Acta Univ. Sapientiae, Informatica, **4,** 2 (2012) 247–259

Abstract levels of programming theorems

Tibor GREGORICS
Eötvös Loránd University, Faculty of Informatics
Budapest
email: gt@inf.elte.hu

Abstract. In this paper three abstract levels of programming theorems are introduced. These levels depend on the form of the sequence of the elements that are produced by a programming theorem. We are going to investigate the difference between the solutions of the same problem if these solutions are derived from altering abstract levels of the same programming theorem. One of the famous programming theorems, the maximum selection, is chosen as an example, all of its three versions will be presented, and their usage will be shown in a case study.

1 Introduction

Programming theorems are used frequently to plan algorithms. A programming theorem is a pattern, a problem-program (task-algorithm) pair where a program can solve the problem. All the tasks that are similar to the problem of a theorem can be solved on the basis of the algorithm of the theorem. Programming theorems (summation, counting, maximum selection, and linear search, etc.) [2, 4] are well-known by all programmers but only a few of them know that these theorems can be expressed in multiple ways. Most programmers consider programming theorems as sample solutions. When they want to solve a task that is similar to the problem of a theorem, they try to repeat the same activities that created the program of the theorem. Thus programming theorems support their algorithmic way of thinking that is used to construct

Computing Classification System 1998: F.3.1
Mathematics Subject Classification 2010: 68N30
Key words and phrases: analogous programming, programming theorem, enumerator, object-oriented programming

the algorithm of their task. However, there exists another method to create programs. This is derivation [1, 4]. Starting from the exact comparison of the task to be solved and of the problem of the candidate programming theorem, the program of the theorem has to be updated according to the differences between the task and the problem. Thus, without algorithmic way of thinking, the program by which the new task is solved can be produced almost automaticly. This method is faster and guarantees the correctness of the algorithm but it requires the formal description of the task. The whole of this paper can be articulated from this single point of view, i.e. when algorithms are planned with derivation.

The quality (efficiency and compactness) and, very often, the success of the solution depend on the degree of the universality of the programming theorem. According to the way that the problem of the theorem is generalized, different versions of the theorem can be obtained. It is obvious that a good programming theorem should be adequately universal so that the class of the tasks to be solved is wide enough. But the theorem must preserve some specialty in order that it can be identified in a simple way. The reason for this, for example, is that counting is a separate theorem; nevertheless, it is a special case of summation.

One of the common properties of programming theorems is that they process a sequence of elementary values. The way these values are produced may differ. Programming theorems may be distinguished according to these three levels. Henceforth these levels are going to be defined, the versions of maximum selection are going to be fully given, and various solutions of the same task are going to be produced by using different levels of the same programming theorem.

2 Different forms of the sequence of elements are processed

A sequence of elementary values can be placed into a container such as a sequential file or a linked list. The most widely-known container, however, is the one-dimensional array. Most programmers use the programming theorems processing the elements of an array. This is the lowest level of the programming theorems.

A higher level is the one when an appropriate function gives the elements that must be proccessed. The domain of this function is always an interval of integers. (Hereafter $[\mathfrak{n}..\mathfrak{m}]$ denotes the integer interval $[\mathfrak{n}, \mathfrak{m}] \cap \mathbb{Z}$ for all

$n, m \in \mathbb{Z}$.) This function is more universal than an array: each array can be interpreted as a function over integer interval.

The third level is when the elements are provided by a special activity, an enumeration. The enumerator is an object that disposes the four enumeration operators: `First(), Next(), End(), Current()` [5]. These operators permit iterating the elements that must be processed. The elements of an array can be iterated like the proper divisors of a natural number. This point of view gives more universal definitions of programming theorems.

3 Different levels of the programming theorems

Now the maximum selection is going to be defined in three different forms. Other programming theorems can be defined in this same way.

3.1 Maximum selection in an array

An array over the non-empty integer interval $[m..n]$ is given where the elements on the array form a totally ordered set, set H (notation: $H^{m..n}$). The greatest element of the array is sought, and one of the indexes should also be given where this element occurs.

Specification:

In the formal specification used below the letter A denotes the state space that enumerates the variables of the problem with their types. The letter Q is the precondition and R is the postcondition of the problem. If v is a variable of the state space, then the notation v' is an arbitrary, initial value of the variable v. The variable i is the index variable of the `for` statement.

$$\begin{aligned} A &= (x : H^{m..n}, max : H, ind : \mathbb{Z}) \\ Q &= ((x = x') \wedge (n \leq m)) \\ R &= ((x = x') \wedge (ind \in [m..n]) \wedge (max = x[ind] = \mathop{MAX}_{i=m}^{n} x[i])) \end{aligned}$$

Algorithm:

```
max, ind := x[m], m;
for i = m+1 ... n do
    if x[i] > max then
        max, ind := x[i], i;
    endif
endfor
```

3.2 Maximum selection over interval

There are many problems, the solution of which can not be derived from a programming theorem in an array but from the same programming theorem over an interval. For example, suppose the average temperatures of successive days are fixed in an array (its elements are indexed by the integer interval $[1..n]$) and the neighboring pairs of temperature must be counted where the first value of the pair is under freezing point and the second one is above it. This task cannot be derived from the counting in an array because the elements that must be checked in the counting are not elements of an array. These elements are logical values provided by a logical function (condition) that is defined over the integer interval $[2..n]$, and these values depend on the pairs of the original array of the task. Sometimes there is no array at all in a problem. For example, if the proper divisors of a given natural number have to be counted, then the function $f(i) = i$ *can divide* n (which is defined over the integer interval $[2..n/2]$) should be checked. Anyway programming theorems on array are looked upon as special cases of programming theorems over interval because each array can be interpreted as a function over integer interval.

There is a non-empty integer interval $[m..n]$ and a function $f : [m..n] \to H$, where H is a totally ordered set. The greatest value of the function is sought, and one of its arguments should also be given.

Specification:

$$\begin{aligned} A &= (m : \mathbb{Z}, n : \mathbb{Z}, max : H, ind : \mathbb{Z}) \\ Q &= ((m = m') \wedge (n = n') \wedge (n \leq m)) \\ R &= ((m = m') \wedge (n = n') \wedge (ind \in [m..n]) \\ &\quad \wedge (max = f(ind) = \mathop{MAX}_{i=m}^{n} f(i))) \end{aligned}$$

The postcondition can be written in a shorter form. In this notation, MAX is a multi-valued function mapping from an interval to H and $\mathbb{Z}$.

$$R = ((m = m') \wedge (n = n') \wedge ((max, ind) = \mathop{MAX}_{i=m}^{n} f(i)))$$

Algorithm:

```
max, ind := f(m), m;
for i = m+1 ... n do
    if f(i) > max then
        max, ind := f(i), i;
    endif
endfor
```

3.3 Maximum selection on enumerator

There is an enumerator that can iterate the elements of a finite non-empty sequence which belongs to set E ($\mathit{enor}(\mathsf{E})$ notates the type of this enumerator). A function is given $\mathsf{f} : \mathsf{E} \to \mathsf{H}$ where H is a totally ordered set. The greatest value over the values mapped from the elements of the enumerator by the function f is sought, and one element should also be given where this value occurs.

Specification:

$$\begin{array}{lcl} \mathsf{A} & = & (\mathsf{t} : \mathit{enor}(\mathsf{E}), \mathit{max} : \mathsf{H}, \mathit{ind} : \mathsf{E}) \\ \mathsf{Q} & = & ((\mathsf{t} = \mathsf{t}') \wedge (|\mathsf{t}| \neq 0)) \\ \mathsf{R} & = & ((e \in \mathsf{t}') \wedge (\mathit{max} = \mathsf{f}(\mathit{ind}) = \underset{e \in \mathsf{t}'}{\mathrm{MAX}}\, \mathsf{f}(e))) \end{array}$$

Algorithm:

```
t.First();
max, ind := f(t.Current()), t.Current();
t.Next();
while ¬t.End() do
    if f(t.Current()) > max then
        max, ind := f(t.Current()), t.Current();
    endif
endwhile
```

4 Case study

Let us solve the following problems. There is a plan where n points are given. Which is the greatest distance between pairs of points?

The points on the plan can be represented by their coordinates if there is a fixed coordinate system. These coordinates are saved in two one-dimensional arrays: x and y. The coordinates of the i^{th} point are $x[i]$ and $y[i]$. The distance between the i^{th} and j^{th} points is $\sqrt{(x[i]-x[j])^2 + (y[i]-y[j])^2}$ but the greatest distance is not wanted, so it will be enough to use their squares.

Let $d(i,j)$ denote the square distance $(x[i]-x[j])^2 + (y[i]-y[j])^2$ $(i = 1..n, j = 1..n)$. These values can be arranged in an $n \times n$ symmetrical matrix. Our task is to select the maximal element of the lower triangular part of this matrix without its diagonal.

It seems that the solution can be created with a maximum selection but the theorem of maximum selection can investigate only one-dimensional forms

and here the values of $d(i,j)$ $\big(i,j \in [1..n]\big)$ are in a two-dimensional shape.

4.1 First solution: numerous maximum selections in an array

The rows of the virtual lower triangular matrix can be seen as many one-dimensional arrays with increasing size. If the greatest element is selected from every row and put into an auxiliary array, then a new maximum selection in this array can solve the original problem. More precisely, this auxiliary array indexed by the interval $[2..n]$ (denoted by z) should store value-index pairs ($\mathrm{rec}(m:\mathbb{R}, k:\mathbb{N})$) and the i^{th} element of this array can show the greatest value and its index in the i^{th} row. By definition $z[i] > z[j]$ if $z[i].m > z[j].m$.

Specification:

$$
\begin{aligned}
A &= (x, y : \mathbb{R}^n, z : \mathrm{rec}(m:\mathbb{R}, k:\mathbb{N})^{2..n}, max : \mathbb{R}, ind, jnd : \mathbb{N}) \\
Q &= ((x = x') \wedge (y = y') \wedge (n \geq 2)) \\
R &= ((x = x') \wedge (y = y') \wedge (\forall i \in [2..n] : z[i] = \underset{j=1}{\overset{i-1}{MAX}}\, d(i,j)) \\
&\quad \wedge ((max, jnd), ind) = \underset{i=2}{\overset{n}{MAX}}\, z[i]))
\end{aligned}
$$

This problem can be solved with n maximum selections. The first n–1 maximum selections fill an auxiliary array z and the last one selects the maximal elements of this array.

Each of the first n–1 maximum selections works in one of the rows of the virtual lower triangular matrix. The i^{th} row of this matrix is considered as an array indexed by the interval $[1..i–1]$, and the j^{th} element of this array is the value $d(i,j)$. Obviously these maximum selections can be derived from the programming theorem in an array. The assignment $z[i] := (d(i,j), j)$ is a shorter form of the assignments $z[i].m, z[i].k := d(i,j), j$

```
for i = 2...n do
    z[i] := (d(i,1), 1);
    for j = 2...i–1 do
        if d(i,j) > z[i].m then
            z[i] := (d(i,j), j);
        endif
    endfor
endfor
```

The last maximum selection is also derived from the programming theorem in an array because it uses the auxiliary array z. The variables i and j are the

index-variables of the for statements.

```
(max, jnd), ind := z[2], 2;
for i = 3...n do
    if z[i].m > max then
        (max, jnd), ind := z[i], i;
    endif
endfor
```

Finally, the whole solution is repeated with minor modifications. Hence the second row of the lower triangular matrix contains only one element, this is the z[2] which can be calculated without maximum selection: z[2] := (d(2, 1), 1). Moreover the assigment (max, jnd) := z[i] is equivalent to the assigment max, jnd := z[i].m, z[i].k, and the value of the variable jnd is enough to be set at the end of the algorithm. At the end the local auxiliary variable s is introduced to contain the value of d(i, j).

```
z[2] := (d(2, 1), 1);
for i = 3...n do
    z[i] := (d(i, 1), 1);
    for j = 2...i–1 do
        s := d(i, j);
        if s > z[i].m then
            z[i] := (s, j);
        endif
    endfor
endfor
max, ind := z[2].m, 2;
for i = 3...n do
    if z[i].m > max then
        max, ind := z[i].m, i;
    endif
endfor
jnd := z[ind].k;
```

Note that the auxiliary array can be eliminated from this program if the two outsider loops (where i goes from 3 to n) are combined. However, this solution can be produced in a simpler way if a more generalized programming theorem is used.

4.2 Second solution: several maximum selections over interval

Let us follow the previous line of thought but instead of the auxiliary array, a function is going to be defined which gives the greatest element and its index from every row of the virtual lower triangular matrix.

Specification:

$$\begin{aligned} A &= (x, y : \mathbb{R}^n, max : \mathbb{R}, ind, jnd : \mathbb{N}) \\ Q &= ((x = x') \wedge (y = y') \wedge (n \geq 2)) \\ R &= ((x = x') \wedge (y = y') \wedge ((max, jnd), ind) = \mathop{MAX}_{i=2}^{n} g(i))) \end{aligned}$$

where $g : [2..n] \to \mathbb{R} \times \mathbb{N}$ and $g(i) = \mathop{MAX}_{j=1}^{i-1} d(i, j))$

This problem can be solved with the maximum selection over interval 2..n with the function g. By definition, $g(i) > g(j)$ if $g(i)_1 > g(j)_1$. The variable i is the index-variable of the for statement, the variable m and k are local auxiliary variables.

```
(max, jnd), ind := g(2), 2;
for i = 3 ... n do
    (m, k) := g(i);
    if m > max then
        max, jnd, ind := m, k, i;
    endif
endfor
```

The subproblem $(m, k) := g(i)$ is also a maximum selection but its interval is [1..i–1] and its function is $d(i, j)$. The variable j is the index-variable of the for statement. The variable s is a local auxiliary variable. The main program calls this subprogram twice.

```
m, k := d(i, 1), 1;
for j = 2 ... i–1 do
    s := d(i, j);
    if s > m then
        m, k := s, j;
    endif
endfor
```

We can compare this solution with the previous one if this program is combined. The value of $g(2)$ can be calculated as $d(2, 1)$ and 1; hence, the initial assignment will be $max, jnd, ind := d(2, 1), 1, 2$.

```
max, jnd, ind := d(2,1), 1, 2;
for i = 3...n do
    m, k := d(i,1), 1;
    for j = 2...i–1 do
        s := d(i,j);
        if s > m then
            m, k := s, j;
        endif
    endfor
    if m > max then
        max, jnd, ind := m, k, i;
    endif
endfor
```

4.3 Third solution: one maximum selection over interval

Let us imagine that the elements of the lower triangular matrix are in a sequence. The first element of this sequence is the single element of the second row (it is indexed with $(2,1)$), the 2^{nd} element is the one on the $(3,1)$ position, the 3^{rd} is the $(3,2)$, 4^{th} is $(4,1)$] and so on. The size of this sequence is $n(n–1)/2$. How can the i^{th} element of this sequence be found in the matrix?

It is easy to see that the $(i,j)^{th}$ element of the lower triangular matrix $(j < i)$ is the $((i–1)(i–2)/2 + j)^{th}$ element of the sequence because there are $(i–1)(i–2)/2$ elements in front of the i^{th} row in the lower triangular matrix. But where can the k^{th} element of the sequence be found in the matrix?

Lemma 1 *The* k^{th} *element of the sequence is the* $(i,j)^{th}$ *element of the matrix where* $j = 2k-(i-1)(i-2)$ *and if* $2k > \lceil\sqrt{2k}\rceil(\lceil\sqrt{2k}\rceil-1)$*, then* $i = \lceil\sqrt{2k}\rceil+1$*, otherwise* $i = \lceil\sqrt{2k}\rceil$*.*

Proof. Because of $k = (i-1)(i-2)/2 + j(j < i)$, we get

$$(i-1)(i-2) < 2k \leq i(i-1). \tag{1}$$

It follows that $(i-2) < \sqrt{2k} < i$, so the value $\lceil\sqrt{2k}\rceil$ (upper integer part) may be i or $i-1$. If $2k > \lceil\sqrt{2k}\rceil(\lceil\sqrt{2k}\rceil-1)$, then $\lceil\sqrt{2k}\rceil = i-1$ because supposing $\lceil\sqrt{2k}\rceil = i$ we get $2k > i(i-1)$ that is a contradiction of (1). If $2k \leq \lceil\sqrt{2k}\rceil(\lceil\sqrt{2k}\rceil-1)$, then $i = \lceil\sqrt{2k}\rceil$ because supposing $\lceil\sqrt{2k}\rceil = i-1$, we get $2k \leq (i-1)(i-2)$ that is also a contradiction. □

Based on this lemma, the following function can be defined:

$h : [1..n(n–1)/2] \to \mathbb{N} \times \mathbb{N}$

$$h(k) = \begin{cases} (\lceil\sqrt{2k}\rceil + 1, 2k\lceil\sqrt{2k}\rceil(\lceil\sqrt{2k}\rceil - 1)) & \text{if} \quad 2k > (\lceil\sqrt{2k}\rceil - 1)\lceil\sqrt{2k}\rceil \\ (\lceil\sqrt{2k}\rceil, 2k(\lceil\sqrt{2k}\rceil - 1)(\lceil\sqrt{2k}\rceil - 2) & \text{if} \quad 2k \leq (\lceil\sqrt{2k}\rceil - 1)\lceil\sqrt{2k}\rceil \end{cases}$$

Now the problem can be re-specified. We introduce the auxiliary variable knd.

$$\begin{aligned} A &= (x, y : \mathbb{R}^n, max : \mathbb{R}, ind, jnd, knd : \mathbb{N}) \\ Q &= ((x = x') \wedge (y = y') \wedge (n \geq 2)) \\ R &= ((x = x') \wedge (y = y') \wedge ((max, knd) = \mathop{MAX}_{k=1}^{n(n-1)/2} d(h(k))) \\ &\quad \wedge((ind, jnd) = h(knd))) \end{aligned}$$

This problem can be derived to the maximum selection over the interval $[1..n(n–1)/2]$ with the function h. In the initial assignment, the expression $d(h(1))$ can be changed to $d(2, 1)$. The variable k is the index-variable of the for statement, the variable s is a local auxiliary variable.

```
max, knd := d(2, 1), 1;
for k = 2 ... n(n–1)/2 do
    if d(h(k)) > max then
        max, knd := s, k;
    endif
endfor
(ind, jnd) := h(knd)
```

Let us take some minor modifications. The knd auxiliary variable can be eliminated but the local auxiliary variables i, j and s are introduced.

```
max, ind, jnd := d(2, 1), 2, 1;
for k = 2 ... n(n–1)/2 do
    (i, j) := h(k);
    s := d(i, j);
    if s > max then
        max, ind, jnd := s, i, j;
    endif
endfor
```

4.4 Fourth solution: one maximum selection on enumerator

The specification of the problem can be rewritten:

$$\begin{aligned} A &= (x, y : \mathbb{R}^n, max : \mathbb{R}, ind, jnd : \mathbb{N}) \\ Q &= ((x = x') \wedge (y = y') \wedge (n \geq 2)) \\ R &= ((x = x') \wedge (y = y') \wedge ((max, (ind, jnd)) = \mathop{MAX}_{i=2}^{n}(\mathop{MAX}_{j=1}^{i-1} d(i,j)))) \\ &= ((x = x') \wedge (y = y') \wedge ((max, (ind, jnd)) = \mathop{MAX}_{i=2,j=1}^{n,j-1} d(i,j))). \end{aligned}$$

The last expression of this specification resembles a two-dimensional enumeration [5]. This enumeration should traverse the elements of a virtual lower triangular matrix, i.e. the sequence of index pairs $(2,1), (3,1), (3,2), (4,1), \ldots, (n,1), (n,2), \ldots, (n,n–1)$ should be enumerated. Let us take this enumerator into the state space.

$$\begin{aligned} A &= (t : enor(\mathbb{N} \times \mathbb{N}), max : \mathbb{R}, ind, jnd : \mathbb{N}) \\ Q &= ((t = t') \wedge (|t| \neq 0)) \\ R &= ((max, (ind, jnd)) = \mathop{MAX}_{(i,j) \in t'} d(i,j))). \end{aligned}$$

The enumerator handles two indexes: i and j. The operator $First()$ set them to the pair $(2,1)$, the operator $Next()$ increases the variable j if $j < i–1$, otherwise $(j = i–1)$ increases the variable i and set j to 1. The operator $End()$ gives $true$ if $i > n$. (This process produces the same sequence of index pairs as in the previous solution.) This enumeration can be implemented with a double loop and can be combined with the maximum selection [5]. The local auxiliary variable s is also introduced.

```
max, ind, jnd := d(2,1), 2, 1;
for i = 3...n do
    for j = 2...i–1 do
        s := d(i,j);
        if s > max then
            max, ind, jnd := s, i, j;
        endif
    endfor
endfor
```

5 Discussion

The first two solutions in the case study are very similar. It is easy to see that the second algorithm can be received from the first one through applying

equivalent transformations [3]. But the second solution, which is based on a more universal programming theorem, avoids these transformations. We emphasize that using a more universal programming theorem results in a more efficient algorithm (it does not require an auxiliary array, so its memory space is smaller).

The second and third solutions are based on the programming theorem over interval but the third one uses a function abstraction. Therefore, in the third solution, it is enough to apply the theorem of maximum selection only once. Here the structure of the solving algorithm is simpler than the algorithm of the second solution; moreover, it is the simplest structure among all solutions.

The relationship between the third and fourth algorithm can be seen clearly. Both of them are founded on the same idea, that is, they traverse the elements of the lower triangular matrix row by row. Actually the third solution is complicated. It uses a function abstraction but this function is not trivial. The fourth solution uses a data abstraction when it defines and implements an appropiate enumerator. The structure of the algorithm of the fourth solution is a loop in the loop which is more difficult than the single loop of the third algorithm, but this double loop is the routine algorithm among matrices. Nevertheless, the enumeration could have been implemented in other ways (as we have pointed out) and in this case the algorithm would be a simple loop. The cost of the production of the fourth solution is surely cheaper than that of the third one.

On the whole we can deduct that the more universal programming theorem is used the cheaper the solution is. The cost of production may be cheaper, the structure of the result algorithm may be simpler or its efficient may be better. Then again, to learn and to use an advanced tool is always more difficult than a simple one. Bescause of this in the teaching of programming, gradation must be followed: firstly, programming theorems in an array are to be taught, then the ones over interval, and at the end the theorems on enumerator.

Acknowledgements

This paper is supported by the European Union and co-financed by the European Social Fund (grant agreement no. TÁMOP 4.2.1/B-09/1/KMR-2010-0003).

References

[1] T. Gregorics, S. Sike, Generic algorithm patterns, *Proc. Formal Methods in Computer Science Education FORMED 2008, Satellite workshop of ETAPS 2008*, Budapest, Hungary, March 29, 2008, 141–150. ⇒248

[2] Á. Fóthi, *Bevezetés a programozáshoz*, ELTE Eötvös Kiadó. 2005. (Introductory programming, in Hungarian) ⇒247

[3] Á. Fóthi, Z. Horváth, J. Nyéky-Gaizler, A relational model of transformation in programming, *Proc. 3th International Conference on Applied Informatics*, Eger-Noszvaj, Hungary, August 24–28. 1997. ⇒258

[4] Sz. Csepregi, A. Dezső, T. Gregorics, S. Sike, Automatic implementation of service required by components, *PROVECS'2007 Workshop*, Zurich, Switzerland, ETH Technical Report, 567. 2007. ⇒247, 248

[5] T. Gregorics, Programming theorems on enumerator, *Proc. Teaching Mathematics and Computer Science*, Debrecen, Hungary, **8**/1 (2010), 89–108. ⇒249, 257

Received: August 29, 2012 • Revised: November 30, 2012