CS-Note* 62

Andrew Ireland
August 26, 2003

Proof Planning for Bubble Sort Verification

Contents

1 Introduction

This note explores the potential application of proof planning to the verification Bubble Sort. Starting with a fully annotated SPARK implementation of Bubble Sort, I focus on partial correctness. In subsequent note(s) I will consider the automatic generation of the intermediate proof annotations in detail. Exception freedom proofs will also be considered in a later note. This note is structured as follows. In §2 I present the version of Bubble Sort that is verified together with its specification (proof annotations). A detailed account of the verification is presented in §3. Finally in §4 I present my observations and thoughts on future developments based upon this exercise.

2 Bubble Sort: Specification & Implementation

The SPARK implementation[1] of Bubble Sort that is verified here is given in appendix A. As mentioned above, the code contains all the proof annotations (assertions) that are required in order to prove partial correctness of the code. The proof functions (predicates) used within the proof annotations are defined in appendix B. In figure 1 the proof annotations are also presented together with some additional explanations.

3 Partial correctness proof

The VCs generated by the SPARK Examiner for the partial correctness of Bubble Sort are given in appendix C. Out of the 11 VCs, 5 are trivial while the remaining 6 require user guided proof. Below I consider each of the VCs, but giving detailed consideration only to the non-trivial VCs.

*Notes in this series are for ϵ-baked ideas, for $1 \geq \epsilon \geq 0$. Only exceptionally should they be cited or distributed outwith the NuSPADE project (EPSRC GR/R24081).

[1] The Bubble Sort verified here corresponds to the version given in Niklaus Wirth's text book *"Algorithms + Data Structures = Programs"*.

Precondition: No preconditions.

```
--# pre true;
```

Outer loop invariant: `Table` is ordered from `0` to `I-2`. Elements from `0` to `I-2` are less than or equal to all the elements from `I-1` upwards. All elements are preserved and `I` remains within its type.

```
--# assert Ordered(Table, 0, I-2) and
--#        Partitioned(Table, 0, I-2, Index_Type'Last) and
--#        Perm(Table, Table~) and
--#        1 <= I and I <= Index_Type'Last;
```

Inner loop invariant: Extends the outer invariant to include that element `J` has the minimum value of the elements from `J` upwards. `I` remains within its type. `I` is less than or equal to `J`. The upper bound on `J` is `Index_Type'Last`.

```
--# assert Ordered(Table, 0, I-2) and
--#        Partitioned(Table, 0, I-2, Index_Type'Last) and
--#        Perm(Table, Table~) and
--#        Minimum(Table, J, Index_Type'Last) and
--#        1 <= I and I <= Index_Type'Last and
--#        I <= J and J <= Index_Type'Last;
```

Postcondition: On termination `Table` is ordered from `0` upwards and all elements are preserved.

```
--# post Ordered(Table, 0, Index_Type'Last) and
--#      Perm(Table, Table~);
```

Note that the definitions for `Ordered`, `Partitioned`, `Perm` *and* `Minimum` *are given in appendix B.*

Figure 1: SPARK annotations associated with Bubble Sort

3.1 The trivial VCs

By inspection, VC1 is trivial: Both C1 and C2 simplify to implications where the premise is false; C3 and C4 are true by their definitions while C5 follows from the proof context. In the case of VCs 2, 8 and 9, inspection of the hypotheses reveals contradictions. VC5 is also trivial, only C4 requires an unfolding of the definition of *minimum*.

3.2 VC3: Inner invariant to outer invariant (via then-branch)

The proof context associated with VC3 has the following hypotheses[2]:

$$ordered(table, 0, i-2) \tag{1}$$
$$partitioned(table, 0, i-2, last) \tag{2}$$
$$perm(table, table_0) \tag{3}$$
$$minimum(table, j, last) \tag{4}$$
$$1 \leq i \tag{5}$$
$$i \leq last \tag{6}$$
$$i \leq j \tag{7}$$
$$j \leq last \tag{8}$$
$$ele(table, [j-1]) > ele(table, [j]) \tag{9}$$
$$j = i \tag{10}$$
$$\neg(i = last) \tag{11}$$

[2]Note that I use *ele* and *upd* as abbreviations for *element* and *update* respectively.

$$\boxed{(X+1)}^{\uparrow} - Y \Rightarrow \boxed{(X-Y)+1}^{\uparrow} \tag{18}$$

$$(\forall X : int.((L \leq X) \wedge (X < \boxed{U+1}^{\uparrow})) \rightarrow P(X)) \Rightarrow \boxed{(\forall X : int.((L \leq X) \wedge (X < U)) \rightarrow P(X)) \wedge ((L \leq U) \rightarrow P(U))}^{\uparrow} \tag{19}$$

$$(\forall X : int.((\boxed{L-1}^{\uparrow} < X) \wedge (X \leq U)) \rightarrow P(X)) \Rightarrow \boxed{(\forall X : int.((L < X) \wedge (X \leq U)) \rightarrow P(X)) \wedge ((L \leq U) \rightarrow P(L))}^{\uparrow} \tag{20}$$

$$\begin{aligned}
&(\forall X : int.((L \leq X) \wedge (X \leq \boxed{M+1}^{\uparrow})) \rightarrow \\
&\quad (\forall Y : int.((\boxed{M+1}^{\uparrow} < Y) \wedge (Y \leq U)) \rightarrow P(X,Y))) \Rightarrow \boxed{(\forall X : int.((L \leq X) \wedge (X \leq M)) \rightarrow} \\
&\qquad \boxed{(\forall Y : int.((M < Y) \wedge (Y \leq U)) \rightarrow P(X,Y))) \wedge} \\
&\qquad \boxed{(\forall Z : int.((M+1 < Z) \wedge (Z \leq U)) \rightarrow P(M+1,Z)}^{\uparrow}
\end{aligned} \tag{21}$$

$$\begin{aligned}
&(\forall X : int.((L \leq X) \wedge (X < U)) \rightarrow P(X)) \Rightarrow \boxed{(\forall X : int.((L \leq X) \wedge (X < \boxed{U-1}^{+})) \rightarrow P(X)) \wedge} \\
&\qquad \boxed{((L \leq (U-1)) \rightarrow P(U-1))}^{\uparrow}
\end{aligned} \tag{22}$$

$$\begin{aligned}
&(\forall X : int.((L < X) \wedge (X \leq U)) \rightarrow P(\boxed{L-1}^{\uparrow}, X) \Rightarrow \boxed{(\forall X : int.((L < X) \wedge (X \leq U)) \rightarrow P(L,X)) \wedge} \\
&\qquad \boxed{((L \leq U) \rightarrow P(L-1,L))}^{\uparrow}
\end{aligned} \tag{23}$$

$$ele(\boxed{upd(X, Y-Z, ele(X,Y)))}^{\uparrow}, \boxed{Y-Z}^{\uparrow}) \Rightarrow ele(X,Y) \tag{24}$$

Figure 2: Wave-rules

The following conclusions are associated with VC3:

$$ordered(upd(upd(table, [j-1], ele(table, [j])), [j], ele(table, [j-1])), 0, (i+1)-2) \tag{12}$$
$$partitioned(upd(upd(table, [j-1], ele(table, [j])), [j], ele(table, [j-1])), 0, (i+1)-2, last) \tag{13}$$
$$perm(up(upd(table, [j-1], ele(table, [j])), [j], ele(table, [j-1])), table_0) \tag{14}$$
$$1 \leq i+1 \tag{15}$$
$$i+1 \leq last \tag{16}$$

Proof of VC3 conclusion (12):

Annotating (12) with respect to (1) gives:

$$ordered(\boxed{upd(upd(table, [j-1], ele(table, [j])), [j], ele(table, [j-1]))}^{\uparrow}, 0, \boxed{(i+1)}^{\uparrow} - 2)$$

Unfolding the definition of *ordered* gives:

$$(\forall p : int.((0 \leq p) \wedge (p < \boxed{(i+1)}^{\uparrow} - 2)) \rightarrow P(table, j, p) \leq P(table, j, p+1)) \tag{17}$$

where P is defined as follows:

$$P(A, J, X) \equiv ele(\boxed{upd(upd(A, [J-1], ele(table, [J])), [J], ele(table, [J-1]))}^{\uparrow}, [X])$$

We can now ripple (17) using (18) to get:

$$(\forall p : int.((0 \leq p) \wedge (p < \boxed{(i-2)+1}^{\uparrow})) \rightarrow P(table, j, p) \leq P(table, j, p+1))$$

Further rippling is achieved via (19):

$$\begin{aligned}
&\boxed{(\forall p : int.((0 \leq p) \wedge (p < i-2)) \rightarrow P(table, j, p) \leq P(table, j, p+1)) \wedge} \\
&\qquad \boxed{(0 \leq (i-2)) \rightarrow P(table, j, i-2) \leq P(table, j, i-2+1)}^{\uparrow}
\end{aligned} \tag{25}$$

To complete the rippling of (25), we must eliminate the wave-fronts within:

$$(\forall p : int.((0 \leq p) \wedge (p < i - 2)) \rightarrow P(table, j, p) \leq P(table, j, p + 1))$$

Focusing on the conclusion of the implication, then unfolding the definition of P on the left-hand side of the inequality gives:

$$ele(\boxed{upd(upd(table, [j-1], ele(table, [j])), [j], ele(table, [j-1]))}^{\uparrow}, [p]) \leq P(table, j, p + 1) \tag{26}$$

From (10) I can infer that the nested updates lie outside the range of p, *i.e.* p is in the range 0 to $i-3$. Consequently rippling reduces (26) to:

$$ele(table, [p]) \leq P(table, j, p + 1) \tag{27}$$

Unfolding the definition of P on the right-hand side (27) we have:

$$ele(table, [p]) \leq ele(\boxed{upd(upd(table, [j-1], ele(table, [j])), [j], ele(table, [j-1]))}^{\uparrow}, [p + 1]) \tag{28}$$

Again the nested updates do not effect the predicate, *i.e.* $p + 1$ is in the range 1 to $i - 2$. Rippling reduces (28) to:

$$ele(table, [p]) \leq ele(table, [p + 1])$$

This completes the rippling part of the proof, *i.e.*

$$\boxed{(\forall p : int.((0 \leq p) \wedge (p < i - 2)) \rightarrow ele(table, [p]) \leq ele(table, [p + 1])) \wedge}$$
$$\boxed{(0 \leq (i - 1)) \rightarrow P(table, j, i - 2) \leq P(table, j, i - 2 + 1)}^{\uparrow}$$

Strong fertilization with hypothesis (1) is now possible (modulo α-conv and definition folding). The post ripple residue takes the form:

$$(0 \leq (i - 2)) \rightarrow P(table, j, i - 2) \leq P(table, j, i - 2 + 1)$$

Decomposing the implication gives us a new hypothesis of the form:

$$0 \leq (i - 2) \tag{29}$$

Note that the conclusion of the implication simplifies as follows:

$$\begin{aligned} P(table, j, i - 2) &\leq P(table, j, i - 2 + 1) \\ P(table, j, i - 2) &\leq P(table, j, i - 1) \\ ele(table, [i - 2]) &\leq ele(table, [i]) \end{aligned} \tag{30}$$

Proof of (30) involves exploiting the fact that the *table* is partitioned (see hypothesis (2)). Given (29), then from (2) we can infer:

$$(\forall q : int.((i - 2 < q) \wedge (q \leq last)) \rightarrow ele(table, [i - 2]) \leq ele(table, [q])) \tag{31}$$

Furthermore, from (31) we can infer

$$ele(table, [i - 2]) \leq ele(table, [i])$$

which matches (30). This completes the proof of conclusion (12).

Proof of VC3 conclusion (13):

Annotating (13) with respect to (2) gives:

$$partitioned(\boxed{upd(upd(table, [j-1], ele(table, [j])), [j], ele(table, [j-1]))}^{\uparrow}, 0, \boxed{(i + 1)}^{\uparrow} - 2, last)$$

Unfolding the definition of *partitioned* gives:

$$(\forall p : int.((0 \leq p) \wedge (p \leq \boxed{(i+1)}^{\uparrow} - 2)) \rightarrow$$
$$(\forall q : int.((\boxed{(i+1)}^{\uparrow} - 2 > q) \wedge (q \leq last)) \rightarrow P(table, j, p) \leq P(table, j, q)))$$

We can now ripple using (18) to give:

$$(\forall p : int.((0 \leq p) \wedge (p \leq \boxed{(i-2)+1}^{\uparrow})) \rightarrow$$
$$(\forall q : int.((\boxed{(i-2)+1}^{\uparrow} < q) \wedge (q \leq last)) \rightarrow P(table, j, p) \leq P(table, j, q)))$$

Using (21) we further ripple to give:

$$\boxed{\begin{array}{l}(\forall p : int.((0 \leq p) \wedge (p \leq i-2)) \rightarrow (\forall q : int.((i-2 < q) \wedge (q \leq last)) \rightarrow P(table, j, p) \leq P(table, j, q))) \wedge \\ \qquad (\forall q' : int.(((i+1) - 2 < q) \wedge (q \leq last)) \rightarrow P(table, j, i-2+1) \leq P(table, j, q)))\end{array}}^{\uparrow} \tag{32}$$

To complete the rippling (32), we must eliminate the wave-fronts within:

$$(\forall p : int.((0 \leq p) \wedge (p \leq i-2)) \rightarrow (\forall q : int.((i-2 < q) \wedge (q \leq last)) \rightarrow P(table, j, p) \leq P(table, j, q)))$$

Focusing on the conclusion of the inner implication, and unfolding the definition of P on the left-hand side of the inequality gives:

$$ele(\boxed{upd(upd(table, [j-1], ele(table, [j])), [j], ele(table, [j-1]))}^{\uparrow}, [p]) \leq P(table, j, q) \tag{33}$$

From (10) we can infer that the nested updates lie outside the range of p. Consequently rippling reduces (33) to:

$$ele(table, [p]) \leq P(table, j, q) \tag{34}$$

Unfolding the definition of P on the right-hand side of (34) we have:

$$ele(table, [p]) \leq ele(\boxed{upd(upd(table, [j-1], ele(table, [j])), [j], ele(table, [j-1]))}^{\uparrow}, [q]) \tag{35}$$

A lemma is required that establishes that permuting elements within a partition does not effect the *partitioned* property. Such a lemma underpins the following wave-rule:

$$\begin{array}{l}(\forall P : int.((L \leq P) \wedge (P \leq M)) \rightarrow (\forall Q : int.((M < Q) \wedge (Q \leq U)) \rightarrow \\ \quad ele(A, [P]) \leq ele(\boxed{upd(upd(A, [X], ele(A, [Y])), [Y], ele(A, [X]))}^{\uparrow}, [Q]))) \Rightarrow \\ \quad \boxed{\begin{array}{l}(\forall P : int.((L \leq P) \wedge (P \leq M)) \rightarrow (\forall Q : int.((M < Q) \wedge (Q \leq U)) \rightarrow ele(A, [P]) \leq ele(A, [Q]))) \wedge \\ ((M < X) \wedge (X \leq U) \wedge (M < Y) \wedge (Y \leq U)) \vee ((L \leq X) \wedge (X \leq M) \wedge (L \leq Y) \wedge (Y \leq M))\end{array}}^{\uparrow}\end{array} \tag{36}$$

Unlike the other wave-rules used within this proof, wave-rule (36) is relatively application specific. Further discussion on this wave-rule (lemma) are delayed until §4. Pressing on with the proof, wave-rule (36) (modulo some trivial inequality reasoning) leads to strong fertilization (hypothesis (2)). The remaining residue takes the form:

$$(\forall q' : int.(((i+1) - 2 < q') \wedge (q' \leq last)) \rightarrow P(table, j, i-2+1) \leq P(table, j, q'))$$

The proof residue simplifies to give:

$$(\forall q' : int.((i-1 < q') \wedge (q' \leq last)) \rightarrow P(table, j, i-1) \leq P(table, j, q')) \tag{37}$$

Re-annotating (37) against (4) gives rise to the following goal:

$$(\forall q' : int.((\boxed{i-1}^{\uparrow} < q') \wedge (q' \leq last)) \rightarrow P(table, j, \boxed{i-1}^{\uparrow}) \leq P(table, j, q'))$$

Using wave-rule (20) this ripples to give:

$$\boxed{\underline{(\forall q' : int.((i < q') \wedge (q' \leq last)) \rightarrow P(table, j, \boxed{\underline{i} - 1}^{\uparrow}) \leq P(table, j, q'))} \wedge ((i \leq last) \rightarrow P(table, j, i-1) \leq P(table, j, i))}^{\uparrow} \quad (38)$$

To complete the ripple part of the proof we must consider the conclusion of the implication, *i.e.*

$$P(table, j, \boxed{\underline{i} - 1}^{\uparrow}) \leq P(table, j, q')$$

Unfolding the definition of P on the left-hand side of the inequality gives:

$$ele(\boxed{upd(upd(\underline{table}, [j-1], ele(table, [j])), [j], ele(table, [j-1]))}^{\uparrow}, [\boxed{\underline{j} - 1}^{\uparrow}]) \leq \ldots$$

At this point in the proof we would like to apply wave-rule (24)[3]. The application of (24), however, is blocked because we only have a partial match between the left-hand side of the wave-rule and the left-hand side of the inequality. The problem becomes clearer if we split the wave-fronts within the blockage, *i.e.*

$$ele(\underbrace{\boxed{upd(\underline{\boxed{upd(\underline{table}, [j-1], ele(table, [j]))}^{\uparrow}}, [j], ele(table, [j-1]))}^{\uparrow}}_{\textbf{blocked}}, [\boxed{\underline{j} - 1}^{\uparrow}]) \leq \ldots$$

Note that while the inner wave-front is required for a match with wave-rule (24), the outer wave-front is preventing the match. The ripple can be unblocked, or patched, by reversing the order of the nested wave-fronts to enable the match, *i.e.*

$$ele(\underbrace{\boxed{upd(\underline{\boxed{upd(\underline{table}, [j], ele(table, [j-1]))}^{\uparrow}}, [j-1], ele(table, [j]))}^{\uparrow}}_{\textbf{unblocked}}, [\boxed{\underline{j} - 1}^{\uparrow}]) \leq \ldots$$

This unblocking step is justified by SPADE proof rule array(4), which checks that the array indices are disjoint. While such an unblocking may require an arbitrary number of re-orderings, I suspect in practice that the number of wave-front re-orderings will be small, *i.e.* 2 or 3. Applying wave-rule (24) to the unblocked goal gives:

$$ele(table, [j]) \leq \ldots$$

Unfolding the definition of P on the right-hand side of the inequality gives:

$$\ldots \leq ele(\boxed{upd(upd(\underline{table}, [j-1], ele(table, [j])), [j], ele(table, [j-1]))}^{\uparrow}, [q']) \quad (39)$$

Given (10) and the fact that elements j^{th} and $j - 1^{th}$ of *table* lie out with the range of q' then (39) ripples to give:

$$\ldots \leq ele(table, [q'])$$

The effect of this ripple is to reduce (38) to:

$$\boxed{\underline{(\forall q' : int.((i < q') \wedge (q' \leq last)) \rightarrow ele(table, i) \leq ele(table, q'))} \wedge ((i \leq last) \rightarrow P(table, j, i-1) \leq P(table, j, i))}^{\uparrow}$$

Strong fertilization with (4) (modulo α-conv and folding) leaves a proof residue of the form:

$$(i \leq last) \rightarrow P(table, j, i-1) \leq P(table, j, i)$$

This residue simplifies to give:

$$ele(table, [j]) \leq ele(table, [j-1])$$

Note that this goal follows from (9). This completes the proof of conclusion (12).

[3]Note that wave-rule (24) is derived from an instance of SPADE proof rule `array(1)`.

Proof of VC3 conclusion (14):

Relatively trivial.

Proof of VC3 conclusion (15):

Relatively trivial.

Proof of VC3 conclusion (16):

Relatively trivial.

3.3 VC4: Inner invariant to outer invariant (skipping then-branch)

The proof context associated with VC4 has the following hypotheses:

$$ordered(table, 0, i-2) \tag{40}$$
$$partitioned(table, 0, i-2, last) \tag{41}$$
$$perm(table, table_0) \tag{42}$$
$$minimum(table, j, last) \tag{43}$$
$$1 \leq i \tag{44}$$
$$i \leq last \tag{45}$$
$$i \leq j \tag{46}$$
$$j \leq last \tag{47}$$
$$\neg(ele(table, [j-1]) > ele(table, [j])) \tag{48}$$
$$j = i \tag{49}$$
$$\neg(i = last) \tag{50}$$

The following conclusions are associated with VC4:

$$ordered(table, 0, (i+1)-2) \tag{51}$$
$$partitioned(table, 0, (i+1)-2, last) \tag{52}$$
$$perm(table, table_0) \tag{53}$$
$$1 \leq i+1 \tag{54}$$
$$i+1 \leq last \tag{55}$$

Proof of VC4 conclusion (51):

Annotating (51) with respect to (40) gives:

$$ordered(table, 0, \boxed{(\underline{i}+1)}^{\uparrow} - 2)$$

Unfolding the definition of *ordered* gives:

$$(\forall p : int.((0 \leq p) \wedge (p < \boxed{(\underline{i}+1)}^{\uparrow} - 2)) \rightarrow ele(table, [p]) \leq ele(table, [p+1])) \tag{56}$$

We can now ripple (56), using (18) to give:

$$(\forall p : int.((0 \leq p) \wedge (p < \boxed{(\underline{i-2})+1}^{\uparrow})) \rightarrow ele(table, [p]) \leq ele(table, [p+1])) \tag{57}$$

Using wave-rule (19), (57) ripples to give:

$$\boxed{\underline{(\forall p : int.((0 \leq p) \wedge (p < i-2)) \rightarrow ele(table, [p]) \leq ele(table, [p+1]))} \wedge (0 \leq (i-2)) \rightarrow ele(table, [i-2]) \leq ele(table, [i-2+1])}^{\uparrow} \tag{58}$$

Fertilize with (40) leaves a proof residue of the form:

$$(0 \leq (i-2)) \rightarrow ele(table, [i-2]) \leq ele(table, [i-2+1])$$

Decomposing the implication gives us a new hypothesis of the form:

$$0 \leq (i - 2) \tag{59}$$

Note that the conclusion of the implication simplifies to give:

$$ele(table, [i - 2]) \leq ele(table, [i - 1]) \tag{60}$$

Proof of (60) involves exploiting the fact that the *table* is partitioned (see hypothesis (41)). Given (59), then from (41) we can infer:

$$(\forall q : int.((i - 2 < q) \wedge (q \leq last)) \rightarrow ele(table, [i - 2]) \leq ele(table, [q])) \tag{61}$$

Furthermore, from (61) we can infer

$$ele(table, [i - 2]) \leq ele(table, [i - 1])$$

which matches (60). This completes the proof of conclusion (51).

Proof of VC4 conclusion (52):

Annotating (52) with respect to (41) gives:

$$partitioned(table, 0, \boxed{(\underline{i} + 1)}^{\uparrow} - 2, last)$$

Unfolding the definition of *partitioned* gives:

$$\begin{array}{l}(\forall p : int.((0 \leq p) \wedge (p \leq \boxed{(\underline{i} + 1)}^{\uparrow} - 2)) \rightarrow \\ \quad (\forall q : int.((\boxed{(\underline{i} + 1)}^{\uparrow} - 2 < q) \wedge (q \leq last)) \rightarrow ele(table, [p]) \leq ele(table, [q])))\end{array}$$

We can now ripple using (18) to give:

$$\begin{array}{l}(\forall p : int.((0 \leq p) \wedge (p \leq \boxed{(\underline{i - 2}) + 1}^{\uparrow})) \rightarrow \\ \quad (\forall q : int.((\boxed{(\underline{i - 2}) + 1}^{\uparrow} < q) \wedge (q \leq last)) \rightarrow ele(table, [p]) \leq ele(table, [q])))\end{array}$$

Using (21) we ripple to give:

$$\boxed{\begin{array}{l}\underline{(\forall p : int.((0 \leq p) \wedge (p \leq i - 2)) \rightarrow (\forall q : int.((i - 2 < q) \wedge (q \leq last)) \rightarrow ele(table, [p]) \leq ele(table, [q])))} \wedge \\ \quad (\forall q' : int.(((i - 2) + 1 < q') \wedge (q' \leq last)) \rightarrow ele(table, [i - 2 + 1]) \leq ele(table, [q'])))\end{array}}^{\uparrow}$$

Strong fertilization is now applicable (hypothesis (41)), leaving a proof residue of the form:

$$(\forall q' : int.(((i - 2) + 1 < q') \wedge (q' \leq last)) \rightarrow ele(table, [i - 2 + 1]) \leq ele(table, [q']))$$

The proof residue simplifies to give:

$$(\forall q' : int.((i - 1 < q') \wedge (q' \leq last)) \rightarrow ele(table, [i - 1]) \leq ele(table, [q'])) \tag{62}$$

Re-annotating (62) against (43) gives rise to the following goal:

$$(\forall q' : int.((\boxed{(\underline{i} - 1)}^{\uparrow} < q') \wedge (q' \leq last)) \rightarrow ele(table, [\boxed{(\underline{i} - 1)}^{\uparrow}]) \leq ele(table, [q'])) \tag{63}$$

Using wave-rule (20), (63) reduces to give:

$$\boxed{\begin{array}{l}\underline{(\forall q' : int.((i < q') \wedge (q' \leq last)) \rightarrow ele(table, [\boxed{(\underline{i} - 1)}^{\uparrow}]) \leq ele(table, [q']))} \wedge \\ \quad (i \leq last) \rightarrow ele(table, [i - 1]) \leq ele(table, [i]))\end{array}}^{\uparrow}$$

Wave-rule (23) completes the rippling[4]:

$$\begin{aligned} & \boxed{(\forall q' : int.((i < q') \wedge (q' \leq last)) \rightarrow ele(table, [i]) \leq ele(table, [q']))} \wedge \\ & (i \leq last) \rightarrow ele(table, [i-1]) \leq ele(table, [i]) \wedge \\ & (i \leq last) \rightarrow ele(table, [i-1]) \leq ele(table, [i])^{\uparrow} \end{aligned}$$

Given (49), strong fertilization with (43) (modulo α-conv and folding) is now applicable. Fertilization leaves a residue that simplifies to give:

$$ele(table, [i-1]) \leq ele(table, [i]) \tag{64}$$

Note that this simplified residue follows from hypothesis (48), and so completes the proof of conclusion (52).

Proof of VC4 conclusion (53):

Relatively trivial.

Proof of VC4 conclusion (54):

Relatively trivial.

Proof of VC4 conclusion (55):

Relatively trivial.

3.4 VC6: Inner invariant to inner invariant (via then-branch)

The proof context associated with VC6 has the following hypotheses:

$$ordered(table, 0, i-2) \tag{65}$$
$$partitioned(table, 0, i-2, last) \tag{66}$$
$$perm(table, table_0) \tag{67}$$
$$minimum(table, j, last) \tag{68}$$
$$1 \leq i \tag{69}$$
$$i \leq last \tag{70}$$
$$i \leq j \tag{71}$$
$$j \leq last \tag{72}$$
$$ele(table, [j-1]) > ele(table, [j]) \tag{73}$$
$$\neg(j = i) \tag{74}$$

The following conclusions are associated with VC6:

$$ordered(upd(upd(table, [j-1], ele(table, [j])), [j], ele(table, [j-1])), 0, i-2) \tag{75}$$
$$partitioned(upd(upd(table, [j-1], ele(table, [j])), [j], ele(table, [j-1])), 0, i-2, last) \tag{76}$$
$$perm(upd(upd(table, [j-1], ele(table, [j])), [j], ele(table, [j-1])), table_0) \tag{77}$$
$$minimum(upd(upd(table, [j-1], ele(table, [j])), [j], ele(table, [j-1])), j-1, last) \tag{78}$$
$$1 \leq i \tag{79}$$
$$i \leq last \tag{80}$$
$$i \leq j-1 \tag{81}$$
$$j-1 \leq last \tag{82}$$

[4] Note that we now have 2 identical conjuncts within the goal. I suspect that this duplication could be eliminated if domain specific lemmas (wave-rules) were used. However, we are interested in minimizing the number of domain specific lemmas that are required.

Proof of VC6 conclusion (75):

Annotating (75) with respect to (65) gives:

$$ordered(\boxed{upd(upd(\underline{table}, [j-1], ele(table, [j])), [j], ele(table, [j-1]))}^{\uparrow}, 0, i-2)$$

Unfolding the definition of *ordered* gives:

$$(\forall p : int.((0 \leq p) \wedge (p < i-2)) \rightarrow P(table, j, p) \leq P(table, j, p+1)) \tag{83}$$

where P is defined as above. Given that the upper bound on p is $i-3$ and $i < j$, then the updates (wave-fronts) that occur within $P(table, j, p)$ and $P(table, j, p+1)$ can be eliminated leaving:

$$(\forall p : int.((0 \leq p) \wedge (p < i-2)) \rightarrow ele(table, [p]) \leq ele(table, [p+1]))$$

Strong fertilization (modulo α-conv and folding) with hypothesis (65) completes the proof of conclusion (75).

Proof of VC6 conclusion (76):

Annotating (76) with respect to (66) gives:

$$partitioned(\boxed{upd(upd(\underline{table}, [j-1], ele(table, [j])), [j], ele(table, [j-1]))}^{\uparrow}, 0, i-2, last)$$

Unfolding the definition of *partitioned* gives:

$$\begin{aligned}&(\forall p : int.((0 \leq p) \wedge (p \leq i-2)) \rightarrow\\ &\qquad\qquad (\forall q : int.((i-2 > q) \wedge (q \leq last)) \rightarrow P(table, j, p) \leq P(table, j, q)))\end{aligned}$$

Unlike in the case of conclusion (75), reasoning about the updates to *table* is non-trivial. Basically it involves proving that permuting 2 elements within a partition does not effect the truth of the *partitioned* property. Note that the need for such a lemma was highlighted above in §3.2, where wave-rule (36) was introduced. Following the reasoning sketched in §3.2, strong fertilization (hypothesis (66)) follows (modulo α-conv and folding). This completes the proof of conclusion (76).

Proof of VC6 conclusion (77):

Proof is relatively trivial, *i.e.* follows from the definition of *perm*.

Proof of VC6 conclusion (78):

Annotating (78) with respect to (68) gives:

$$minimum(\boxed{upd(upd(\underline{table}, [j-1], ele(table, [j])), [j], ele(table, [j-1]))}^{\uparrow}, \boxed{\underline{j}-1}^{\uparrow}, last)$$

Unfolding the definition of *minimum* gives:

$$(\forall p : int.((\boxed{\underline{j}-1}^{\uparrow} < p) \wedge (p \leq last)) \rightarrow P(table, j, \boxed{\underline{j}-1}^{\uparrow}) \leq P(table, j, p))$$

Using wave-rule (20) we get:

$$\begin{aligned}&\boxed{\underline{(\forall p : int.((j < p) \wedge (p \leq last)) \rightarrow P(table, j, \boxed{\underline{j}-1}^{\uparrow}) \leq P(table, j, p))} \wedge}\\ &\qquad\qquad\qquad \boxed{((j \leq last) \rightarrow P(table, j, j-1) \leq P(table, j, j))}^{\uparrow}\end{aligned} \tag{84}$$

To complete the ripple part of the proof we must consider the conclusion of the implication, *i.e.*

$$P(table, j, \boxed{\underline{j}-1}^{\uparrow}) \leq P(table, j, p)$$

Unfolding the definition of P on the left-hand side of the inequality gives:

$$ele(\boxed{upd(upd(\underline{table}, [j-1], ele(table, [j])), [j], ele(table, [j-1]))}^{\uparrow}, [\boxed{\underline{j}-1}^{\uparrow}]) \leq \ldots$$

As illustrated earlier (see §3.2), an unblocking step is required, *i.e.*

$$ele(\underbrace{\boxed{upd(upd(\boxed{table},[j],ele(table,[j-1])),[j-1],ele(table,[j]))}^{\uparrow},[\boxed{\boxed{j}-1}^{\uparrow}]}_{\textbf{unblocked}}) \leq \ldots$$

As before, the re-ordering of the wave-fronts enables the application of wave-rule (24) to the unblocked goal, *i.e.*

$$ele(table,[j]) \leq \ldots$$

Unfolding the definition of P on the right-hand side of the inequality gives:

$$\ldots \leq ele(\boxed{upd(upd(\boxed{table},[j-1],ele(table,[j])),[j],ele(table,[j-1]))}^{\uparrow},[p]) \tag{85}$$

Given that the j^{th} and $j-1^{th}$ elements of *table* lie out with the range of p then (85) ripples to give:

$$\ldots \leq ele(table,[p])$$

The effect of this ripple is to reduce (84) to:

$$\boxed{\boxed{(\forall p: int.((j<p) \land (p \leq last)) \to ele(table,[j]) \leq ele(table,[p]))} \land ((j \leq last) \to P(table,j,j-1) \leq P(table,j,j))}^{\uparrow}$$

Strong fertilization (hypothesis (68)) leaves a proof residue of the form:

$$(j \leq last) \to P(table,j,j-1) \leq P(table,j,j)$$

This residue simplifies to give:

$$ele(table,[j]) \leq ele(table,[j-1]) \tag{86}$$

Note that (86) follows from hypothesis (73). This completes the proof of (78).

Proof of VC6 conclusion (79):

Relatively trivial.

Proof of VC6 conclusion (80):

Relatively trivial.

Proof of VC6 conclusion (81):

Given hypotheses (71) and (74), a proof of (81) is relatively trivial.

Proof of VC6 conclusion (82):

Relatively trivial.

3.5 VC7: Inner invariant to inner invariant (skipping then-branch)

The proof context associated with VC7 has the following hypotheses:

$$ordered(table,0,i-2) \tag{87}$$
$$partitioned(table,0,i-2,last) \tag{88}$$
$$perm(table,table_0) \tag{89}$$
$$minimum(table,j,last) \tag{90}$$
$$1 \leq i \tag{91}$$
$$i \leq last \tag{92}$$
$$i \leq j \tag{93}$$
$$j \leq last \tag{94}$$
$$\neg(ele(table,[j-1]) > ele(table,[j])) \tag{95}$$
$$\neg(j = i) \tag{96}$$

The following conclusions are associated with VC7:

$$ordered(table, 0, i - 2) \tag{97}$$
$$partitioned(table, 0, i - 2, last) \tag{98}$$
$$perm(table, table_0) \tag{99}$$
$$minimum(table, j - 1, last) \tag{100}$$
$$1 \leq i \tag{101}$$
$$i \leq last \tag{102}$$
$$i \leq j - 1 \tag{103}$$
$$j - 1 \leq last \tag{104}$$

Proof of VC7 conclusion (97):

Conclusion (97) follows directly from (87).

Proof of VC7 conclusion (98):

Conclusion (98) follows directly from (88).

Proof of VC7 conclusion (99):

Conclusion (99) follows directly from (89).

Proof of VC7 conclusion (100):

Annotating (100) with respect to (90) gives:

$$minimum(table, \boxed{j - 1}^{\uparrow}, last)$$

Unfolding the definition of *minimum* gives:

$$(\forall p : int.((\boxed{j - 1}^{\uparrow} < p) \land (p \leq last)) \to ele(table, [\boxed{j - 1}^{\uparrow}]) \leq ele(table, [p])) \tag{105}$$

Using wave-rule (20), (105) ripples to give:

$$\boxed{(\forall p : int.((j < p) \land (p \leq last)) \to ele(table, [\boxed{j - 1}^{\uparrow}]) \leq ele(table, [p])) \land \\ ((j < last) \to ele(table, [j - 1]) \leq ele(table, [j]))}^{\uparrow}$$

Further rippling using (23) gives us:

$$\boxed{(\forall p : int.((j < p) \land (p \leq last)) \to ele(table, [j]) \leq ele(table, [p])) \land \\ (j < last) \to ele(table, [j - 1]) \leq ele(table, [j]) \land \\ (j < last) \to ele(table, [j - 1]) \leq ele(table, [j])}^{\uparrow}$$

Strong fertilization is now applicable (hypothesis (90)) is now applicable (modulo α-conv and folding). The resulting proof residue simplifies to give:

$$ele(table, [j - 1]) \leq ele(table, [j]))$$

Given hypothesis (95), this residue is relatively trivial to prove. This completes the proof of conclusion (100).

Proof of VC7 conclusion (101):

Trivial.

Proof of VC7 conclusion (102):

Trivial.

Proof of VC7 conclusion (103):

Given hypotheses (93) and (96), a proof of (103) is relatively trivial.

Proof of VC7 conclusion (104):

Relatively trivial.

3.6 VC10: Inner invariant to finish (via then-branch)

The proof context associated with VC10 has the following hypotheses:

$$ordered(table, 0, i-2) \tag{106}$$

$$partitioned(table, 0, i-2, last) \tag{107}$$

$$perm(table, table_0) \tag{108}$$

$$minimum(table, j, last) \tag{109}$$

$$1 \leq i \tag{110}$$

$$l \leq last \tag{111}$$

$$i \leq j \tag{112}$$

$$j \leq last \tag{113}$$

$$ele(table, [j-1]) > ele(table, [j]) \tag{114}$$

$$j = i \tag{115}$$

$$i = last \tag{116}$$

The following conclusions are associated with VC10:

$$ordered(upd(upd(table, [j-1], ele(table, [j])), [j], ele(table, [j-1])), 0, last) \tag{117}$$

$$perm(upd(upd(table, [j-1], ele(table, [j])), [j], ele(table, [j-1])), table_0) \tag{118}$$

Proof of VC10 conclusion (117):

Using hypothesis (113) rewrite (117) to give:

$$ordered(upd(upd(table, [j-1], ele(table, [j])), [j], ele(table, [j-1])), 0, i) \tag{119}$$

Annotating (119) with respect to (106) gives:

$$ordered(\boxed{upd(upd(\underline{table}, [j-1], ele(table, [j])), [j], ele(table, [j-1]))}^{\uparrow}, 0, i) \tag{120}$$

However, in order annotate correcting we must annotate the target hypothesis as well, *i.e.* hypothesis (106) becomes:

$$ordered(table, 0, \boxed{\underline{i} - 2}^{-}) \tag{121}$$

Note that I use an anti-wave-front (borrowed from rippling for destructor style induction). In order to *neutralize* the anti-wave-front we need to induce a matching wave-front within (120). This can be achieved via range splitting, this involves unfolding the definition of *ordered* to give:

$$(\forall p : int.((0 \leq p) \wedge (p < i)) \rightarrow P(table, j, p) \leq P(table, j, p+1)) \tag{122}$$

where P is as defined earlier. Applying (22) (twice + simplification) to (122) gives:

$$\boxed{\begin{array}{r} \underline{(\forall p : int.((0 \leq p) \wedge (p < \boxed{\underline{i} - 2}^{+})) \rightarrow P(table, j, p) \leq P(table, j, p+1))} \wedge \\ (0 \leq (i-2)) \rightarrow P(table, j, i-2) \leq P(table, j, i-1) \wedge \\ (0 \leq (i-1)) \rightarrow P(table, j, i-1) \leq P(table, j, i) \end{array}}^{\uparrow} \tag{123}$$

Note that the introduction of the $\boxed{\ldots\ldots}^{+}$ wave-front within the goal neutralizes the $\boxed{\ldots\ldots}^{-}$ wave-front within hypothesis (121). As previously, reasoning about the range of p allows us to ripple (123) to give:

$$\boxed{\begin{array}{c}(\forall p : int.((0 \leq p) \wedge (p < i-2)) \rightarrow ele(table,[p]) \leq ele(table,[p+1])) \wedge \\ P(table,j,i-2) \leq P(table,j,i-1) \ \wedge \ P(table,j,i-1) \leq P(table,j,i)\end{array}}^{\uparrow}$$

Strong fertilization (hypothesis (106)) leaves a proof residue of the form:

$$\begin{array}{l}(0 \leq (i-2)) \rightarrow P(table,j,i-2) \leq P(table,j,i-1) \ \wedge \\ \qquad (0 \leq (i-1)) \rightarrow P(table,j,i-1) \leq P(table,j,i)\end{array}$$

This residue simplifies to give:

$$\begin{array}{l}(0 \leq (i-2)) \rightarrow ele(table,[i-2]) \leq ele(table,[i]) \ \wedge \\ \quad (0 \leq (i-1)) \rightarrow ele(table,[i]) \leq ele(table,[i-1])\end{array}$$

The first conjunct follows by specializing hypothesis (107), while the second conjunct follows from hypothesis (114). This completes the proof of conclusion (117).

Proof of VC10 conclusion (118):

Given the definition of *permute*, (118) is relatively trivial.

3.7 VC11: Inner invariant to finish (skipping then-branch)

The proof context associated with VC11 has the following hypotheses:

$$ordered(table,0,i-2) \tag{124}$$
$$partitioned(table,0,i-2,last) \tag{125}$$
$$perm(table,table_0) \tag{126}$$
$$minimum(table,j,last) \tag{127}$$
$$1 \leq i \tag{128}$$
$$i \leq last \tag{129}$$
$$i \leq j \tag{130}$$
$$j \leq last \tag{131}$$
$$\neg(ele(table,[j-1]) > ele(table,[j])) \tag{132}$$
$$j = i \tag{133}$$
$$i = last \tag{134}$$

The following conclusions are associated with VC11:

$$ordered(table,0,last) \tag{135}$$
$$perm(table,table_0) \tag{136}$$

Proof of VC11 conclusion (135):

Using hypothesis (134) to rewrite (135) to give:

$$ordered(table,0,i) \tag{137}$$

Annotating (137) with respect to (124) introduces annotations into (124), *i.e.*

$$ordered(table,0,\boxed{\underline{i}-2}^{-}) \tag{138}$$

In order to *neutralize* the anti-wave-front we need to introduce a matching wave-front within (137). As before this is achieved by an unfolding step followed by range splitting. Unfolding the definition of *ordered* gives:

$$(\forall p : int.((0 \leq p) \wedge (p < i)) \rightarrow ele(table,[p]) \leq ele(table,[p+1])) \tag{139}$$

Applying (22) (twice + simplification) to (139) gives: achieves the required neutralization:

$$\boxed{(\forall p : int.((0 \leq p) \wedge (p < \boxed{i - 2}^{+})) \rightarrow ele(table, [p]) \leq ele(table, [p + 1]))} \wedge$$
$$(0 \leq (i - 2)) \rightarrow ele(table, [i - 2]) \leq ele(table, [i - 1]) \wedge$$
$$(0 \leq (i - 1)) \rightarrow ele(table, [i - 1]) \leq ele(table, [i])^{\uparrow}$$

Neutralization occurs allowing strong fertilization (hypothesis (124)). After fertilization we are left with residue of the form:

$$(0 \leq (i - 2)) \rightarrow ele(table, [i - 2]) \leq ele(table, [i - 1]) \ \wedge$$
$$(0 \leq (i - 1)) \rightarrow ele(table, [i - 1]) \leq ele(table, [i])$$

While the first conjunct follows by specializing hypothesis (125) (as described above), the second conjunct follows from hypothesis (132). This completes the proof of conclusion (135).

Proof of VC11 conclusion (136):

Trivial, (136) matches hypothesis (126).

4 Conclusion

Some observations:

1. The unfolding of definitions before the application of rippling is unconventional. There are 2 reasons why this strategy was adopted. Firstly, it resulted in a ripple proof that was based upon a relatively small set of general purpose wave-rules (see figure 2). Secondly, a key goal is invariant discovery, *e.g.* discovering properties such as *partitioned*. To achieve this goal, the proof patching requires access to the definition of predicates such as *ordered*, and not just application specific wave-rules that hide the low-level definitional structure.

2. An unblocking step was required for rippling that I do not think has arisen before, *i.e.* the re-ordering of nested wave-fronts.

3. It was interesting to see the need for re-annotation of a goal after strong fertilization. This corresponds to re-evaluating the target hypothesis once an initial target has been exploited.

4. An application wave-front neutralization was also observed out with the context of destructor style induction.

5. As noted earlier, one rather special purpose lemma was exploited within the proof (wave-rule (36)). It will be worth considering how such a lemma might be conjectured via proof patching (given the general purpose lemmas currently used within the proof).

6. The specialization steps that arise at various points in the proof exhibit a general pattern. This pattern could be represented as a new method or as an extension to the fertilization method.

The next step will involve focusing on how proof patching can be used to generate automatically the loop invariants required for this proof. Bill and Tommy's work on bottom-up analysis provides all of the low-level properties concerning the variables `I` and `J`. The challenging invariants are *partitioned* and *minimum*. The approach I have been working on involves the use of middle-out reasoning to combine the results of bottom-up and top-down analysis within the search for a proof. I believe this approach will provide benefits over earlier work in this area.

A SPARK code for Bubble Sort

```
package BubbleSort is

   subtype Index_Type is Integer range 0 .. 9;
   type Array_Type is array (Index_Type) of Integer;

   --# function Ordered(A: Array_Type; L, U: Index_Type) return Boolean;
   --# function Perm(A, B: Array_Type) return Boolean;

   procedure Bubble_Sort(Table: in out Array_Type);
   --# derives Table from Table;
   --# pre  true;
   --# post Ordered(Table, 0, Index_Type'Last) and
   --#      Perm(Table, Table~);

end BubbleSort;



package body BubbleSort is

   --# function Partitioned(A: Array_Type; L, M, U: Index_Type)
   --#                                                 return Boolean;

   --# function Minimum(A: Array_Type; L, U: Index_Type)
   --#                                          return Boolean;

   procedure Bubble_Sort(Table: in out Array_Type)
   is
      T: Integer;
   begin
      for I in Index_Type range 1 .. Index_Type'Last loop
      --# assert Ordered(Table, 0, I-2) and
      --#        Partitioned(Table, 0, I-2, Index_Type'Last) and
      --#        Perm(Table, Table~) and
      --#        1 <= I and I <= Index_Type'Last;
         for J in reverse Index_Type range I .. Index_Type'Last loop
         --# assert Ordered(Table, 0, I-2) and
         --#        Partitioned(Table, 0, I-2, Index_Type'Last) and
         --#        Perm(Table, Table~) and
         --#        Minimum(Table, J, Index_Type'Last) and
         --#        1 <= I and I <= Index_Type'Last and
         --#        I <= J and J <= Index_Type'Last;
            if Table(J-1) > Table(J) then
               T:= Table(J-1);
               Table(J-1):= Table(J);
               Table(J):= T;
            end if;
         end loop;
      end loop;
   end Bubble_Sort;

end BubbleSort;
```

B Proof function definitions

```
rule_family ord: ordered(A, L, U) requires [A: any, L:i, U:i].

ord(1): ordered(A, L, U) may_be_replaced_by
                  for_all(p: integer, L <= p and p < U ->
                             element(A,[p]) <= element(A,[p+1])).
```

```
rule family perm: perm(X, Y) requires [X: any, Y: any].

perm(1): perm(update(update(update(A, [I], X), [J], Y), B)
                may_be_deduced_from
                       [ perm(update(update(A, [I], Y), [J], X), B) ].
perm(2): perm(A, A) may_be_deduced.
perm(3): perm(A, B) may_be_replaced_by perm(B, A).
```

```
rule_family part: partitioned(A, L, M, U) requires [A: any, L:i, M:i, U:i].

part(1): partitioned(A, L, M, U) may_be_replaced_by
                        for_all(p: integer, L <= p and p <= M ->
                          for_all(q: integer, M < q and q <= U ->
                             element(A,[p]) <= element(A,[q]))).
```

```
rule_family min: minimum(A, L, U) requires [A: any, L:i, U:i].

min(1): minimum(A, L, U) may_be_replaced_by
                  for_all(p: integer, L < p and p <= U ->
                             element(A,[L]) <= element(A,[p])).
```

C Partial correctness VCs for BubbleSort

```
          *******************************************************
                        Semantic Analysis of SPARK Text
       SPARK95 Examiner with VC and RTC Generator Release 6.0 / 11.01
                             Demonstration Version
          *******************************************************


                        DATE : 20-AUG-2003 15:28:23.66

                       procedure BubbleSort.Bubble_Sort




For path(s) from start to assertion of line 15:

procedure_bubble_sort_1.
H1:    true .
H2:     for_all (i___1: integer, ((i___1 >= index_type__first) and (
          i___1 <= index_type__last)) -> ((element(table, [
          i___1]) >= integer__first) and (element(table, [
          i___1]) <= integer__last))) .
H3:    1 <= index_type__last .
        ->
C1:    ordered(table, 0, 1 - 2) .
C2:    partitioned(table, 0, 1 - 2, index_type__last) .
C3:    perm(table, table) .
C4:    1 <= 1 .
C5:    1 <= index_type__last .


For path(s) from assertion of line 15 to assertion of line 15:

procedure_bubble_sort_2.
H1:    ordered(table, 0, loop__1__i - 2) .
H2:    partitioned(table, 0, loop__1__i - 2, index_type__last) .
H3:    perm(table, table~) .
H4:    1 <= loop__1__i .
H5:    loop__1__i <= index_type__last .
H6:    not (loop__1__i <= index_type__last) .
H7:    not (loop__1__i = index_type__last) .
        ->
C1:    ordered(table, 0, loop__1__i + 1 - 2) .
C2:    partitioned(table, 0, loop__1__i + 1 - 2, index_type__last) .
C3:    perm(table, table~) .
C4:    1 <= loop__1__i + 1 .
C5:    loop__1__i + 1 <= index_type__last .


For path(s) from assertion of line 20 to assertion of line 15:

procedure_bubble_sort_3.
H1:    ordered(table, 0, loop__1__i - 2) .
H2:    partitioned(table, 0, loop__1__i - 2, index_type__last) .
H3:    perm(table, table~) .
H4:    minimum(table, loop__2__j, index_type__last) .
H5:    1 <= loop__1__i .
H6:    loop__1__i <= index_type__last .
H7:    loop__1__i <= loop__2__j .
H8:    loop__2__j <= index_type__last .
H9:    element(table, [loop__2__j - 1]) > element(table, [
          loop__2__j]) .
H10:   loop__2__j = loop__1__i .
H11:   not (loop__1__i = index_type__last) .
```

```
        ->
C1:    ordered(update(update(table, [loop__2__j - 1], element(
           table, [loop__2__j])), [loop__2__j], element(
           table, [loop__2__j - 1])), 0, loop__1__i + 1 - 2) .
C2:    partitioned(update(update(table, [loop__2__j - 1], element(
           table, [loop__2__j])), [loop__2__j], element(
           table, [loop__2__j - 1])), 0, loop__1__i + 1 - 2,
           index_type__last) .
C3:    perm(update(update(table, [loop__2__j - 1], element(
           table, [loop__2__j])), [loop__2__j], element(
           table, [loop__2__j - 1])), table~) .
C4:    1 <= loop__1__i + 1 .
C5:    loop__1__i + 1 <= index_type__last .


procedure_bubble_sort_4.
H1:    ordered(table, 0, loop__1__i - 2) .
H2:    partitioned(table, 0, loop__1__i - 2, index_type__last) .
H3:    perm(table, table~) .
H4:    minimum(table, loop__2__j, index_type__last) .
H5:    1 <= loop__1__i .
H6:    loop__1__i <= index_type__last .
H7:    loop__1__i <= loop__2__j .
H8:    loop__2__j <= index_type__last .
H9:    not (element(table, [loop__2__j - 1]) > element(
           table, [loop__2__j])) .
H10:   loop__2__j = loop__1__i .
H11:   not (loop__1__i = index_type__last) .
        ->
C1:    ordered(table, 0, loop__1__i + 1 - 2) .
C2:    partitioned(table, 0, loop__1__i + 1 - 2, index_type__last) .
C3:    perm(table, table~) .
C4:    1 <= loop__1__i + 1 .
C5:    loop__1__i + 1 <= index_type__last .


For path(s) from assertion of line 15 to assertion of line 20:

procedure_bubble_sort_5.
H1:    ordered(table, 0, loop__1__i - 2) .
H2:    partitioned(table, 0, loop__1__i - 2, index_type__last) .
H3:    perm(table, table~) .
H4:    1 <= loop__1__i .
H5:    loop__1__i <= index_type__last .
H6:    loop__1__i <= index_type__last .
        ->
C1:    ordered(table, 0, loop__1__i - 2) .
C2:    partitioned(table, 0, loop__1__i - 2, index_type__last) .
C3:    perm(table, table~) .
C4:    minimum(table, index_type__last, index_type__last) .
C5:    1 <= loop__1__i .
C6:    loop__1__i <= index_type__last .
C7:    loop__1__i <= index_type__last .
C8:    index_type__last <= index_type__last .


For path(s) from assertion of line 20 to assertion of line 20:

procedure_bubble_sort_6.
H1:    ordered(table, 0, loop__1__i - 2) .
H2:    partitioned(table, 0, loop__1__i - 2, index_type__last) .
H3:    perm(table, table~) .
H4:    minimum(table, loop__2__j, index_type__last) .
H5:    1 <= loop__1__i .
H6:    loop__1__i <= index_type__last .
```

```
H7:    loop__1__i <= loop__2__j .
H8:    loop__2__j <= index_type__last .
H9:    element(table, [loop__2__j - 1]) > element(table, [
           loop__2__j]) .
H10:   not (loop__2__j = loop__1__i) .
        ->
C1:    ordered(update(update(table, [loop__2__j - 1], element(
           table, [loop__2__j])), [loop__2__j], element(
           table, [loop__2__j - 1])), 0, loop__1__i - 2) .
C2:    partitioned(update(update(table, [loop__2__j - 1], element(
           table, [loop__2__j])), [loop__2__j], element(
           table, [loop__2__j - 1])), 0, loop__1__i - 2,
           index_type__last) .
C3:    perm(update(update(table, [loop__2__j - 1], element(
           table, [loop__2__j])), [loop__2__j], element(
           table, [loop__2__j - 1])), table~) .
C4:    minimum(update(update(table, [loop__2__j - 1], element(
           table, [loop__2__j])), [loop__2__j], element(
           table, [loop__2__j - 1])), loop__2__j - 1,
           index_type__last) .
C5:    1 <= loop__1__i .
C6:    loop__1__i <= index_type__last .
C7:    loop__1__i <= loop__2__j - 1 .
C8:    loop__2__j - 1 <= index_type__last .


procedure_bubble_sort_7.
H1:    ordered(table, 0, loop__1__i - 2) .
H2:    partitioned(table, 0, loop__1__i - 2, index_type__last) .
H3:    perm(table, table~) .
H4:    minimum(table, loop__2__j, index_type__last) .
H5:    1 <= loop__1__i .
H6:    loop__1__i <= index_type__last .
H7:    loop__1__i <= loop__2__j .
H8:    loop__2__j <= index_type__last .
H9:    not (element(table, [loop__2__j - 1]) > element(
           table, [loop__2__j])) .
H10:   not (loop__2__j = loop__1__i) .
        ->
C1:    ordered(table, 0, loop__1__i - 2) .
C2:    partitioned(table, 0, loop__1__i - 2, index_type__last) .
C3:    perm(table, table~) .
C4:    minimum(table, loop__2__j - 1, index_type__last) .
C5:    1 <= loop__1__i .
C6:    loop__1__i <= index_type__last .
C7:    loop__1__i <= loop__2__j - 1 .
C8:    loop__2__j - 1 <= index_type__last .


For path(s) from start to finish:

procedure_bubble_sort_8.
H1:    true .
H2:     for_all (i___1: integer, ((i___1 >= index_type__first) and (
           i___1 <= index_type__last)) -> ((element(table, [
           i___1]) >= integer__first) and (element(table, [
           i___1]) <= integer__last))) .
H3:    not (1 <= index_type__last) .
        ->
C1:    ordered(table, 0, index_type__last) .
C2:    perm(table, table) .


For path(s) from assertion of line 15 to finish:
```

```
procedure_bubble_sort_9.
H1:    ordered(table, 0, loop__1__i - 2) .
H2:    partitioned(table, 0, loop__1__i - 2, index_type__last) .
H3:    perm(table, table~) .
H4:    1 <= loop__1__i .
H5:    loop__1__i <= index_type__last .
H6:    not (loop__1__i <= index_type__last) .
H7:    loop__1__i = index_type__last .
        ->
C1:    ordered(table, 0, index_type__last) .
C2:    perm(table, table~) .
```

For path(s) from assertion of line 20 to finish:

```
procedure_bubble_sort_10.
H1:    ordered(table, 0, loop__1__i - 2) .
H2:    partitioned(table, 0, loop__1__i - 2, index_type__last) .
H3:    perm(table, table~) .
H4:    minimum(table, loop__2__j, index_type__last) .
H5:    1 <= loop__1__i .
H6:    loop__1__i <= index_type__last .
H7:    loop__1__i <= loop__2__j .
H8:    loop__2__j <= index_type__last .
H9:    element(table, [loop__2__j - 1]) > element(table, [
           loop__2__j]) .
H10:   loop__2__j = loop__1__i .
H11:   loop__1__i = index_type__last .
        ->
C1:    ordered(update(update(table, [loop__2__j - 1], element(
           table, [loop__2__j])), [loop__2__j], element(
           table, [loop__2__j - 1])), 0, index_type__last) .
C2:    perm(update(update(table, [loop__2__j - 1], element(
           table, [loop__2__j])), [loop__2__j], element(
           table, [loop__2__j - 1])), table~) .


procedure_bubble_sort_11.
H1:    ordered(table, 0, loop__1__i - 2) .
H2:    partitioned(table, 0, loop__1__i - 2, index_type__last) .
H3:    perm(table, table~) .
H4:    minimum(table, loop__2__j, index_type__last) .
H5:    1 <= loop__1__i .
H6:    loop__1__i <= index_type__last .
H7:    loop__1__i <= loop__2__j .
H8:    loop__2__j <= index_type__last .
H9:    not (element(table, [loop__2__j - 1]) > element(
           table, [loop__2__j])) .
H10:   loop__2__j = loop__1__i .
H11:   loop__1__i = index_type__last .
        ->
C1:    ordered(table, 0, index_type__last) .
C2:    perm(table, table~) .
```