TÖL304G Forritunarmál

Hópverkefni 9

Hjörvar Sigurðsson

Kóði:

Kóðinn er á mynda-formi svo að syntax helst og kóðinn sé eins auðlesinn og hægt er. Sé þörf á að eiga við kóðann má finna hann í viðauka 1 aftast í skýrslunni.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Design document
;;; ===============
;;;
;;; Exported
;;; --------
;;;
;;; Use:   val s = makeSet();
;;; Pre:   Nothing.
;;; Post:  s contains a new empty set of
;;;        values that are allowed as
;;;        arguments to the imported
;;;        function comp.
;;;
;;; Imported
;;; --------
;;;
;;; Use:  val c = comp(x,y);
;;; Pre:  x and y are values that are
;;;       allowed to be stored in the sets
;;;       implemented here.
;;; Post: c is an integer that is <0 if x
;;;       must precede y, >0 if y must
;;;       precede x, and ==0 if x and y
;;;       are equal.
;;; Note: comp should define an ordering on
;;;       the values allowed in the sets.
;;;       The ordering should ensure that
;;;       any finite set of values has a
;;;       least element.
;;;
;;; Use:  s.add(x);
;;; Pre:  s is a set that can contain x.
;;; Post: x has been added to s if it was
;;;       not already in s. If x was
;;;       was already in s then s is
;;;       unchanged.
;;;
```

```
;;;
;;; Use:  val e = s.isEmpty();
;;; Pre:  s is a set.
;;; Post: e contains true if s is empty,
;;;       false otherwise.
;;;
;;; Use:  val c = s.contains(x);
;;; Pre:  s is a set that can contain x.
;;; Post: c is true if s contains x, false
;;;       otherwise.
;;;
;;; Use:  val m = s.min();
;;; Pre:  s is a set, not empty.
;;; Post: m is the minimal value in s,
;;;       according to the imported
;;;       function comp.
;;;
;;; Use:  s.remove(x);
;;; Pre:  s is a set that can contain x.
;;; Post: If s contained x then x has
;;;       been removed from s, otherwise
;;;       s is unchanged.
;;;
;;; Use:  val r = s.mapReduce(op,f,u);
;;; Pre:  s is a set.
;;;       op is a binary function,
;;;       f is a unary function.
;;;       u is some value such that
;;;       the expression in the post-
;;;       condition can be computed.
;;; Post: The expression
;;;        u ! f(x1) ! f(x2) ! ... ! f(xN)
;;;       has been computed, where x!y
;;;       is equivalent to op(x,y) and
;;;       the computation is performed
;;;       from left to right, and the
;;;       values x1,x2,...,xN are all the
;;;       values in s in ascending order.
;;;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

"set.mmod" =
{{
makeSet = fun makeSet();
}}
*
!
{{
makeSet =
    obj()
    {
        var set;

        ;;; Data invariant:
        ;;;     set contains a set of integers x1, x2, ..., xN, represented by a
list of integers [x1, x2, ..., xN].

        msg add(x)
        {
            if (this.contains(x) == false)
            {
                set = x:set;
            }
        };

        msg isEmpty()
        {
            set == [];
        };
```

```
    msg contains(x)
    {
        if (set == [])
        {
            false;
        }
        else
        {
            val f = filterList(fun(z){z==x}, set);
            if (f == null)
            {
                false;
            }
            else
            {
                true;
            }
        }
    };

    msg min()
    {
        minHelper(head(set), set);
    };

    msg remove(x)
    {
        if (this.contains(x))
        {
            set = filterList(fun(z){z!=x}, set);
        }
    };

    msg mapReduce(op,f,u)
    {
        mapReduceHelper(op, f, u, set);
    };
};
```

```
;;; Helper functions.

;;; Use:     val m = minHelper(x, thisList);
;;; Pre:     x is an integer.
;;;          thisList is a set of integers x1, x2, ..., xN.
;;; Post:    Minimum number
minHelper =
    fun(x, thisList)
    {
        if (head(thisList) < x)
        {
            if (tail(thisList) == null)
            {
                x;
            }
            else
            {
                minHelper(head(thisList), tail(thisList));
            }
        }
        else
        {
            if (tail(thisList) == null)
            {
                x;
            }
            else
            {
                minHelper(x, tail(thisList));
            }
        }
    };
```

```
    ;;; Use:    mapReduceHelper(op, f, u, x);
    ;;; Pre:    op is a function 'c -> 'b -> 'c,
    ;;;         f is a function 'a -> 'b,
    ;;;         u is a value of type 'c,
    ;;;         x=x1, s2, ..., xN is a set of values of type 'a.
    ;;; Post:   u+f(x1)+f(x2)+...+f(xN), calculated from left to right,
    ;;;         where p+g = (op p q). This is a value of type 'c.
    mapReduceHelper =
        fun(op, f, u, x)
        {
            if (x==[])
            {
                u;
            }
            else
            {
                mapReduceHelper(op, f, op(u, f(head(x))), tail(x));
            }
        };
}}
*
BASIS
;
```

```
;;; A test program.
"testset.mexe" = main in
{{
main =
    fun()
    {
        try
        {
            var x = [1,9,2,8,3,7,4,6,5];
            val s = makeSet();  ;;; A set of integers
            while( x )
            {
                ;;; Loop invariant:
                ;;; s contains a subset of the set {1..9}.
                ;;; The list x contains exactly the rest of
                ;;; the set {1..9}.
                s.remove(head(x));  ;;; Should have no effect
                s.add(head(x));
                s.add(head(x));     ;;; Should have no effect
                x = tail(x);
            };
            writeln(s.isEmpty());   ;;; Should write false
            writeln(s.mapReduce(fun(x,y){x+y},fun(x){x},0));      ;;; Should write 4
5
            s.mapReduce(fun(x,y){[]},fun(x){write(x)},[]);        ;;; Should write 1
23456789
            writeln();
            while( !s.isEmpty() ) { s.remove(s.min()) };          ;;; Should empty t
he set
            writeln(s.isEmpty());   ;;; Should write true
        }
        catch(e)
        {
            printExceptionTrace(&e)
        }
    };
}}

*
"set.mmod"
*
{{
;;; Use:  val c = comp(x,y);
;;; Pre:  x and y are integers.
;;; Post: x is <0 if x<y, =0 if x==y, >0 if x>y.
comp =
    fun(x,y)
    {
        x<y && (return -1);
        y<x && (return 1);
        0
    };
}}
*
BASIS
;
```

```
;;; Another test program.
"testset2.mexe" = main in
{{
main =
    fun()
    {
        try
        {
            var x = [[1],[9],[2],[8],[3],[7],[4],[6],[5]];
            val s = makeSet();     ;;; A set of non-empty integer lists
            while( x )
            {
                ;;; Loop invariant:
                ;;; s contains a subset of the set {[1]..[9]}.
                ;;; The list x contains exactly the rest of
                ;;; the set {[1]..[9]}.
                s.remove(head(x));  ;;; Should have no effect
                s.add(head(x));
                s.add(head(x));     ;;; Should have no effect
                x = tail(x);
            };
            writeln(s.isEmpty());     ;;; Should write false
            writeln(s.mapReduce(fun(x,y){x+y},fun(x){head(x)},0));  ;;; Should write
 45
            s.mapReduce(fun(x,y){[]},fun(x){write(x)},[]);          ;;; Should write
 [1][2][3][4][5][6][7][8][9]
            writeln();
            while( !s.isEmpty() ) { s.remove(s.min()) };          ;;; Should empty t
he set
            writeln(s.isEmpty());     ;;; Should write true
        }
        catch(e)
        {
            printExceptionTrace(&e)
        }
    };
}}

*
"set.mmod"
*
{{
;;; Use:  val c = comp(x,y);
;;; Pre:  x and y are non-empty lists containing integers.
;;; Post: c is <0 if head(x)<head(y), =0 if head(x)==head(y),
;;;       >0 if head(x)>head(y).
comp =
    fun(x,y)
    {
        head(x)<head(y) && (return -1);
        head(y)<head(x) && (return 1);
        0
    };
}}
*
BASIS
;
```

Viðauki 1

<u>Kóði:</u>

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Design document
;;; ===============
;;;
;;; Exported
;;; --------
;;;
;;; Use:   val s = makeSet();
;;; Pre:   Nothing.
;;; Post:  s contains a new empty set of
;;;        values that are allowed as
;;;        arguments to the imported
;;;        function comp.
;;;
;;; Imported
;;; --------
;;;
;;; Use:  val c = comp(x,y);
;;; Pre:  x and y are values that are
;;;       allowed to be stored in the sets
;;;       implemented here.
;;; Post: c is an integer that is <0 if x
;;;       must precede y, >0 if y must
;;;       precede x, and ==0 if x and y
;;;       are equal.
;;; Note: comp should define an ordering on
```

```
;;;       the values allowed in the sets.
;;;       The ordering should ensure that
;;;       any finite set of values has a
;;;       least element.
;;;
;;; Use:  s.add(x);
;;; Pre:  s is a set that can contain x.
;;; Post: x has been added to s if it was
;;;       not already in s. If x was
;;;       was already in s then s is
;;;       unchanged.
;;;
;;; Use:  val e = s.isEmpty();
;;; Pre:  s is a set.
;;; Post: e contains true if s is empty,
;;;       false otherwise.
;;;
;;; Use:  val c = s.contains(x);
;;; Pre:  s is a set that can contain x.
;;; Post: c is true if s contains x, false
;;;       otherwise.
;;;
;;; Use:  val m = s.min();
;;; Pre:  s is a set, not empty.
;;; Post: m is the minimal value in s,
;;;       according to the imported
;;;       function comp.
;;;
;;; Use:  s.remove(x);
;;; Pre:  s is a set that can contain x.
```

```
;;; Post: If s contained x then x has
;;;       been removed from s, otherwise
;;;       s is unchanged.
;;;
;;; Use:  val r = s.mapReduce(op,f,u);
;;; Pre:  s is a set.
;;;       op is a binary function,
;;;       f is a unary function.
;;;       u is some value such that
;;;       the expression in the post-
;;;       condition can be computed.
;;; Post: The expression
;;;        u ! f(x1) ! f(x2) ! ... ! f(xN)
;;;       has been computed, where x!y
;;;       is equivalent to op(x,y) and
;;;       the computation is performed
;;;       from left to right, and the
;;;       values x1,x2,...,xN are all the
;;;       values in s in ascending order.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

"set.mmod" =
{{
makeSet = fun makeSet();
}}
*
!
{{
makeSet =
```

```
obj()
{
	var set;

	;;; Data invariant:
	;;;	set contains a set of integers x1, x2, ..., xN, represented by a list of integers [x1, x2, ..., xN].

	msg add(x)
	{
		if (this.contains(x) == false)
		{
			set = x:set;
		}
	};

	msg isEmpty()
	{
		set == [];
	};

	msg contains(x)
	{
		if (set == [])
		{
			false;
		}
		else
		{
			val f = filterList(fun(z){z==x}, set);
			if (f == null)
```

```
                {
                        false;
                }
                else
                {
                        true;
                }
        }
    };

    msg min()
    {
        minHelper(head(set), set);
    };

    msg remove(x)
    {
        if (this.contains(x))
        {
            set = filterList(fun(z){z!=x}, set);
        }
    };

    msg mapReduce(op,f,u)
    {
        mapReduceHelper(op, f, u, set);
    };
};

;;; Helper functions.
```

```
;;; Use:val m = minHelper(x, thisList);
;;;        Pre:    x is an integer.
;;;                        thisList is a set of integers x1, x2, ..., xN.
;;;        Post:   Minimum number
minHelper =
        fun(x, thisList)
        {
                if (head(thisList) < x)
                {
                        if (tail(thisList) == null)
                        {
                                x;
                        }
                        else
                        {
                                minHelper(head(thisList), tail(thisList));
                        }
                }
                else
                {
                        if (tail(thisList) == null)
                        {
                                x;
                        }
                        else
                        {
                                minHelper(x, tail(thisList));
                        }
                }
```

```
        };

    ;;; Use:mapReduceHelper(op, f, u, x);
    ;;; Pre: op is a function 'c -> 'b -> 'c,
    ;;;                 f is a function 'a -> 'b,
    ;;;                 u is a value of type 'c,
    ;;;                 x=x1, s2, ..., xN is a set of values of type 'a.
    ;;; Post:       u+f(x1)+f(x2)+...+f(xN), calculated from left to right,
    ;;;                 where p+g = (op p q). This is a value of type 'c.
    mapReduceHelper =
        fun(op, f, u, x)
        {
            if (x==[])
            {
                u;
            }
            else
            {
                mapReduceHelper(op, f, op(u, f(head(x))), tail(x));
            }
        };
}}
*
BASIS
;

;;; A test program.
"testset.mexe" = main in
{{
main =
```

```
fun()
{
	try
	{
		var x = [1,9,2,8,3,7,4,6,5];
		val s = makeSet();	;;; A set of integers
		while( x )
		{
			;;; Loop invariant:
			;;; s contains a subset of the set {1..9}.
			;;; The list x contains exactly the rest of
			;;; the set {1..9}.
			s.remove(head(x));  ;;; Should have no effect
			s.add(head(x));
			s.add(head(x));	;;; Should have no effect
			x = tail(x);
		};
		writeln(s.isEmpty());  ;;; Should write false
		writeln(s.mapReduce(fun(x,y){x+y},fun(x){x},0));	;;; Should write
45
		s.mapReduce(fun(x,y){[]},fun(x){write(x)},[]);	;;; Should write
123456789
		writeln();
		while( !s.isEmpty() ) { s.remove(s.min()) };	;;; Should empty the
set
		writeln(s.isEmpty());  ;;; Should write true
	}
	catch(e)
	{
		printExceptionTrace(&e)
	}
};
```

```
}}
*
"set.mmod"
*
{{
;;; Use:  val c = comp(x,y);
;;; Pre:  x and y are integers.
;;; Post: x is <0 if x<y, =0 if x==y, >0 if x>y.
comp =
        fun(x,y)
        {
                x<y && (return -1);
                y<x && (return 1);
                0
        };
}}
*
BASIS
;

;;; Another test program.
"testset2.mexe" = main in
{{
main =
   fun()
   {
      try
      {
         var x = [[1],[9],[2],[8],[3],[7],[4],[6],[5]];
         val s = makeSet();    ;;; A set of non-empty integer lists
```

```
        while( x )
        {
           ;;; Loop invariant:
           ;;; s contains a subset of the set {[1]..[9]}.
           ;;; The list x contains exactly the rest of
           ;;; the set {[1]..[9]}.
           s.remove(head(x));  ;;; Should have no effect
           s.add(head(x));
           s.add(head(x));     ;;; Should have no effect
           x = tail(x);
        };
        writeln(s.isEmpty());    ;;; Should write false
        writeln(s.mapReduce(fun(x,y){x+y},fun(x){head(x)},0));  ;;; Should write 45
        s.mapReduce(fun(x,y){[]},fun(x){write(x)},[]);          ;;; Should write
[1][2][3][4][5][6][7][8][9]
        writeln();
        while( !s.isEmpty() ) { s.remove(s.min()) };          ;;; Should empty the set
        writeln(s.isEmpty());    ;;; Should write true
      }
      catch(e)
      {
        printExceptionTrace(&e)
      }
   };
}}
*
"set.mmod"
*
{{
;;; Use:  val c = comp(x,y);
;;; Pre:  x and y are non-empty lists containing integers.
```

```
;;; Post: c is <0 if head(x)<head(y), =0 if head(x)==head(y),
;;;       >0 if head(x)>head(y).
comp =
   fun(x,y)
   {
      head(x)<head(y) && (return -1);
      head(y)<head(x) && (return 1);
      0
   };
}}
*
BASIS
;
```