GVE proof of array minimum example

John McHugh

11 March 1999

Introduction

This note contains an annotated transcript of an edited log of from the GVE. False starts and typos have been edited out. In most case abbreviated versions of the commands were typed, but the edited transcript shows the full versions echoed by the gve. The echoed characters appear in UPPER case.

Commentary appears in roman, the actual transcript in a `fixed width font`.

Preliminaries

The files for this example are in the course directory in `/home/comp527/gve-dir/example1`. The gve is in `/home/comp527/bin`. When you start the gve, it is important to specify line mode as character mode seems to lead to an immediate segmentation fault. This occurs whether it is run in an emacs shell window or from an xterm. Once the gve comes up, help or ? can be typed to most prompts. Instructions for running the gve can be found in the file `/home/comp527/gve-dir/documentation/report-002-gypsy-methodology.ps`.

```
%m:%~> cd gve-dir/example1
%m:%~> dir
total 206
 200 -rw-r--r--   1 comp527    195889 Mar 10 20:56 min.database
   1 -rw-r--r--   1 comp527       719 Mar 10 18:32 min1.gyp
   1 -rw-r--r--   1 comp527       787 Mar 10 18:34 min2.gyp
   1 -rw-r--r--   1 comp527       702 Mar 10 18:37 min3.gyp
   1 -rw-r--r--   1 comp527       938 Mar 10 19:11 min3a.gyp
   2 -rw-r--r--   1 comp527      1678 Mar 10 22:37 start.txt
%m:%~> gve
GCL (GNU Common Lisp)  Version(2.2.2) Tue Mar  9 20:06:58 CST 1999
Licensed under GNU Public Library License
Contains Enhancements by W. Schelter
Do you wish to run the GVE in character mode or line mode?
(Character, Line, Help)? l


Note: System Hacking [CLI=SYSTEM-HACKING] is set.

Checking loaded configuration of the GVE ...configuration ok!

Gypsy System version 20.70 (EXPERIMENTAL) of 1-Aug-1990 05:39:39 AM

Type 'NEWS' for changes in this version.

Report problems by using vbr at Dockmaster.
```

```
gve-init.lisp not found in user directory or default directory.

Welcome to the Experimental Gypsy 2.05 GVE ... you may begin.
Good luck!
```

Parsing

Parsing a gypsy source file is the easiest way to get information into the system. Gypsy supports cpp preprocessor statements, but it is simpler not to use them and it may not be able to find cpp. Setting the echo on helps see error messages, etc. Setting the path to the directory where the input files are located saves typing. The `.gyp` extension is assumed.

```
Gve -> set parSE prePROCESSOR off

Gve -> set parSE echO on

Gve -> set defAULT-FILE-NAME /home/comp527/gve-dir/example1/

Gve -> parse min1

scope minimum_procedure =
 begin

   procedure loc_of_min (var l : index; a : int_array; i, j : index) =
   begin
     var k : index := i;
     l := i;
     loop
       if k = j then leave end;
       k := k + 1;
       if a[k] < a[l] then l := k end;
     end;
   end;

   name {type} index, int_array from integer_array_types;

 end; {scope minimum_procedure}


scope integer_array_types =
 begin

    type int_array = array (index) of some_int;
    type index = integer[lo_index..hi_index];
    const lo_index : integer := 0;
    const hi_index : integer := 63;
    type some_int = integer[lo_int..hi_int];
    const lo_int : integer := - 1000;
    const hi_int : integer := 1000;

 end; {scope integer_array_types}


    No syntax errors detected
    No semantic errors detected

Gve -> parse min2
```

```
$extending
scope minimum_procedure =
 begin

   procedure loc_of_min (var l : index; a : int_array; i, j : index) =
   begin
     entry i le j;
     exit  is_minimum (a[l], a, i, j) & l in [i..j];
     var k : index := i;
     l := i;
     loop
       assert   is_minimum (a[l], a, i, k) & l in [i..j]
              & k in [i..j];
       if k = j then leave end;
       k := k + 1;
       if a[k] < a[l] then l := k end;
     end;
   end;

  name {function} is_minimum from minimum_specs;
  end; {scope minimum_procedure}

scope minimum_specs =
 begin

   function is_minimum (m : some_int;
                        a : int_array;
                        p, q : index) : boolean = pending;
    name {type} some_int, int_array, index from integer_array_types;

 end; {scope minimum_specs}


    No syntax errors detected
    No semantic errors detected

Gve -> parse min3

$extending
scope minimum_specs =
 begin

   lemma singleton_min (a : int_array; p : index) =
      (assume is_minimum (a[p], a, p, p));

   lemma extend_old_min_up (m : some_int; a : int_array; p, q : index) =
           (assume    is_minimum (m, a, p, q - 1) & m le a[q]
                   -> is_minimum (m, a, p, q));
   lemma extend_new_min_up (m : some_int; a : int_array; p, q : index) =
           (assume    is_minimum (m, a, p, q - 1) & a[q] le m
                   -> is_minimum (a[q], a, p, q));

 end; {scope minimum_specs}

$extending
scope minimum_procedure =
 begin
```

```
   name singleton_min, extend_old_min_up, extend_new_min_up
        from minimum_specs;

 end; {scope minimum_procedure}


    No syntax errors detected
    No semantic errors detected
```

Well formedness

Dynamic semantic errors such as potential type mismatches can give rise to situations in which it might be possible to prove "FALSE" in the prover. Since "FALSE" can be used to prove anything, the soundness of the system is called into doubt. The two extend lemmas in the previous section have this problem since it is not possible to know whether `q-1` is in the type `index` (consider the case where `q = lo_index`). This manifests itself in a well formedness vc that asks us to prove that `q-1` is in the type `index`. We can tell from the structure of the program that is not an issue since the first time we need the lemma is a case where `q > p` but the lemma is more general. One fix is to give `q` a type with a more restricted range so that `q - 1` is always in the type `index`. This is done in the revision in file `min3a.gyp`.

```
Gve -> genERATE vcS minimum_specs

 Changing default scope to MINIMUM_SPECS

Note: the body of IS_MINIMUM is not fully defined.

Generating VCs for FUNCTION IS_MINIMUM in scope MINIMUM_SPECS

Found 1st path
Found 2nd path
1 pending path encountered.


-------------------------------------------------
Beginning 1st of 2 paths...

Beginning VC generation for this path.

Well definedness VC's for function

End of path
-------------------------------------------------




-------------------------------------------------
Beginning 2nd of 2 paths...

Beginning VC generation for this path.

Routine has no ENTRY specification.

Routine has no CENTRY specification.

Initializing function result
```

```
RESULT := INITIAL (BOOLEAN)


Pending statement: assuming every variable modified.

Assert TRUE & TRUE

NORMAL exit from routine.

End of path
-------------------------------------------------



Generating VCs for LEMMA SINGLETON_MIN in scope MINIMUM_SPECS

Found 1st path


-------------------------------------------------
Beginning 1st of 1 paths...

Beginning VC generation for this path.

End of path
-------------------------------------------------



Generating VCs for LEMMA EXTEND_OLD_MIN_UP in scope MINIMUM_SPECS

Found 1st path


-------------------------------------------------
Beginning 1st of 1 paths...

Beginning VC generation for this path.
- - - - - - - - - - - - - - - - - - - - - - - -
Must verify (WELL-DEFINEDNESS-VC WELL-FORMED LEMMA) condition

Verification condition EXTEND_OLD_MIN_UP#1
  H1:    Q
      in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]
 -->
  C1:    Q - 1
      in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]
- - - - - - - - - - - - - - - - - - - - - - - -
Continuing in path...

End of path
-------------------------------------------------



Generating VCs for LEMMA EXTEND_NEW_MIN_UP in scope MINIMUM_SPECS

Found 1st path
```

```
------------------------------------------------
Beginning 1st of 1 paths...

Beginning VC generation for this path.
- - - - - - - - - - - - - - - - - - - - - - -
Must verify (WELL-DEFINEDNESS-VC WELL-FORMED LEMMA) condition

Verification condition EXTEND_NEW_MIN_UP#1
  H1:    Q
      in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]
 -->
  C1:    Q - 1
      in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]
- - - - - - - - - - - - - - - - - - - - - - -
Continuing in path...

End of path
------------------------------------------------



Gve -> prove extend_new_min_up#1



Entering Prover with verification condition EXTEND_NEW_MIN_UP#1 for LEMMA EXTEND_NEW_MIN_UP.


Updating typelist ............

     H1 :    Q
          in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]

   ->
     C1 :    Q - 1
          in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]

Prvr -> proc
procEED



Ran out of tricks.

Prvr -> th
thEOREM

     H1 :    Q
          in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]

   ->
     C1 :    Q - 1
          in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]

Prvr -> abort yes
```

```
The proof of EXTEND_NEW_MIN_UP#1 has been aborted.

Gve -> parse min3a

$extending
scope minimum_specs =
 begin

   type index_1 = integer[lo_index_1..hi_index];
   const lo_index_1 : integer := lo_index+1;

   lemma singleton_min (a : int_array; p : index) =
      (assume is_minimum (a[p], a, p, p));

   lemma extend_old_min_up (m : some_int; a : int_array;
                            p : index; q : index_1) =
         (assume    is_minimum (m, a, p, q - 1) & m le a[q]
                 -> is_minimum (m, a, p, q));
   lemma extend_new_min_up (m : some_int; a : int_array;
                            p : index; q : index_1) =
         (assume    is_minimum (m, a, p, q - 1) & a[q] le m
                 -> is_minimum (a[q], a, p, q));
   name {constant} lo_index, hi_index from integer_array_types;

 end; {scope minimum_specs}

$extending
scope minimum_procedure =
 begin

   name singleton_min, extend_old_min_up, extend_new_min_up
        from minimum_specs;

 end; {scope minimum_procedure}


    No syntax errors detected
    No semantic errors detected
```

Verification Conditions

Now we can generate verification conditions for all of the units in the program. As the VCs are generated, the system attempts to prove them using the symbolic evaluator, xeval. If this succeeds, no further action is required by the user. As VCs are generated, the program traces out the execution path that applies and displays the formulae to be proven.

```
Gve -> genERATE vcS all

 Changing default scope to INTEGER_ARRAY_TYPES

Generating new verification conditions for EXTEND_NEW_MIN_UP

Generating VCs for LEMMA EXTEND_NEW_MIN_UP in scope MINIMUM_SPECS

Found 1st path
```

```
-------------------------------------------------
Beginning 1st of 1 paths...

Beginning VC generation for this path.
- - - - - - - - - - - - - - - - - - - - - - -
Must verify (WELL-DEFINEDNESS-VC WELL-FORMED LEMMA) condition

Verification condition EXTEND_NEW_MIN_UP#3
  C1:    Q in [LO_INDEX_1..HI_INDEX]
      -> Q - 1 in [LO_INDEX..HI_INDEX]
  C2: Q in [LO_INDEX_1..HI_INDEX] -> LO_INDEX le Q
- - - - - - - - - - - - - - - - - - - - - - -
Continuing in path...

End of path
-------------------------------------------------


Still correct VCs:
           EXTEND_NEW_MIN_UP#2

Generating new verification conditions for EXTEND_OLD_MIN_UP

Generating VCs for LEMMA EXTEND_OLD_MIN_UP in scope MINIMUM_SPECS

Found 1st path


-------------------------------------------------
Beginning 1st of 1 paths...

Beginning VC generation for this path.
- - - - - - - - - - - - - - - - - - - - - - -
Must verify (WELL-DEFINEDNESS-VC WELL-FORMED LEMMA) condition

Verification condition EXTEND_OLD_MIN_UP#3
  C1:    Q in [LO_INDEX_1..HI_INDEX]
      -> Q - 1 in [LO_INDEX..HI_INDEX]
  C2: Q in [LO_INDEX_1..HI_INDEX] -> LO_INDEX le Q
- - - - - - - - - - - - - - - - - - - - - - -
Continuing in path...

End of path
-------------------------------------------------


Still correct VCs:
           EXTEND_OLD_MIN_UP#2
**** HI_INDEX is not provable.
**** HI_INT::INTEGER_ARRAY_TYPES is not provable.
**** INDEX is not provable.
**** INDEX_1 is not provable.
**** INT_ARRAY is not provable.

Note: the body of IS_MINIMUM is not fully defined.

Generating new verification conditions for IS_MINIMUM
```

```
Generating VCs for FUNCTION IS_MINIMUM in scope MINIMUM_SPECS

Found 1st path
Found 2nd path
1 pending path encountered.


-------------------------------------------------
Beginning 1st of 2 paths...

Beginning VC generation for this path.

Well definedness VC's for function

End of path
-------------------------------------------------




-------------------------------------------------
Beginning 2nd of 2 paths...

Beginning VC generation for this path.

Routine has no ENTRY specification.

Routine has no CENTRY specification.

Initializing function result
RESULT := INITIAL (BOOLEAN)


Pending statement: assuming every variable modified.

Assert TRUE & TRUE

NORMAL exit from routine.

End of path
-------------------------------------------------


Still correct VCs:
           IS_MINIMUM#1

Generating VCs for PROCEDURE LOC_OF_MIN in scope MINIMUM_PROCEDURE

Found 1st path
Found 2nd path
Found 3rd path
Found 4th path


-------------------------------------------------
Beginning 1st of 4 paths...

Beginning VC generation for this path.
```

```
Binding initial varparam values
L' := L

Assume (explicitly assumed)

     I le J

Assume (explicitly assumed)

     I le J


Routine has no CENTRY specification.

Initializing local variables and constants
K := I
L := I

Entering loop...

Evaluating  IS_MINIMUM (A[L], A, I, K)

Assert IS_MINIMUM (A[L], A, I, K) & L in [I..J] & K in [I..J]
- - - - - - - - - - - - - - - - - - - - - - - -
Must verify Assert condition

Verification condition LOC_OF_MIN#2
  H1: I le J
 -->
  C1: IS_MINIMUM (A[I], A, I, I)
- - - - - - - - - - - - - - - - - - - - - - - -

End of path
-------------------------------------------------




-------------------------------------------------
Beginning 2nd of 4 paths...

Continuing in Loop ...

Assume (from last assertion)

     IS_MINIMUM (A[L], A, I, K) & L in [I..J] & K in [I..J]


Beginning VC generation for this path.

Else branch of if ...

Assume (Cumulative failed If and Elif tests)

     not K = J
```

```
Routine INTEGER_ADD::PREDEFINED!

Returns Condition ROUTINEERROR having trivial path.
K := K + 1

Assume for well-formedness: Assigned value in value set of type of variable

     K


Else branch of if ...

Routine ARRAY_SELECT::PREDEFINED!

Returns Condition ROUTINEERROR having trivial path.

Routine ARRAY_SELECT::PREDEFINED!

Returns Condition ROUTINEERROR having trivial path.

Assume (Cumulative failed If and Elif tests)

     not A[K] < A[L]


Entering next iteration of loop...

Evaluating  IS_MINIMUM (A[L], A, I, K)

Assert IS_MINIMUM (A[L], A, I, K) & L in [I..J] & K in [I..J]
- - - - - - - - - - - - - - - - - - - - - - -
Must verify Assert condition

Verification condition LOC_OF_MIN#3
  H1: IS_MINIMUM (A[L], A, I, K)
  H2:    K + 1
      in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]
  H3: K in [I..J]
  H4: L in [I..J]
  H5: A[L] le A[K + 1]
  H6: J ne K
 -->
  C1: IS_MINIMUM (A[L], A, I, K + 1)
  C2: K + 1 in [I..J]
- - - - - - - - - - - - - - - - - - - - - - -

End of path
-------------------------------------------------




-------------------------------------------------
Beginning 3rd of 4 paths...

Continuing in Loop ...

Assume (from last assertion)
```

```
     IS_MINIMUM (A[L], A, I, K) & L in [I..J] & K in [I..J]


Beginning VC generation for this path.

Assume (If test succeeded)

     K = J

Leaving loop...


Evaluating  IS_MINIMUM (A[L], A, I, J)

Assert TRUE & IS_MINIMUM (A[L], A, I, J) & L in [I..J]
- - - - - - - - - - - - - - - - - - - - - - -
Must verify (NORMAL  exit specification for unit  LOC_OF_MIN) condition

Verification condition LOC_OF_MIN#4
  H1: J = K
  H2: IS_MINIMUM (A[L], A, I, K)
  H3: L in [I..J]
  H4: I le K
 -->
  C1: IS_MINIMUM (A[L], A, I, J)
- - - - - - - - - - - - - - - - - - - - - - -

NORMAL exit from routine.

End of path
------------------------------------------------





------------------------------------------------
Beginning 4th of 4 paths...

Continuing in Loop ...

Assume (from last assertion)

     IS_MINIMUM (A[L], A, I, K) & L in [I..J] & K in [I..J]


Else branch of if ...

Assume (Cumulative failed If and Elif tests)

     not K = J


Routine INTEGER_ADD::PREDEFINED!

Returns Condition ROUTINEERROR having trivial path.
K := K + 1
```

```
Assume for well-formedness: Assigned value in value set of type of variable

     K


Beginning VC generation for this path.

Routine ARRAY_SELECT::PREDEFINED!

Returns Condition ROUTINEERROR having trivial path.

Routine ARRAY_SELECT::PREDEFINED!

Returns Condition ROUTINEERROR having trivial path.

Assume (If test succeeded)

     A[K] < A[L]

L := K

Assume for well-formedness: Assigned value in value set of type of variable

     L


Entering next iteration of loop...

Evaluating  IS_MINIMUM (A[L], A, I, K)

Assert IS_MINIMUM (A[L], A, I, K) & L in [I..J] & K in [I..J]
- - - - - - - - - - - - - - - - - - - - - - -
Must verify Assert condition

Verification condition LOC_OF_MIN#5
  H1: IS_MINIMUM (A[L], A, I, K)
  H2:    K + 1
      in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]
  H3: K in [I..J]
  H4: L in [I..J]
  H5: A[K + 1] + 1 le A[L]
  H6: J ne K
 -->
  C1: IS_MINIMUM (A[K + 1], A, I, K + 1)
  C2: K + 1 in [I..J]
- - - - - - - - - - - - - - - - - - - - - - -

End of path
------------------------------------------------


**** LO_INDEX::INTEGER_ARRAY_TYPES is not provable.
**** LO_INDEX_1::MINIMUM_SPECS is not provable.
**** LO_INT::INTEGER_ARRAY_TYPES is not provable.

Generating new verification conditions for SINGLETON_MIN

Generating VCs for LEMMA SINGLETON_MIN in scope MINIMUM_SPECS
```

```
Found 1st path


-------------------------------------------------
Beginning 1st of 1 paths...

Beginning VC generation for this path.

End of path
-------------------------------------------------


Still correct VCs:
           SINGLETON_MIN#1
**** SOME_INT::INTEGER_ARRAY_TYPES is not provable.
```

Status

We can examine the status of the system at any time. At this point, we have 4 basic (or correctness) VCs and 2 well-formedness VCs to prove.

```
Gve -> show statUS all

SCOPE INTEGER_ARRAY_TYPES

   Types, constants:
         HI_INDEX  HI_INT    INDEX     INT_ARRAY LO_INDEX  LO_INT    SOME_INT


SCOPE MINIMUM_PROCEDURE

   VCs generated, not all proved:
         LOC_OF_MIN

   LOC_OF_MIN
      Correctness vcs:
         Basic vcs, proved in VC generator:
           LOC_OF_MIN#1
         Basic vcs, not yet attempted:
           LOC_OF_MIN#2, LOC_OF_MIN#3, LOC_OF_MIN#4, LOC_OF_MIN#5
         Well-formedness vcs, proved in VC generator:
           LOC_OF_MIN#1


SCOPE MINIMUM_SPECS

   Completely specified, all correctness VCs proved:
         SINGLETON_MIN
   All correctness VCs proved, but contains PENDING:
         IS_MINIMUM
   VCs generated, not all proved:
         EXTEND_NEW_MIN_UP EXTEND_OLD_MIN_UP
   Types, constants:
         INDEX_1    LO_INDEX_1

   EXTEND_NEW_MIN_UP
```

```
   Correctness vcs:
      Basic vcs, proved in VC generator:
        EXTEND_NEW_MIN_UP#2
      Well-formedness vcs, not yet attempted:
        EXTEND_NEW_MIN_UP#3

EXTEND_OLD_MIN_UP
   Correctness vcs:
      Basic vcs, proved in VC generator:
        EXTEND_OLD_MIN_UP#2
      Well-formedness vcs, not yet attempted:
        EXTEND_OLD_MIN_UP#3
```

Proving Well-formedness

Both of the well-formedness VCs can be proven simply by using the definition of the lower bound of the restricted index type. Note that, while the gve recognizes duplicate VCs within a unit, it does not recognize that it has produced identical VCs in different units. This is an instance of a general problem with equivalent VCs (VCs that have identical conclusions and a common subset of hypotheses sufficient for proof) that makes proof more tedious than it need be.

```
Gve -> prove extend_old_min_up#3


Entering Prover with verification condition EXTEND_OLD_MIN_UP#3 for LEMMA EXTEND_OLD_MIN_UP.


Updating typelist ............

     C1 :    Q in [LO_INDEX_1..HI_INDEX]
          -> Q - 1 in [LO_INDEX..HI_INDEX]
     C2 : Q in [LO_INDEX_1..HI_INDEX] -> LO_INDEX le Q

Prvr -> use lo_index_1


Updating typelist .............

Prvr -> thEOREM

     H1 : LO_INDEX + 1 = LO_INDEX_1

   ->
     C1 :    Q in [LO_INDEX_1..HI_INDEX]
          -> Q - 1 in [LO_INDEX..HI_INDEX]
     C2 : Q in [LO_INDEX_1..HI_INDEX] -> LO_INDEX le Q

Prvr -> procEED


3.  ANDSPLIT

Updating typelist ................

   5.  PROMOTE
      8.  SUPINF-INRANGE
         10.  TRUEC
```

```
Updating typelist ...............

   6.  PROMOTE
      14.  SUPINF-LEQ
         16.  TRUEC
Last Conjunct proved.

Theorem Proved!!
Prvr -> exit yes
Do you want to save this proof?   (Y or N) y

   EXTEND_OLD_MIN_UP#3
     Proved in theorem prover.
     Used in proof:
           HI_INDEX, LO_INDEX_1, LO_INDEX, INDEX_1

Gve -> prove extend_new_min_up#3
        .
        .
        .
       This VC is the same as the previous one and the proof is
       identical
        .
        .
        .
```

Proof of "correctness"

All that remains is to prove the functionality of the procedure LOC_OF_MIN. For the most part, this is straight forward, appealing to one of the specification lemmas and possibly appealing to a constant definition. Note that use of the lemmas requires matching variables in the lemma with variables or expressions in the theorem. In most cases, this is automatic, but manual guidance is required when there are multiple possibilities of the same type.

```
Gve -> prove loc_of_min

 Changing default scope to MINIMUM_PROCEDURE

   LOC_OF_MIN#1
     Proved in VC generator (true conclusion).



Entering Prover with verification condition LOC_OF_MIN#2 for PROCEDURE LOC_OF_MIN.


Updating typelist ...................

     H1 : I le J

   ->
     C1 : IS_MINIMUM (A[I], A, I, I)

Prvr -> use singleton_min

Prvr -> thEOREM
```

```
     H1 : IS_MINIMUM (A$#1[P$#1], A$#1, P$#1, P$#1)
     H2 : I le J

   ->
     C1 : IS_MINIMUM (A[I], A, I, I)

Prvr -> procEED


3.  UNIFY
   4.  TRUEC
Theorem Proved!!
Prvr -> exit yes
Do you want to save this proof?   (Y or N) y

   LOC_OF_MIN#2
     Proved in theorem prover.
     Used in proof:
           HI_INDEX::INTEGER_ARRAY_TYPES, LO_INDEX::INTEGER_ARRAY_TYPES,
          INT_ARRAY, SOME_INT::INTEGER_ARRAY_TYPES,
          LO_INT::INTEGER_ARRAY_TYPES, HI_INT::INTEGER_ARRAY_TYPES,
          SINGLETON_MIN, IS_MINIMUM, INDEX



Entering Prover with verification condition LOC_OF_MIN#3 for PROCEDURE LOC_OF_MIN.


Updating typelist ..................................................

     H1 : IS_MINIMUM (A[L], A, I, K)
     H2 :    K + 1
          in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]
     H3 : K in [I..J]
     H4 : L in [I..J]
     H5 : A[L] le A[K + 1]
     H6 : J ne K

   ->
     C1 : IS_MINIMUM (A[L], A, I, K + 1)
     C2 : K + 1 in [I..J]

Prvr -> procEED


1.  ANDSPLIT

Ran out of tricks.

Prvr -> thEOREM

     H1 : IS_MINIMUM (A[L], A, I, K)
     H2 :    K + 1
          in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]
     H3 : K in [I..J]
     H4 : L in [I..J]
     H5 : A[L] le A[K + 1]
```

```
     H6 : J ne K

   ->
     C1 : IS_MINIMUM (A[L], A, I, K + 1)

Prvr -> use extend_old_min_up

Prvr -> thEOREM

     H1 :    IS_MINIMUM (M$#1, A$#2, P$#2, Q$#1 - 1) & M$#1 le A$#2[Q$#1]
          -> IS_MINIMUM (M$#1, A$#2, P$#2, Q$#1)
     H2 : IS_MINIMUM (A[L], A, I, K)
     H3 :    K + 1
          in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]
     H4 : K in [I..J]
     H5 : L in [I..J]
     H6 : A[L] le A[K + 1]
     H7 : J ne K

   ->
     C1 : IS_MINIMUM (A[L], A, I, K + 1)

Prvr -> put

For what? [Enter a variable name, $help for help]: q$#1

Put what? ($help for help):k+1;

For what? [Enter a variable name, Type $done if finished, $help for help]: $done

Trying to justify PUT...


Updating typelist ................................

7.  PROMOTE
   9.  SIMPLIFY

Ran out of tricks.

The justification for the PUT hasn't been proved.
Do you want to assume the justification for now?   (Y or N) n

Ok. Prove the justification first.
Prvr -> thEOREM

     H1 : IS_MINIMUM (A[L], A, I, K)
     H2 :    K + 1
          in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]
     H3 : K in [I..J]
     H4 : L in [I..J]
     H5 : A[L] le A[K + 1]
     H6 : J ne K

   ->
     C1 :       K
             in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]
          ->    K + 1
```

```
            in [LO_INDEX_1::MINIMUM_SPECS..HI_INDEX::INTEGER_ARRAY_TYPES]

Prvr -> use lo_index_1

Updating typelist ..................................

Prvr -> procEED



Updating typelist ....................................

10.  PROMOTE
   12.  SUPINF-INRANGE
      14.  TRUEC

The justification for the PUT is proved.

Now using the PUT.

Prvr -> thEOREM

     H1 :      IS_MINIMUM (M$#1, A$#2, P$#2, K + 1 + -1)
             & M$#1 le A$#2[K + 1]
          -> IS_MINIMUM (M$#1, A$#2, P$#2, K + 1)
     H2 : IS_MINIMUM (A[L], A, I, K)
     H3 :    K + 1
          in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]
     H4 : K in [I..J]
     H5 : L in [I..J]
     H6 : A[L] le A[K + 1]
     H7 : J ne K

   ->
     C1 : IS_MINIMUM (A[L], A, I, K + 1)

Prvr -> procEED


8.  BACKCHAIN
   9.  UNIFY
      10.  ANDSPLIT
         12.  TRUEC
         13.  TRUEC
   10.  ANDSPLIT
      21.  SIMPLIFY
      22.  UNIFY

Conjunct 1 proved.
Proving Conjunct 2.

Prvr -> thEOREM

     H1 : IS_MINIMUM (A[L], A, I, K)
     H2 :    K + 1
          in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]
     H3 : K in [I..J]
     H4 : L in [I..J]
```

```
     H5 : A[L] le A[K + 1]
     H6 : J ne K

   ->
     C1 : K + 1 in [I..J]

Prvr -> procEED


4.  SUPINF-INRANGE
   6.  TRUEC
Last Conjunct proved.

Theorem Proved!!
Prvr -> exit yes
Do you want to save this proof?   (Y or N) y

   LOC_OF_MIN#3
     Proved in theorem prover.
     Used in proof:
           INDEX_1::MINIMUM_SPECS, HI_INDEX::INTEGER_ARRAY_TYPES,
          LO_INDEX::INTEGER_ARRAY_TYPES, INT_ARRAY,
          SOME_INT::INTEGER_ARRAY_TYPES, LO_INT::INTEGER_ARRAY_TYPES,
          HI_INT::INTEGER_ARRAY_TYPES, EXTEND_OLD_MIN_UP,
          LO_INDEX_1::MINIMUM_SPECS, IS_MINIMUM, INDEX



Entering Prover with verification condition LOC_OF_MIN#4 for PROCEDURE LOC_OF_MIN.


Updating typelist ...................................

     H1 : J = K
     H2 : IS_MINIMUM (A[L], A, I, K)
     H3 : L in [I..J]
     H4 : I le K

   ->
     C1 : IS_MINIMUM (A[L], A, I, J)

Prvr -> eqSUB
        Hypothesis label -> h1

  J := K


Updating typelist .............................

Prvr -> procEED


3.  UNIFY
Theorem Proved!!
Prvr -> exit yes
Do you want to save this proof?   (Y or N) y

   LOC_OF_MIN#4
```

```
     Proved in theorem prover.
     Used in proof:
           HI_INDEX::INTEGER_ARRAY_TYPES, LO_INDEX::INTEGER_ARRAY_TYPES,
          INT_ARRAY, SOME_INT::INTEGER_ARRAY_TYPES,
          LO_INT::INTEGER_ARRAY_TYPES, HI_INT::INTEGER_ARRAY_TYPES,
          IS_MINIMUM, INDEX



Entering Prover with verification condition LOC_OF_MIN#5 for PROCEDURE LOC_OF_MIN.


Updating typelist ...................................................

     H1 : IS_MINIMUM (A[L], A, I, K)
     H2 :    K + 1
          in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]
     H3 : K in [I..J]
     H4 : L in [I..J]
     H5 : A[K + 1] + 1 le A[L]
     H6 : J ne K

   ->
     C1 : IS_MINIMUM (A[K + 1], A, I, K + 1)
     C2 : K + 1 in [I..J]

Prvr -> use extend_new_min_up

Prvr -> thEOREM

     H1 :    IS_MINIMUM (M$#2, A$#3, P$#3, Q$#2 - 1) & A$#3[Q$#2] le M$#2
          -> IS_MINIMUM (A$#3[Q$#2], A$#3, P$#3, Q$#2)
     H2 : IS_MINIMUM (A[L], A, I, K)
     H3 :    K + 1
          in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]
     H4 : K in [I..J]
     H5 : L in [I..J]
     H6 : A[K + 1] + 1 le A[L]
     H7 : J ne K

   ->
     C1 : IS_MINIMUM (A[K + 1], A, I, K + 1)
     C2 : K + 1 in [I..J]

Prvr -> put

For what? [Enter a variable name, $help for help]: q$#2

Put what? ($help for help):k+1;

For what? [Enter a variable name, Type $done if finished, $help for help]: $done

Trying to justify PUT...


Updating typelist ................................

4.  PROMOTE
```

```
   6.  SIMPLIFY

Ran out of tricks.

The justification for the PUT hasn't been proved.
Do you want to assume the justification for now?   (Y or N) n

Ok. Prove the justification first.
Prvr -> thEOREM

     H1 : IS_MINIMUM (A[L], A, I, K)
     H2 :    K + 1
          in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]
     H3 : K in [I..J]
     H4 : L in [I..J]
     H5 : A[K + 1] + 1 le A[L]
     H6 : J ne K

   ->
     C1 :       K
             in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]
          ->    K + 1
             in [LO_INDEX_1::MINIMUM_SPECS..HI_INDEX::INTEGER_ARRAY_TYPES]

Prvr -> use lo_index_1

Updating typelist ..................................

Prvr -> procEED



Updating typelist ....................................

7.  PROMOTE
   9.  SUPINF-INRANGE
      11.  TRUEC

The justification for the PUT is proved.

Now using the PUT.

Prvr -> thEOREM

     H1 :      IS_MINIMUM (M$#2, A$#3, P$#3, K + 1 + -1)
             & A$#3[K + 1] le M$#2
          -> IS_MINIMUM (A$#3[K + 1], A$#3, P$#3, K + 1)
     H2 : IS_MINIMUM (A[L], A, I, K)
     H3 :    K + 1
          in [LO_INDEX::INTEGER_ARRAY_TYPES..HI_INDEX::INTEGER_ARRAY_TYPES]
     H4 : K in [I..J]
     H5 : L in [I..J]
     H6 : A[K + 1] + 1 le A[L]
     H7 : J ne K

   ->
     C1 : IS_MINIMUM (A[K + 1], A, I, K + 1)
     C2 : K + 1 in [I..J]
```

```
Prvr -> procEED


5.  ANDSPLIT
   7.  BACKCHAIN
      8.  UNIFY
         9.  TRUEC
      10.  ANDSPLIT
         16.  SIMPLIFY
            19.  UNIFY
               20.  TRUEC
         17.  SUPINF-LEQ
            25.  TRUEC
   8.  SUPINF-INRANGE
      29.  TRUEC
Last Conjunct proved.

Theorem Proved!!
Prvr -> exit yes
Do you want to save this proof?   (Y or N) y

   LOC_OF_MIN#5
     Proved in theorem prover.
     Used in proof:
           LO_INDEX::INTEGER_ARRAY_TYPES, HI_INDEX::INTEGER_ARRAY_TYPES,
          SOME_INT::INTEGER_ARRAY_TYPES, LO_INT::INTEGER_ARRAY_TYPES,
          HI_INT::INTEGER_ARRAY_TYPES, EXTEND_NEW_MIN_UP,
          INDEX_1::MINIMUM_SPECS, INT_ARRAY, LO_INDEX_1::MINIMUM_SPECS,
          IS_MINIMUM, INDEX
```

Finishing

A check of the status of the system shows that everything that could be proven has been. We can exit and save the database, allowing us to extend the system at a future date. Note that we saved the proofs as well so that they can be replayed, if desired.

```
Gve -> show stATUS all

SCOPE INTEGER_ARRAY_TYPES

   Types, constants:
         HI_INDEX  HI_INT    INDEX     INT_ARRAY LO_INDEX  LO_INT    SOME_INT


SCOPE MINIMUM_PROCEDURE

   Completely specified, all correctness VCs proved:
         LOC_OF_MIN


SCOPE MINIMUM_SPECS

   Completely specified, all correctness VCs proved:
         EXTEND_NEW_MIN_UP EXTEND_OLD_MIN_UP SINGLETON_MIN
   All correctness VCs proved, but contains PENDING:
         IS_MINIMUM
   Types, constants:
```

```
         INDEX_1    LO_INDEX_1


Gve -> exit
exit
       Save the data base? -> yes
yes
       Database file -> min
min

Saving.....................................................

No Dribble file to close, no dribbling on this machine



No log file to close.


Do you want to enter lisp? (N means killing this job
and going back to the shell)   (Y or N) n
%m:%~>
```