CS-Note* 52

Tommy Ingulfsen
July 8, 2003

Exploring AUTOGAP heuristics for supporting the proof of bubble sort: Take Two

1 Intro

In this draft note we further explore the questions raised in CS-Note 51, which deals with supporting the formal verification of bubble sort. There we concluded that it is indeed feasible to modify AUTOGAP in such a way that it generates properties like `minimum(Table, J, 9)` and similarly `minimum(Table, I, 9)`.

The main problem, however, lies in discovering the meta-data that allows us to conclude that a `partitioned` property applies to the array that is sorted. In this note, we discuss in more detail how AUTOGAP could be improved to discover algorithmic properties for aiding the proof. Everything up to section 3.2 is a copy of CS-Note 50. After that we explore some new ideas.

We are moving away from the restrictions of AUTOGAP now. From now on we look at the bigger picture. How can the bottom-up analysis of AUTOGAP be combined with top-down methods? Clearly this is the way forward for us, combining the common techniques with our new ones. Very little in this note has been implemented, and so it only represents a thought experiment. However, we present the argument in such a way that it is obvious that AUTOGAP could be extended to cover our new ideas.

2 Recap

This is the bubble sort algorithm:

```
(Declaration in specification file)

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~);

(Code)

procedure Bubble_Sort(Table: in out Array_Type)
is
    T: Integer;
begin
    for I in Index_Type range 1..Index_Type'Last loop
        for J in reverse Index_Type I..Index_Type'Last loop
        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;
```

*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).

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 9 (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: integer, 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 use this to come up with meta-data that is close to the minimum knowledge needed.

The clue is in the swapping operation. 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 already correctly ordered 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 and denotes the left end of a partition in `Table`, we can arrive at something like

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

expressed in meta-data form, with the same meaning as the `minimum` property from section 2.1.

3.2 *partitioned*

This information is harder to arrive at using bottom-up analysis. 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` is monotonically increasing 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].

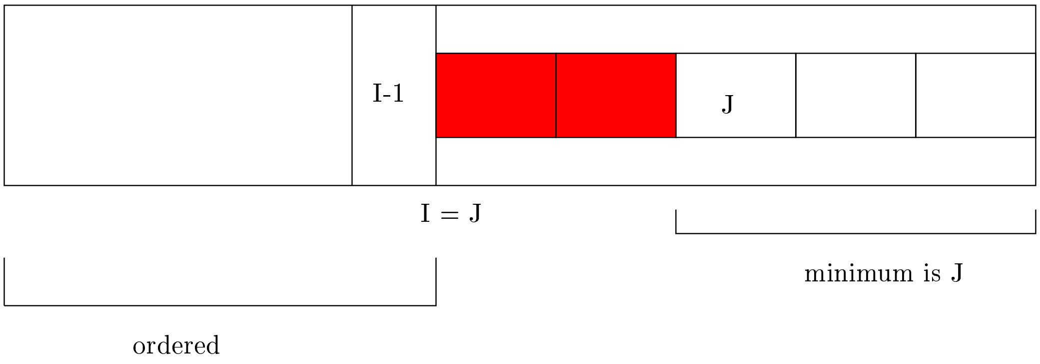

Figure 1: Two main partitions are identified. Everything up to `I-1` is ordered. In the upper partition `J` indexes the minimum element. The picture shows a snapshot of the inner loop as `J` works its way down towards `I`. In between `J` and `I` the elements have no ordered or minimum property.

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 the first element (since `I` is initialised to point to the first element. 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)
```

3.3 Linking `I` to `J` and discovering an order

Sorting algorithms impose an order upon the elements of an array, in our case this is expressed by the postcondition, which states that upon completion the array is ordered ascendingly. There is a way in which we can deduce that such an ordering is being imposed upon this array.

Presently, AUTOGAP finds a partition in `Table` from the last element, growing leftwards towards `J`. Above we observed that there are in fact two partitions, the second being bounded above by `I`. The reason

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

why this last partition is not discovered by AUTOGAP at this stage is that `I` is not an index variable. But `J` is bounded from below by `I`, and `I` is monotonically decreasing. Therefore, there must be a partition indexed by `I` in `Table` that grows from `I`'s initial value.

At the end of each round of the inner loop, `J` always points to the smallest value in the partition indexed by `J`. This is conveyed by the `minimum(Table, J, 9)` property that we derived above, which lead to our conclusion that `minimum(Table, I, 9)` is also true. We now have the information we need to make a statement about the ordering of the array.

The three pieces of knowledge - `mono_inc(I)`, `partition(Table, integer(0), I, 18)` and `minimum(Tabl` - allow us to conclude that *an ascending order is being imposed on the array.* Expressed as meta-data,

```
ordering(Table, integer(0), I, ascending)
```

We can take this to mean that at any time, the elements in array `Table` from the first element towards `I` are ordered ascendingly. So it's very close to the `partitioned` property discussed in section 2.1.

4 Conclusion and future developments

We have outlined how AUTOGAP can deduce that an ascending order is being imposed on the array, and how we can detect the swapping of elements to conclude that `J` (and also `I`) points to a minimum element in the array.

Here are some ideas for how further progress can be made:

- Is it possible to extend the reasoning about order here to searching algorithms? We could potentially exploit that when searching, no assignments are made to array elements, so when a (monotonically increasing) variable is compared to array elements in a loop can we deduce that an ordering already exists?
- Find a way to actually implement Katz and Manna's ideas in AUTOGAP, especially their ideas on generalisation look promising.
- Exploit the pre- and postconditions, combining bottom-up and top-down techniques.
- Is it possible to find an invariant for expanding partitions, using the post- or preconditions?
- Classify subprograms according to what kind of array operation they perform (identify single elements, modify subranges only , identify ranges, modify complete ranges...).
- Find "work still to do"-partitions for other than sorting algorithms, e.g. searching, swapping equal length segments, finding shortest prefix, majority vote etc.
- Find a counter example of `ordered` so as to derive `partitioned`, since `partitioned` is a kind of poor `ordered` (`ordered` claims ascending for entire array, `partitioned` claims ascending for partition only).
- Code alternative versions of bubble sort to see how our techniques so far can be modified to work on more than one example.