(Repr));
        ensures ghostseq == old(ghostseq[1..]);
        ensures x == old(ghostseq[0]);
    {
        size := size-1;
        x := a[front];
        front := front+1;
        ghostseq := ghostseq[1..];
        if front == a.Length { front := 0; }
    }
}
```

IntQueueArray – Implementation (2/2)

```
method Put( x: int )
    modifies this, Repr;
    requires Valid();
    ensures Valid() && fresh(Repr-old(Repr));
    ensures ghostseq == old(ghostseq)+[x];
{
    if size == a.Length
    {
        var newa := new int[2*a.Length];
        var i := 0;
        var j := front;
        while i < size && j != a.Length
            decreases size-i;
            invariant 0 <= j <= a.Length;
            invariant 0 <= i <= |ghostseq| == size <= a.Length < newa.Length;
            invariant j == front+i;
            invariant newa[..i] == ghostseq[..i];
            invariant ghostseq == old(ghostseq);
            invariant Valid();
        {
            newa[i] := a[j];
            i := i+1;
            j := j+1;
        }
        if j == a.Length
        {
            j := 0;
            assert j == front+i-a.Length;
            while i < size
                decreases size-i;
                invariant 0 <= j <= size <= a.Length;
                invariant 0 <= i <= |ghostseq| == size <= a.Length < newa.Length;
                invariant j == front+i-a.Length;
                invariant newa[..i] == ghostseq[..i];
                invariant ghostseq == old(ghostseq);
                invariant Valid();
            {
                newa[i] := a[j];
                i := i+1;
                j := j+1;
            }
        }
        a := newa;
        Repr := {this,a};
        front := 0;
    }
    if front+size >= a.Length
    {
        a[front+size-a.Length] := x;
    }
    else
    {
        a[front+size] := x;
    }
    size := size+1;
    ghostseq := ghostseq+[x];
  }
}
```

IntMultiset – Definition

```
trait IntMultiset
{
    ghost var ghostbag: multiset<int>;
    ghost var Repr: set<object>;

    predicate Valid()
        reads this, Repr;

    method Add( x: int )
        modifies this, Repr;
        requires Valid();
        ensures Valid();
        ensures ghostbag == old(ghostbag)+multiset{x};
        ensures fresh(Repr-old(Repr));

    method Remove() returns( x: int )
        modifies this, Repr;
        requires Valid();
        requires |ghostbag| != 0;
        ensures Valid();
        ensures x in old(ghostbag);
        ensures ghostbag == old(ghostbag)-multiset{x};
        ensures fresh(Repr-old(Repr));

    predicate method IsEmpty()
        reads this, Repr;
        requires Valid();
        ensures IsEmpty() <==> |ghostbag| == 0;
}
```

IntMultisetArray – Implementation (1/2)

```
class IntMultisetArray extends IntMultiset
{
    var a: array<int>;
    var size: int;

    predicate Valid()
        reads this, Repr;
    {
        Repr == {this,a} &&
        0 <= size <= a.Length &&
        multiset(a[..size]) == ghostbag &&
        a.Length > 0
    }
```

```
    constructor()
        ensures Valid() && fresh(Repr-{this});
        ensures |ghostbag| == 0;
    {
        a := new int[100];
        size := 0;
        Repr := {this,a};
        ghostbag := multiset{};
    }
```

IntMultisetArray – Implementation (2/2)

```
predicate method IsEmpty()
    reads this, Repr;
    requires Valid();
    ensures IsEmpty() <==> |ghostbag| == 0;
{
    size == 0
}
```

```
method Remove() returns ( x: int )
    modifies this, Repr;
    requires Valid();
    requires |ghostbag| != 0;
    ensures Valid();
    ensures fresh(Repr-old(Repr));
    ensures x in old(ghostbag);
    ensures ghostbag == old(ghostbag)-multiset{x};
{
    size := size-1;
    x := a[size];
    ghostbag := ghostbag-multiset{x};
}
```

```
method Add( x: int )
    modifies this, Repr;
    requires Valid();
    ensures Valid();
    ensures fresh(Repr-old(Repr));
    ensures ghostbag == old(ghostbag)+multiset{x};
{
    if size == a.Length
    {
        var newa := new int[2*a.Length];
        assert newa.Length > a.Length;
        var i := 0;
        while i != size
            decreases size-i;
            invariant 0 <= i <= size <= a.Length < newa.Length;
            invariant newa[..i] == a[..i];
            invariant ghostbag == old(ghostbag);
            invariant Valid();
        {
            newa[i] := a[i];
            i := i+1;
        }
        a := newa;
        Repr := {this,a};
    }
    a[size] := x;
    size := size+1;
    ghostbag := ghostbag+multiset{x};
}
```

IntMinPriQueue – Definition

```
trait IntMinPriQueue
{
    ghost var ghostbag: multiset<int>;
    ghost var Repr: set<object>;

    predicate Valid()
        reads this, Repr;

    predicate method IsEmpty()
        reads this, Repr;
        requires Valid();
        ensures IsEmpty() <==> |ghostbag| == 0;

    function Contents(): multiset<int>
        reads this;
    {
        ghostbag
    }

    method Add( x: int )
        modifies this, Repr;
        requires Valid();
        ensures Valid() && fresh(Repr-old(Repr));
        ensures ghostbag == old(ghostbag)+multiset{x};

    method RemoveMin() returns ( x: int )
        modifies this, Repr;
        requires Valid();
        requires |ghostbag| != 0;
        ensures Valid() && fresh(Repr-old(Repr));
        ensures x in old(ghostbag);
        ensures ghostbag == old(ghostbag)-multiset{x};
        ensures forall z | z in ghostbag :: x <= z;
}
```

IntMinPriQueueHeap – Implementation (1/2)

```
class IntMinPriQueueHeap extends IntMinPriQueue
{
    var a: array<int>;
    var n: int;

    predicate Valid()
        reads this, Repr;
    {
        Repr == {a,this} &&
        a.Length >= 2 &&
        0 <= n <= a.Length &&
        IsMinHeap(a[..],0,n) &&
        multiset(a[..n]) == ghostbag
    }
```

```
    constructor()
        ensures Valid() && fresh(Repr-{this});
        ensures |ghostbag| == 0;
    {
        n := 0;
        ghostbag := multiset{};
        a := new int[2];
        Repr := {a,this};
    }

    predicate method IsEmpty()
        reads this, Repr;
        requires Valid();
        ensures IsEmpty() <==> |ghostbag| == 0;
    {
        n == 0
    }
```

IntMinPriQueueHeap – Implementation (2/2)

```
method Add( x: int )
    modifies this, Repr;
    requires Valid();
    ensures Valid() && fresh(Repr-old(Repr));
    ensures ghostbag == old(ghostbag)+multiset{x};
{
    if n == a.Length
    {
        var newa := new int[2*a.Length];
        var i := 0;
        while i < n
            decreases n-i;
            invariant 0 <= i <= |ghostbag| ==
                  n <= a.Length < newa.Length;
            invariant a == old(a);
            invariant a[..] == old(a[..]);
            invariant newa[..i] == a[..i];
            invariant ghostbag == old(ghostbag);
            invariant Valid();
        {
            newa[i] := a[i];
            i := i+1;
        }
        a := newa;
        Repr := {a,this};
    }
    a[n] := x;
    RollUpMinHeap(a,n);
    ghostbag := ghostbag+multiset{x};
    n := n+1;
}
```

```
    method RemoveMin() returns ( x: int )
        modifies this, Repr;
        requires Valid();
        requires |ghostbag| != 0;
        ensures Valid() && fresh(Repr-old(Repr));
        ensures x in old(ghostbag);
        ensures ghostbag == old(ghostbag)-multiset{x};
        ensures forall z | z in ghostbag :: x <= z;
    {
        ZeroHasMin(a[..],n);
        x := a[0];
        n := n-1;
        ghostbag := ghostbag-multiset{x};
        if n == 0 { return; }
        a[0] := a[n];
        RollDownMinHeap(a,0,n);
    }
}
```