PROGRAMMING THEOREMS HAVE THE SAME ORIGIN

Péter SZLÁVI, Gábor TÖRLEY, László ZSAKÓ, HU

Abstract: One of the classical ways of learning programming is to divide programming tasks into large groups, so-called programming theorems, and then tracing the specific tasks back to the programming theorems. Each teaching method introduces a different amount of programming theorems into the learning process, occasionally even combining them. In this article we will show that the basic programming theorems have the same origin; consequently, it would be enough to present one theorem and trace everything back to it. At the end of the article, then, we will explain the practical essence of still introducing more theorems.

Keywords: methodological programming, programming theorems, algorithm, program specification.

1 Introduction

In order to be able to solve a programming task, we need to identify its essence: programming tasks can be categorized into groups according to their type, which is useful because for each group we can create an algorithm rule that solves all of the tasks in that specific group. These task types are called programming theorems because their solutions are justifiably the correct solutions. Their number can vary from teaching method to teaching method [1,2].

If we are familiar with the programming theorems, all we have to do to solve most tasks is to recognize the suitable programming theorem, use the specific data of the general task type, and in the general algorithm substitute them with the task-specific data. Applying this method is supposed to lead us to the correct solution [3,4].

In these tasks we usually have to assign a certain result to one (or more) data collection(s), which, for simplicity's sake, we will handle as some sort of sequences. The essence of a sequence is the processing order of the given elements. Most of the times, it is sufficient to deal with sequences whose elements can be processed – one by one – one after another. In the input sequence, this requires an operation that is capable of giving the elements of the sequence one by one, while in the output sequence elements can be followed by a new element. In simple cases sequences can be illustrated as arrays.

The basic programming theorems are those that assign one value to one sequence:

- (sequential) computing
- counting
- maximum selection
- decision
- selection
- search

2 The first programming theorem: Sequential computing

The first programming theorem comes from a simple task, that of calculating the sum of numbers. In the following, we will provide the specification and the algorithm of the task, which are the two pillars of the programming theorem. We are applying the marking system from [3,4]:

Input: $N\in\mathbb{N}$, $X\in H^N$, $\Sigma:H^*\to H$, $+:H\times H\to H$ ($H=\mathbb{N}$ or $H=\mathbb{Z}$ or $H=\mathbb{R}$)
$\Sigma(X_1,...,X_N)=\Sigma(X_1,...,X_{N-1})+X_N$, $F()=0$

Output: $S\in H$

Precondition: —

Postcondition: $S=\Sigma(X_1,...,X_N)$

```
Sum(N,X,S):
  S:=0
  For i:=1 to N do
    S:=S+X(i)
  End for
End.
```

Generalizing this task, we will get the (sequential) **computing** programming theorem. We generalize the operation **+**:

1) the generalized, binary (**f**) operation should have a (left-side) zero element ($\mathbf{F_0}$);

2) when we apply the operations one after another, the result should not depend on the execution order; that is, the operations should be associative.

The operation, based on binary operation **f** and zero element $\mathbf{F_0}$, and interpreted at H^*, is indicated with **F**.

Input: $N\in\mathbb{N}$, $X\in H^N$, $F:H^*\to H$, $f:H\times H\to H$, $F_0\in H$
$F(X_1,...,X_N)=f(F(X_1,...,X_{N-1}),X_N)$, $F()=F_0$

Output: $S\in H$

Precondition: —

Postcondition: $S=F(X_1,...,X_N)$

Note: many times F is Σ, average, deviation, scalar multiplication, Π, ∪, ∩, ∀, ∃, writing one after another, Max, Min.

```
Computing(N,X,S):
  S:=F0
  For i:=1 to N do
    S:=f(S,X(i))
  End For
End.
```

Note: we are indicating changes from the previous algorithms with **bold**, both here and from now on.

3 Counting

(Sequential) Computing can be formulated so that its result is the sum of all elements with T feature; thus, leading us to the **counting** programming theorem.

Function **F** will be a **conditional sum**. This means that we define function **f** as a **conditional function**, whose value is the first parameter+1 if the second parameter is of T feature, otherwise it is the value of the first parameter.

Input: $N\in\mathbb{N}$, $X\in H^N$, $F:H^*\to\mathbb{N}$, $f:\mathbb{N}\times H\to\mathbb{N}$, $F_0\in\mathbb{N}$

$$F(X_1,\ldots,X_N)=\begin{cases}F(X_1,\ldots,X_{N-1})+1 & \text{if } T(X_N)\\ F(X_1,\ldots,X_{N-1}) & \text{otherwise}\end{cases},\quad F()=F_0$$

$$f(a,b):=\begin{cases}a+1 & \text{if } T(b)\\ a & \text{otherwise}\end{cases},$$

$F_0=0$

Output: $Db\in\mathbb{N}$

Precondition: —

Postcondition: $Db = F(X_1,\ldots,X_N) \to Db = \sum_{i=1,\ T(X_i)}^{N} 1$

```
Counting(N,X,S):
  Db:=0
  For i:=1 to N do
    Db:=f(Db,X(i))   →  If T(X(i)) then Db:=Db+1*
  End For
End.
```

Note: the change marked with * is the normalized algorithmic transformation of the conditional expression.

4 Maximum selection

If we replace function **f** with function **max**, we will get to one of the versions of the **maximum selection** theorem.

Input: $N\in\mathbb{N}$, $X\in H^N$, $F:H^*\to H$, $\max:H\times H\to H$, $F_0\in H$, (H ordered set)

$F(X_1,...,X_N)=\max(F(X_1,...,X_{N-1}),X_N)$, $F()=F_0$

$$\max(a,b)=\begin{cases}a & \text{if } a\geq b\\ b & \text{otherwise}\end{cases},$$

$F_0=-\infty$

Output: $MaxVal\in H$

Precondition: $N>0$

Postcondition: $MaxVal=F(X_1,...,X_N)$

```
Maximum(N,X,MaxVal):
  MaxVal:=-∞
  For i:=1 to N do
    MaxVal:=max(MaxVal,X(i))
              → If X(i)>MaxVal then MaxVal:=X(i)
  End For
End.
```

With a very simple modification, we can even define the version that provides the maximum index as well. The output, the postcondition and the algorithm change as follows:

Output: $MaxVal\in H$, $MaxInd\in\mathbb{N}$

Postcondition: $MaxVal=F(X_1,...,X_N)$ and $1\leq MaxInd\leq N$ and $X_{MaxInd}=MaxVal$

```
Maximum(N,X,MaxVal,MaxInd):
  MaxVal:=-∞
  For i:=1 to N do
    If X(i)>MaxVal then MaxVal:=X(i); MaxInd:=i
  End For
End.
```

If we make use of the precondition, we can leave out the check of the first element from the iteration; in fact, in the classical version we do not need the maximum value either. Hence, the final version is like this:

```
Maximum(N,X,MaxInd):
  MaxInd:=1
  For i:=2 to N do
    If X(i)>X(Maxind) then MaxInd:=i
  End For
End.
```

5 Decision

In sequential computing, function **F** can be operator ∃ too, which leads us to the **decision** theorem. The operator 'or' is associative, its zero element is 'false'; thus, we can include it in the basic version.

Input: $N \in \mathbb{N}$, $X \in H^N$, $\exists: H^* \to L$, $T: H \to L$, $or: L \times L \to L$, $F_0 \in L$
$F(X_1,...,X_N) = \exists i(1 \le i \le N): T(X_i)$
$\exists i(1 \le i \le N): T(X_i) \equiv \exists i(1 \le i \le N-1): T(X_i)$ or $T(X_N)$,
$f(a,b) = a$ or b, $F() = F_0 = $ false

Output: Exists $\in L$

Precondition: —

Postcondition: Exists $= \exists i(1 \le i \le N)\ T(X_i)$

```
Computing(N,X,S):
  Exists:=false
  For i:=1 to N do
    Exists:=Exists or T(X(i))
  End For
End.
```

Note: in the iteration the variable **Exists** can change in the following way:

- `false`, ..., `false`, `true`, ..., `true`
- `false`, ..., `false`

that is, either it remains to be false all through, or it becomes true and stays like that. Consequently, once it becomes true, the iteration can stop. Before it does, it is constantly false, so we do not need to change value. The decision programming theorem derives from this: by the end of the iteration we will be able to distinguish which of the two cases we are dealing with.

Input: $N \in \mathbb{N}$, $X \in H^N$, $T: H \to L$

Output: Exists $\in L$

Precondition: —

Postcondition: Exists $= \exists i(1 \le i \le N)\ T(X_i)$

```
Decision(N,X,Exists):
  i:=1
  While i≤N and not T(X(i))
    i:=i+1
  End While
  Exists:=(i≤N)
End.
```

6 Selection, Search

At the end of the decision iteration, the value of variable **i** is the ordinal number of the item of feature T, provided that we know such an item exists (that is, the iteration will stop once an item is found) → **Selection** theorem

Input: $N \in \mathbb{N}$, $X \in H^N$, $T:H \to L$

Output: $S \in \mathbb{N}$

Precondition: $\exists i(1 \leq i \leq N)\ T(Xi)$

Postcondition: $1 \leq S \leq N$ and $T(X_S)$

```
Seeking(N,X,S):
  i:=1
  While ~~i≤N and~~ not T(X(i))
    i:=i+1
  End While
  S:=i
End.
```

The ordinal number of an element of T quality is N+1 if it has not gone beyond the end of the sequence. If it has → **Search** theorem

Input: $N \in \mathbb{N}$, $X \in H^N$, $T:H \to L$

Output: $Exists \in L$, $S \in \mathbb{N}$

Precondition: —

Postcondition: $Exists = \exists i(1 \leq i \leq N)\ T(X_i)$ and $Exists \to 1 \leq S \leq N$ and $T(X_S)$

```
Search(N,X,Exists,S):
  i:=1
  While i≤N and not T(X(i))
    i:=i+1
  End While
  Exists:=(i≤N)
  If Exists then S:=i
End.
```

7 Conclusion

From the above analysis, we can see that the first 6 theorems (sequential computing, counting, maximum selection, decision, selection and search) all originate from one programming theorem, namely sequential computing, which is simple summation.

Nevertheless, we are not saying that it is useless to manage these 6 theorems independently. Beginner programmers will recognize the programming theorems while dealing with the different task types, which, in the case of basic theorems, correspond to the above defined six [1,3]. With more advanced programmers, it would be worth to base the theorems not

on the task types but on the solution types. In that case, decision, selection and search are three sub-types of the same solution principle; therefore, they can be considered as one programming theorem [2].

This article was supposed to demonstrate that theoretically it is sufficient to check the validity of the first programming theorem because all the rest can be deducted from it. In sum, if sequential computing is correct, so are the others.

References

1. SZLÁVI, P. – ZSAKÓ, L. *Módszeres programozás.* Műszaki Könyvkiadó, Budapest. 1986. ISBN 963106820-x
2. Gregorics, T. *Programozás – tervezés*. ELTE-Eötvös Kiadó, 2013. ISBN 9789633120644
3. SZLÁVI, P. – ZSAKÓ, L. et al. *Programozási alapismeretek*. – on-line curriculum progalap.elte.hu/downloads/seged/eTananyag/, ELTE Informatikai Kar, 2012
4. SZLÁVI, P. – ZSAKÓ, L. et al. *Módszeres programozás: programozási tételek*. Mikrológia 19. ELTE Informatikai Kar, 2008

Reviewed by: doc. dr. Pap Gáborné, PhD

Contact address:

doc. Dr. Péter Szlávi, PhD.

Department of Media and Educational Informatics, Faculty of Informatics, Eötvös Loránd University,

H-1117 Budapest, Pázmány P. sétány 1/C, Hungary

e-mail: szlavip@elte.hu

Dr. Gábor Törley, PhD.

Department of Media and Educational Informatics, Faculty of Informatics, Eötvös Loránd University

H-1117 Budapest, Pázmány P. sétány 1/C, Hungary,

e-mail: pezsgo@inf.elte.hu

doc. Dr. hab. László Zsakó, PhD.

Department of Media and Educational Informatics, Faculty of Informatics, Eötvös Loránd University

H-1117 Budapest, Pázmány P. sétány 1/C, Hungary,

e-mail: zsako@caesar.elte.hu