CS-Note* 51

Tommy Ingulfsen
July 8, 2003

# Exploring AUTOGAP heuristics for supporting the proof of bubble sort

## 1 In the beginning there were problems...

This draft note is a first try at the problem of creating new heuristics in support of sorting algorithms. The main issue we discuss is how to create meta-data that conveys the ordering of elements in array partitions. Presently, AUTOGAP does not have any meta-data heuristics that help us decide what values individual array elements have. This is a weakness we seek to alleviate, initially focusing on the bubble sort algorithm. Inspiration is taken from the CS-Note entitled "Bubble Sort".

## 2 Proof-problems

This is the bubble sort algorithm in a form parseable by AUTOGAP:

```
procedure Bubble_Sort(Table: in out Array_Type)
is
   subtype Index is Integer range 0..9;

   T: Integer;
   I: Index;
   J: Index;
begin

   I := 1;
   loop
      --# assert true;

      exit when I = Index'Last;

      J := Index'Last;
      loop
         --# assert true;

         exit when J = I;

         if Table(J-1) > Table(J) then
            T:= Table(J-1);
            Table(J-1):= Table(J);
            Table(J):= T;
         end if;

         J := J - 1;
      end loop;

      I := I + 1;
   end loop;
end Bubble_Sort;
```

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

There is a nested loop, the outer one being controlled by the loop counter `I`, while `J` controls the inner loop. This version of bubble sort orders the array in ascending order, so that upon completion of the algorithm `Table` has the largest element last and the smallest first.

For the moment AUTOGAP generates two facts pertaining to the array `Table`, namely that `J` is an index variable and that there is a partition in `Table` from `100` (last element) to `J`. Unfortunately, when we combine the information in the meta-data file generated by AUTOGAP and the VCs, we are still unable to prove the code correct. For reference, these are all the meta-data properties generated for the code above:

```
initial_val('J', integer(9)).
initial_val('T', element(variable('Table'),variable('J')-integer(1))).
initial_val('I', integer(1)).
count_var('I', 12).
count_var('J', 18).
mono_inc(value('I',now), greaterthan, value('I',previous), line(12)).
mono_dec(value('J',now), lessthan, value('J',previous), line(18)).
bound('I', 9, upper).
index_var('J', 'Table').
array_partition('Table', 'J', integer(9), 18).
invar(12, variable('I')<=integer(9)).
invar(12, variable('I')>=integer(1)).
invar(18, variable('J')<=integer(9)).
invar(18, variable('J')>=variable('I')).
```

### 2.1 Crucial information

There are several pieces of knowledge about the code missing. Here we look at two of these, involving the values of individual array elements and values of partitions. In order to prove the partial correctness of our bubble sort we need to know that *partitioned(table, 0, J-1, Index'Last)* and *minimum(table, J, Index'Last)* are both true. We define these properties as

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

minimum(A, L, U) : for_all(p: integr, L <= p and p <= U ->
                       element(A, [L]) <= element(A, [p]))
```

So in this case, the required knowledge is that all elements below `i-1` are less than or equal to those including and above `i-1` (*partitioned)* and that element `j` is the smallest in the partition from `J` and upwards (*minimum*).

## 3 Thoughts on new heuristics

Deriving the *partitioned* and *minimum* properties exactly as above is tricky if using the bottom-up approach adopted by AUTOGAP. So, at least to start with, we limit our ambitions to derive simpler properties that will give NUSPADE “hints” rather than complete answers.

### 3.1 *minimum*

AUTOGAP observes that `J` is a monotonically decreasing variable. Looking at the code it should be possible to come up with meta-data that is close to the *minimum* knowledge needed.

The clue is in the swapping operation. `J` starts off pointing to the final element and then decreases. Each time round the inner loop `Table(J)` is compared to `Table(J-1)` to see if `Table(J-1)` is larger than `Table(J)`. If the test succeeds the elements are swapped, otherwise they are ordered correctly and no action is taken. Therefore, just after the if-statement, we can conclude that the value of `Table(J)` is less than or equal to `Table(J-1)`. Combining this conclusion with the knowledge that `J` is monotonically decreasing we can arrive at something like

```
minimum(Table, J, 9)
```

expressed in meta-data form.

### 3.2 *partitioned*

This information is harder to arrive at using bottom-up analaysis. Using the current heuristics, AUTOGAP finds an array partition from `J` to `9`. We need to find a way of linking this partition to the variable `I` in order to find the partition below `J`.

Firstly, we note that `I` is the lower bound on `J` in the inner loop. Since `I` changes we may conclude that there must be a partition below this lower bound as well as the partition AUTOGAP finds now. However, deriving information on the actual values of the elements in this partition is trickier (arriving at the ordering property that *partitioned* implies). Intuitively, some ordering property should exist since we have found that the elements from `J` and upwards are ordered and because `I` controls the value of `J`. Let us look at a picture [1]. - then say minimum is lowest, then bubble down second smallest to first+1 etc

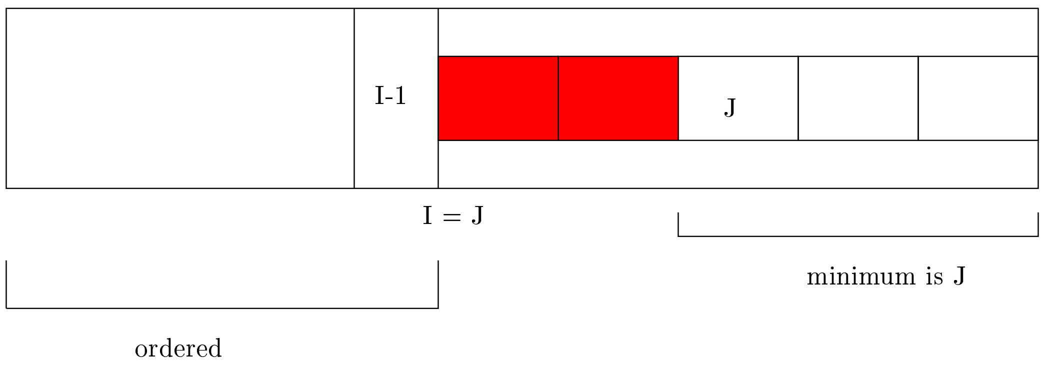


Figure 1: Two main partitions are identified. Everything up to `I-1` is ordered. In the upper partition `J` indexes the minimum element. The picture shows a snapshot of the inner loop as `J` works its way down towards `I`. In between `J` and `I` the elements have no ordered or minimum property.

The bubble sort algorithm operates by bubbling the smallest element down from its original location to the becoming the first element of the array. Then it bubbles the second smallest element down and so on. In the first round of the outer loop, `J` ranges from the final element to `I` (the first one). In the second round `J` again ranges from the final element to `I`, and so on. We can therefore deduce that the **`minimum`** property we derived is transitive in this case. That is,

```
minimum(Table, I, 9)
```

This is how far our investigation has got at the moment.

## 4 The geeks want to program

In order to implement these thoughts one would start with *minimum*. As can be seen from the code it is necessary to recognise that elements in the array are swapped in such a way that the smaller element

[1]Thanks to Bill J Ellis for displaying his artistic talents.

ends up at the lowest index. This is deduced by identifying the if-condition and studying the swapping statements that succeed it. Neither of these is particularly hard.

Implementing the discovery made for the *partitioned* predicate could be done by extending the property for *minimum* with respect to `J` to cover `I` also. The link is present in the code, as `I` is the lower bound of `J`. Presently, AUTOGAP does not discover this bound, but amending the existing heuristics to cover this should be straightforward.

## 5 In the end, questions arose

This note is just a draft, intended for discussion. There are many weaknesses in the above outline.

Firstly, only 1 example has been considered. We must take care to avoid making a specialised heuristic for bubble sort, but make the new heuristics general enough to work for many examples. This might entail weakening the methods above in order to extend the range of applicability.

Secondly, the properties found above may not be enough to prove bubble sort, since they're only a hint to the theorem prover of what to do.

In the immediate future we seek to extend this new course for AUTOGAP by finding further examples.