A Strategy to Introduce Functional Data Modeling at School Informatics

Markus Schneider

Institut für Informatik,
Technische Universität München,
Boltzmannstraße 3, 85748 Garching, Germany
markus.schneider@in.tum.de

Abstract. Having analyzed standard applications by use of object oriented modeling in the 6th and 7th grade functional data modeling is the first topic to be discussed in the 8th grade of the mandatory subject informatics at the Bavarian Gymnasium. First, the data flow modeling technique is introduced and the resulting data flow models are implemented on spreadsheets maintaining the geometrical structure of the diagrams. Yet the first data flow models show the necessity to introduce the concept of functions in full strength. In contrast to the mathematical way the concept of functions is introduced informally using graphical concepts. In a second phase the data flow models are transformed to one complex term. Exemplarily an appropriate algorithm to perform this transformation is presented. Boolean functions and conditional expressions deepen the modeling technique and introduce further central concepts of informatics. Conditional expressions are discussed as functions of arity 3 modeling again commonly known structures. Finally iterative calculations are simulated using a step by step strategy.

1 Introduction

Some reader may wonder why functional data modeling opens the mandatory subject informatics in the 8th grade, since until now the so called classical way was favored, i.e. the teaching of some "hard" programming skills, namely imperative-like control structures. Moreover, one will be reminded through the attribute "functional" to the paradigm of functional programming. Will be recursion the content of the 8th grade? Not to raise such fears one has to emphasize that functional modeling is a pure sequential modeling technique. Only the causal structure and the functional data flow of a context can be represented; recursion or loop-like structure cannot and should not be modeled.

On the other hand, a new empirical study on the learning process of students at university level [5] has shown that students have lowest problems with the functional modeling technique but greater problems with imperative one. So it is obvious to start yet at school with functional data modeling (evidentially, without the concept of recursion). Recently P.Hubwieser [4] has presented the basic principles of functional data flow modeling for the 8th grade. The present paper deals with the details of this concept and offers a strategy to teach functional data modeling. The strategy results from an experimental course carried out from May 2004 to July 2004 in the 9th grade of a Gymnasium near Munich.

The first chapter presents the technical elements of data flow modeling and the transformation of a model to spreadsheet applications. The second chapter, a more mathematical one, introduces the general concept of functions informally using graphical means. Afterwards the third chapter deals with the algorithm to compress the data flow diagram to one term and give heuristics to implement the term in concrete spreadsheet applications. The fourth and fifth chapters on Boolean functions and the conditional expression deepen the concepts discussed so far and introduce at the same time central elements of informatics. The sixth and last chapter discusses problems demanding iterative strategies.

2 Data Flow Modeling and Spreadsheets

Data flow modeling is one of the central abilities to be taught in the 8th grade in Informatics according the new curriculum of the Bavarian Gymnasium. Similar to the suggestion of Hubwieser [4] we use simplified data flow diagrams containing the following elements: Data storages represented by rectangles, data flows represented by arrows and data transforming processes represented by ellipses.

2.1 An Introductive Example

Introducing new modeling techniques at school, one has to show students the benefits resulting from such a technique. Therefore one has to choose examples of which the term structure is not obvious but of which the data flow is of intuitive evidence. This seems to be a contradiction. But this is not the case: Often students have the raw idea to solve algebraic problems, but they cannot transform their idea to term structure. We illustrate this by an example:

Dr. Mabuse has bought an apartment for 140 000 Euro. Having a capital of 70 000 Euro he borrows the half of this amount from a credit institution with an interest rate of 8% per year. Renting the apartment, he wants to finance the credit, i.e. to pay the interests. Calculate the monthly rent (obviously exclusive of heating) for the apartment

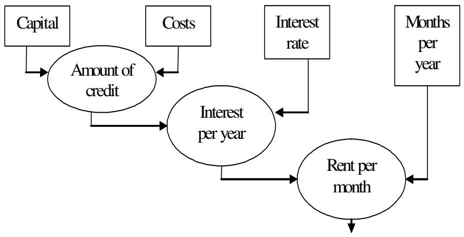


Fig. 1. The Data flow diagram of the introductive example

This problem is simple but complex enough not to apply some predefined formula. To manage such problems, data flow modeling acts as a mediator between the contexts formulated in natural language and algebraic expressions. First we establish the data flow diagram as shown in Figure 1. Up to now, no algebraic elements are used. The diagram shows only the data flow and the causal relations between the objects to be calculated.

Next, the transformation processes will be modeled in detail using standard mathematical functions (Figure 2); finally the resulting detailed model can simply be projected to spreadsheets as shown in Figure 3.

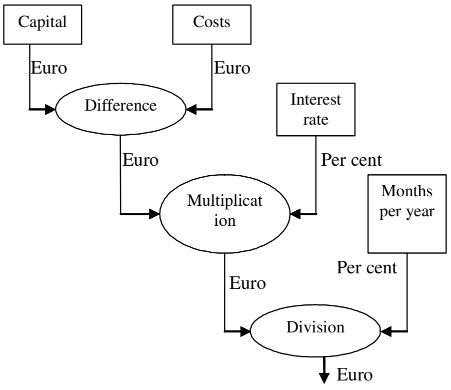
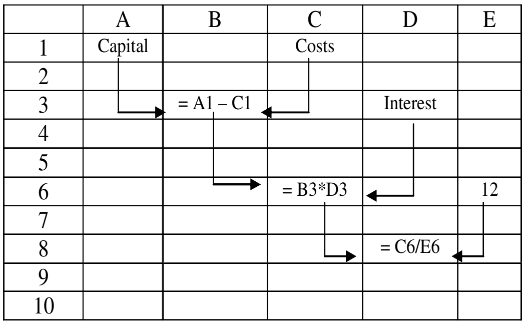


Fig. 2. The detailed model of the introductive example



	A	B	C	D	E
1	Capital		Costs		
2					
3		= A1 – C1		Interest	
4					
5					
6			= B3*D3		12
7					
8				= C6/E6	
9					
10					

Fig. 3. The implementation of the model in figure 2

The detailed model in Figure 2 contains a non standard element: The arrows are labeled with the formats of the data elements. This technique has tried and tested in practice and it has been turned out that these annotations are of great importance for the transformation process from data flow model to spreadsheet implementation. The students are forced to model the various formats and the adjustment of the format can be embedded in the automatic transformation process from data flow model to spreadsheet implementation. For standard spreadsheet applications this adjustment is really necessary, since most operations are polymorphic and incorrect formats results in subtle behavior. Moreover, this technique prepares the concept of the signature of a function discussed below.

A hint ought to be made with respect to the naming of the transformation processes: The above figure uses the identifiers `Difference`, `Multiplication` and `Division`. Some reader may ask why standard binary operations have not been used. The answer will be given in the next chapter where we present an algorithm to compress a data flow diagram to one single term. The use of binary operations would make this algorithm more complex.

2.2 The Learning Targets

Analyzing the above example one recognizes the learning targets of the first phase:

- Students get to know the data flow diagram as a technique to model the functional behavior of problems from the everyday life and they are able to apply this knowledge.
- Students are able to transform data flow models to spreadsheet implementation maintaining the geometrical structure of the diagram, i.e. they will not derive compressed terms. This transformation will be done in the following way:
  - Data elements and transformation processes correspond to spreadsheet cells.
  - Transformation processes correspond to functions or binary operations, often introduced with an "="-sign.
  - Arrows correspond to references inside functions.
  - The format of the cells has to be adjusted according to the annotations of the arrows.
- In the first phase we do not discuss the compressed term structure of the whole problem. Instead we transform the data flow diagram intuitively to the spreadsheet as shown above. So we avoid the discussion of the problem of binary operations and their corresponding prefix-notation. This will be the topic of the next section.

2.3 Further Examples

Evidently, the data flow modeling technique described has to be deepened by suitable examples. The introductory example mentioned above illustrates basic conditions for suitable problems:

- The problem should be embedded in everyday contexts, so that no standard mathematical formula can be applied.
- The term representing the solution of the problem ought to contain the standard functions of spreadsheet applications.

Considering the current curriculum of Bavarian Gymnasia (secondary schools), tasks like the following are conceivable:

- The so called "text-problems" ("Textaufgaben" in German), where the formal structure of the solution is not obvious, but the data flows are of intuitive evidence (standard mathematical textbooks offer numerous examples),

- Percentage and interest calculations as shown above,
- Calculation of the mean value by explicit means,
- The simplification of expressions containing fractions; i.e. these expressions have to be simplified using addition, subtraction, multiplication and division. Here the greatest common divisor (or the lowest common multiple) plays an essential role,
- The areas (volumes) of figures (bodies) composed by standard geometric figures (bodies).

3 The Concept of Functions

According to the current mathematic curriculum of the Bavarian Gymnasium functions are discussed in the middle of the 8th grade. As shown in the previous section, data flow modeling uses this concept from the very beginning in this grade. Moreover, this modeling technique works with functions having arbitrary arity. Therefore, the concept of function has to be introduced yet in the informatics course. It is enough, though, to introduce this concept in an informal manner, since the formal discussion of this concept lies beyond the scope of school mathematics and (evidently) of school informatics.

3.1 Functions of Arity 1

Text books on school mathematics define the function (evidently the function of arity one) as "a map from some set *D* to another set *I* which associates each element of *D* a unique element of *I*" (see for example Barth et al., [1]). The interpretation of this definition by a suitable data flow diagram results in the one given in Figure 4.

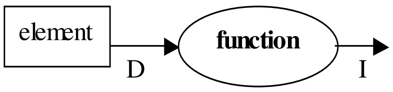


Fig. 4. The data flow diagram of a function having arity one

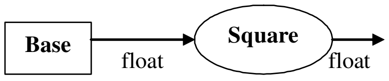


Fig. 5. A concrete example of function having arity one

The basic idea for the introduction of functions in informatics is to describe a function as graphical concept. Therefore, a function is defined as a process which transforms each element of the data type D to a unique element of data type I. Evidently, at school one should proceed in an inductive manner and discuss known functions like `square` or `clean`. These functions have data flow diagrams like shown in Fig. 5.

Having introduced functions of arity 1 as graphical objects the functional behavior of such objects may be characterized textually in a second step using signatures, e.g. `square (base: float): float`, which is very close to the mathematical way of function definition. Moreover, the signature smoothes the way to the syntax of the function call which is nothing than a textual representation of the data flow diagram (without data types).

3.2 Functions of Arity 2

Having introduced the principals of functions of arity 1, it is natural to proceed with functions of arity 2 using functions like

```
power (base : float, exponent : nat ) : float,
gcd (arg1 : nat, arg2 : nat ) : nat,
concatenate (text1 : string, text2 : string) : string.
```

It is helpful to start with functions having prefix-notation. The data flow diagram of such a function suggests the definition of a function of arity 2 as a process which transforms two arbitrary elements of the data types D_1, and D_2 to a unique element of the data type I (figure 6).

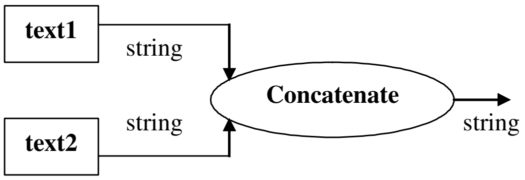


Fig. 6. The data flow diagram of a function having arity two

Again, the structure of the function call is just the textual representation of the data flow diagram resulting from the signature by dropping the data types. To avoid confusions about the order of the arguments inside the function call we use the convention that the order of arguments corresponds to the graphical order from left to right.

Binary operations and prefix-notation

Introducing the concept of functions in such a manner, we have no problems with binary operations and their interpretation as functions of arity 2, since the function is introduced primarily as a graphical concept, whereas the binary operation is a concept of the implementation. Performing this purely syntactical conversion students recognize intuitively the equivalence of functions having arity 2 and binary operations. In contrast to the data flow diagram and its textual representation the implementation is application-specific and some applications offer several implementation possibilities: For example in Microsoft-Excel, the exponentiation can be implemented both as the term `x^y` and `power(x;y)`; similarly, the addition in the standard form `A1+B1` can be also expressed in the form `sum(A1;B1)`.

3.3 Functions of Arity Zero

Functions of arity zero ought to be discussed carefully since some standard spreadsheet functions seem to have arity zero but in fact they use hidden variables. For example the standard spreadsheet operations `today()` or `random()`: The result of these functions depends on the date in the case of the operation `today()` or some other influences (state of the computer, etc.), i.e. we have global, hidden variables, which determine the result.

In contrast the function `Pi()` is a function of arity zero; constant data defined in a data flow diagram can also be interpreted as function of arity zero. Though the number π will formally be introduced yet in the 9th/10th grade, most students know this number informally and the comparison of the functions `today()` or `random()` with the function `Pi()` may be instructive for the students. However, the interprettation of data elements as function of arity zero is perhaps to abstract.

4 From Data Flow Diagram to Compressed Terms

Yet in the first phase of data flow modeling some students will intuitively compress some parts of the data flow diagram to one expression, since this shortens the implementation process. This chapter deals with the formal way to compress a data flow diagram to one single term. Discussing the data flow diagram for addition of fractions we illustrate the principal way. For brevity we restrict ourselves to the data flow diagram describing the calculation of the numerator of the result (shown in Fig. 7):

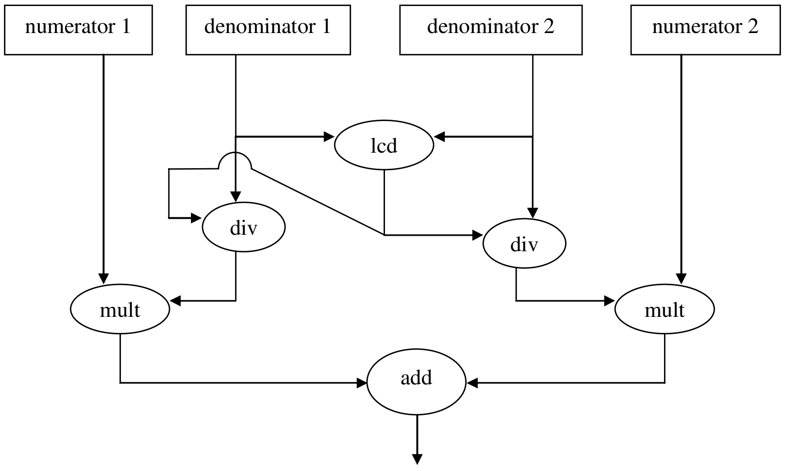


Fig. 7. Addition of fractions: The calculation of the numerator

We derivate the structure of the compressed term using a "bottom up" strategy. The outermost function is the function `"add"` in the last (5) row of the data flow diagram. So we have initially the textual representation

```
        add( ? , ? ).
```

Were the question mark denotes the result of the inner function calls. The arguments of the function add are the elements of the fourth row and one gets:

```
add( mult(?,?), mult(?,?)).
```

Inserting the arguments of the function mult results in

```
add( mult(nominator1,div(?,?)),
     mult(div(?,?),nominator2)).
```

Translating the remaining graphical elements, we get finally:

```
add( mult(nominator1,
          div(lcd(denumerator1,denumerator2),
          denumerator1)),
     mult(div(lcd(denumerator1,denumerator2),
          denumerator2))
          nominator2)).
```

Initially, the term presented is nothing than a textual representation of the data flow diagram. To implement this term into a concrete spreadsheet application it has to be adjusted to the respective application. Dependent on the application, some parts of the term have to be translated to infix-notation or to special function-identifiers. Best, one starts with the innermost terms, i.e. with the function div in the above example and proceeds outwards. So, one gets first:

```
add( mult(nominator1,
          lcd(denumerator1,denumerator2)
          /denumerator1))),
     mult((lcd(denumerator1,denumerator2)
          /denumerator2))),
          nominator2)).
```

After two further translations we finally have:

```
  (nominator1*(lcd(denumerator1,denumerator2)
  /denumerator1))
+((lcd(denumerator1,denumerator2)/denumerator2)*
nominator2).
```

The application-specific term presented here is suitable for standard application as Microsoft-Excel for example.

5 Boolean Data Types and Boolean Functions

Some reader will wonder why to discuss Boolean expressions and functions? What is the benefit of such a unit, since this topic has disappeared yet from some curricula in mathematics? On one hand, the Boolean data type is a fundamental data type in informatics appearing yet in simple expressions containing relational operators like "=", ">" or "<". On the other hand, the conditional expression, which will be discussed later, contains Boolean functions as a central element.

Sometimes, the discussion of this topic at school is carried out abstract emphasizing primarily the mathematical relevance. But as the some textbooks on formal logic (see. for example R. Winter, [6]) show, this must not be the case: Taking statements and contexts from everyday life, students realize that Boolean functions are a natural element of our communication. As outlined above, one has to start with examples were the mathematical solution is not obvious but were the model contains standard Boolean functions of spreadsheet applications.

What class of problems might be suitable? Standard spreadsheet operations offer some Boolean functions: the logic standard operations and the polymorphic Boolean functions: `greater, less, equal`. The last three functions work both on numbers and strings. Since the lexicographic order ought to be well known to students on an informal level (telephone book, class list, etc.) problems based on the lexicographic order seem suitable to introduce the concept of Boolean functions.

5.1 Boolean Data Types

First of all the concept of Boolean data type has to be introduced. This can be done analyzing some simple statements, i.e. sentences which are either true or false:

- In an encyclopedia the name “Radlbacher” is found first, the name “Wastlhofer” behind.
- The name “Häberle” is equal to the name “Häfele”.

To delimit the concept of statement from other structures one ought to compare this statements with other grammatical structures, for example with questions like “What is your name?” or with orders like “Write down your name!”

Modeling the data flow of the above statements one gets the data flow diagram shown in Figure 8:

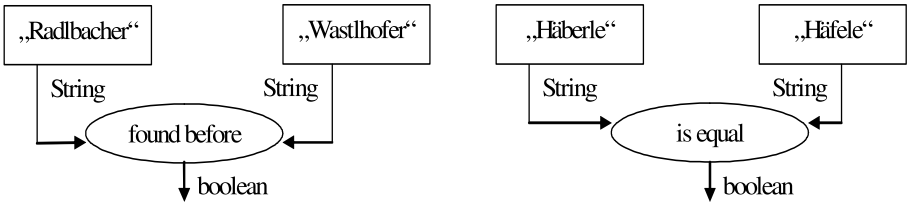


Fig. 8. Examples to motivate the use of boolean expressions

Interpreting these statements as functions they have a result value which is either true or false, i.e. the result value is of the Boolean data type. Therefore, we define Boolean functions as transformation processes of arbitrary arity and Boolean result values.

To implement the above examples we turn to the textual representation:

- `foundBefore("Radlbacher", "Wastlhofer")`
- `isEqual("Häberle", "Häfele")`

Implementing these terms, the students are really surprised recognizing that the somehow cryptic functions `foundBefore` and `isEqual` are represented by the common "<" and "=" operators. Beside, the teacher has the possibility to deepen the structural similarity between the lexicographic order and the order over numbers.

Obviously the strategy presented could be carried out using numbers instead of strings. But then one has the problem to motivate the data flow modeling: Starting with expressions like "3 is less than 4", one has no necessity to model the data flow, since the final term is evident.

5.2 Complex Boolean Functions

Having introduced the basics of Boolean data types, one continues with complex Boolean functions. Again, we discuss problems based on the lexicographic order and combine them with the standard logic operations `and, or, not`.

Suitable problems may look like the following:

- Give the data flow model of a function, which determines if two names are different.
- Give the data flow model of a function, which determines if your name is found in an encyclopedia between the two names "Dimpflmoser" and "Schlotterbeck".

Solving these problems one gets the following data flow diagrams:

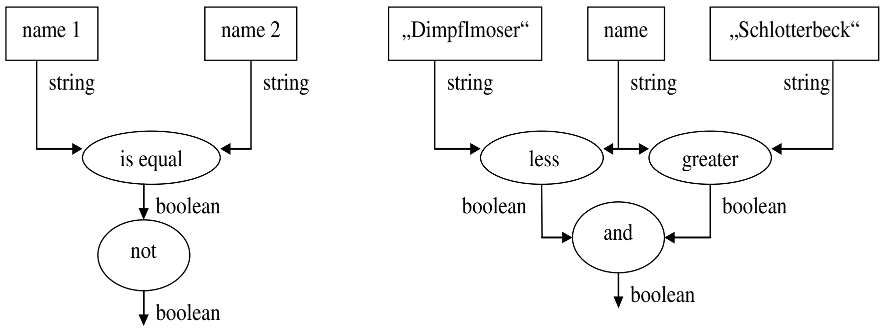


Fig. 9. Two examples of complex boolean functions

Without efforts, the logic operators `not, and` are introduced in a natural way. The transformation of the data flow diagrams to compressed term and finally to application specific form is straightforward and will not be discussed here. Having implemented the models one ought to analyze the semantic of the standard logic functions in detail.

6 Conditional Expressions

The conditional expression is a central control structure in Informatics. In this unit students get to know the conditional expression as a function of arity 3, having a

Boolean function as the first argument and two other expressions with identical data type as second and third expression.

Though possible, it is not recommendable to begin with pure mathematical examples. Instead one might begin with an analysis of common conditional sentences, like the following rule applicable an hot pre-summer day:

> *If the temperature at school exceeds 30° at 10 o'clock, the instruction terminates at 11 o'clock, otherwise instruction happens according to class schedule.*

The Raw analysis of this sentence shows that a conditional sentence contains three parts:

- The condition, prefaced with "if"; this condition is either `true` or `false`.
- Two consequences; the first is carried out, if the condition is true, the second otherwise (sometimes the second alternative is omitted).
- Therefore, the modeling of conditional sentences with conditional functions results in a function having 3 arguments:
- A Boolean function.
- Two functions with arbitrary but identical result-data type; the first function is called, if the condition is true, the second otherwise.

Therefore, we get a function with a data flow diagram as shown below in figure 10.

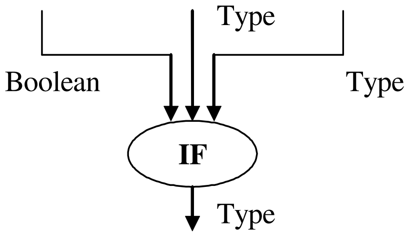


Fig. 10. The data flow diagram of the conditional expression

The signature of such a function has the structure:

```
If (condition: Boolean,
    true-function: TYPE, else-function: TYPE): TYPE
```

were `TYPE` denotes an arbitrary data type.

Since conditional sentences on everyday contexts cannot be implemented via spreadsheets, more formal examples (but not necessary mathematical ones) have to be used. Here we present the problem to compare three strings with respect to their lexicographic order and to calculate the "greatest" among them. (It should be mentioned that this example is too complex as unproductively example at school. Instead one ought to proceed step by step and discuss first the analogue problem for two strings.)

The solution of this problem cannot be modeled using one conditional expression. Instead, one has to nest two conditional expressions. The considerations concerning data flow diagrams of conditional expressions suggest the model shown in Figure 11: First, two strings are compared and the greater one is returned; afterwards this result is compared with the third string in the same manner.

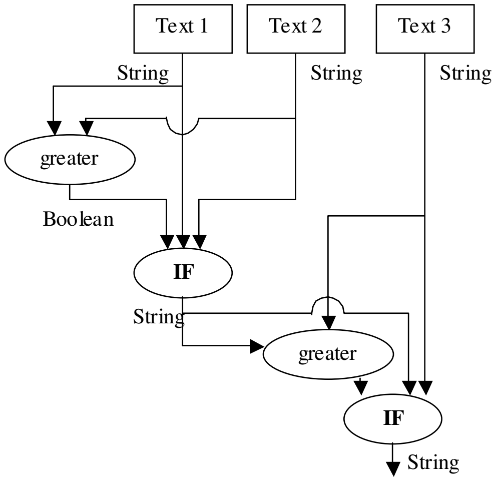


Fig. 11. The data flow diagram to calculate the "greatest" of three strings

As above the compressed term is calculated using the "bottom up" strategy. Translating the outermost conditional expression we get:

```
If(greater(?,text3),?, text3)
```

Now, the question marks have to be substituted by the term

```
If(greater(text1,text2), text1, text2).
```

So we get:

```
If(greater(If(greater(text1,text2),text1, text2),
                text3),
    If(greater(text1, text2),text1, text2), text3)
```

Finally this implementation independent term ought to be prepared for specific implementation:

```
If((If(text1>text2), text1, text2)>text3,
               (If(text1>text2), text1, text2), text3)
```

Translating the data flow diagram to term structure, it is important to mind the order of the arguments. As above, we have the convention that the graphical order of arguments (left to right) corresponds to the textual one.

The example illustrates a specific feature of functional modeling: There is no possibility to store a result. For example, the term `(If(text1>text2), text1, text2)` has been used two times and it has been calculated two times. This is due to the fact that functional modeling lacks the concept of variables.

Finally, we present some possibilities to deepen conditional expressions:

- Calculation of interest: Dependent on the "sign of the account" the sign and the value of the interest rate changes.
- Discount calculations: Dependent on the amount of products bought one gets a specific discount. Here, nested conditional expressions can be used
- Further ideas for exercises can be found in standard school-textbooks on informatics in the context of the conditional statement (for example: Fuchs et al., [3]).

7 Iterative Calculations

Standard courses on imperative programming languages at school contain a chapter discussing loops as a central element. In contrast, functional data models are pure sequential structures and we have no possibility to model loop structures. Instead, such calculations have to be carried out step by step performing the various iterations explicitly. With respect to didactical aspects this is no disadvantage: The step by step calculations discussed give students the fundamental experienc to recognize the necessity of loops as an efficient tool to model repetitive structures.

A treasure of ideas on iterative calculations with spreadsheets can be found in Dopfer G., Reimer R., [2]. Unfortunately, some interesting examples like Heron's algorithm to calculate the square root of a number, or the algorithms to approximate the number π cannot be used due to curricular conditions. Some other examples, however, like the one on the greatest common divisor or interest calculations with regular (or irregular) payments are undoubtedly applicable.

Though the cited textbook of Dopfer G., Reimer R. presents valuable ideas, these ideas have to be designed for functional data modeling. In the sequel, we illustrate the principal strategy discussing the step by step calculation of the greatest common divisor:

Yet in the 5th grade students learn Euclid's algorithm to calculate the greatest common divisor of two natural numbers. Obviously, the algorithm used here ought to show a non-recursive structure like the following:

Euclid's algorithm to calculate the greatest common divisor of two natural numbers a and b, gcd(a, b):

1. Let $a_1 = a$ and $b_1 = b$;
2. If a_1 equal b_1, then a_1 is the gcd(a, b) and the algorithm terminates
3. Calculate a_2 and b_2 as follows:
   If $a_1 > b_1$, then $a_2 = a_1 - b_1$ and $b_2 = b_1$, otherwise $a_2 = a_1$ and $b_2 = b_1 - a_1$;
4. If a_2 equal b_2, then a_2 is the gcd(a, b) and the algorithm terminates
5. Calculate a_i and b_i, for $i > 2$ analogous until $a_i = b_i$; then $\gcd(a, b) = a_i$.

The data flow model (Figure 12) for this algorithm is developed in two steps: First one ought to model the global data flow to calculate the various values a_i and b_i. This initial model is shown on the left side of Fig. 12. The second step deals with the data flow model to calculate a_{i+1} from a_i and b_i. This model is given on the right side of the figure. It shows the details of a transformation process from the left side (colored ellipse).

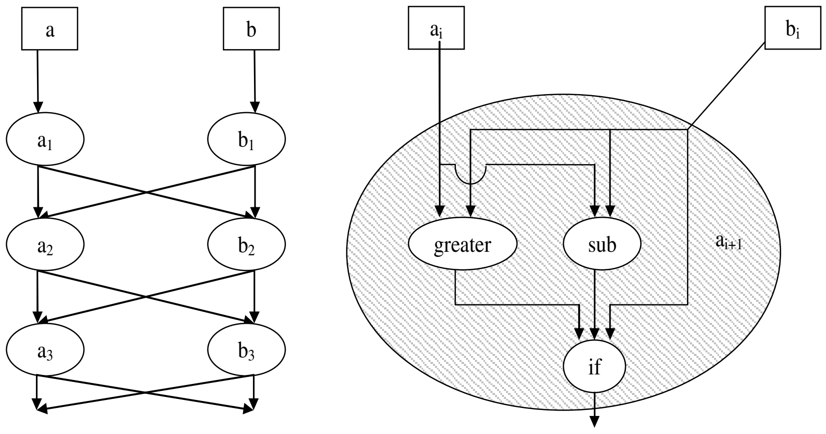


Fig. 12. Data flow diagrams to calculate the greatest common divisor of two numbers

The implementation of this model proceeds analogously: First, the global model is transformed to the spreadsheet maintaining the geometrical structure. Afterwards, the detailed model has to be compressed to one term and inserted into the respective cells (Figure 13).

	A	B	C
1	a_1		b_1
2			
3			
4	= If(A1>B1; A1-B1; A1)		= If (A1>B1; B1; B1 – A1
5			
6			
7	= If(A3>B3; A3-B3; A3)		= If (A3>B3; B3; B3 – A3
8			
9			

Fig. 13. The implementation of the data flow diagram in figure 12

Abstracting from this example, one recognizes the well known structure of loops: The detailed (inner) model corresponds to the body of the loop, the global or outer model to the various iterations. But in contrast to loop or recursive structures, the user himself decides whether the algorithm terminates after a certain step or not.

Further topics for exercises:

- The calculation of roots by nested intervals,
- Growth or decay-rates
- Interest calculations with regular (or irregular) payments; (see P. Hubwieser, 2004 for the basics of such exercises).

8 Final Remarks

To my knowlede the only official publication on functional data modeling at school is the one of P. Hubwieser (2004) [4]. Based on this publication the author of the present paper has taught functional data modeling from May 2004 to July 2004 in the 9th grade of a Bavarian Gymnasium near Munich. The experiences made there are already incorporated into the strategy presented. But nevertheless, we have great lack of suitable material. Since the mandatory subject informatics starts in 2006 with functional data modeling in the 8th grade, it is really urgent to continue the phase of material development. Teachers and scientists ought to conceive and evaluate examples and problems on functional data flow modeling.

A central problem omitted here concerns the methodology of. teaching functional data modeling: The method of project based teaching, a central element of informatics at school, should be integrated into the course strategy by the development of suitable worksheet sequences.

Apart from the topic discussed here the mandatory subject informatics of the Bavarian Gymnasium offers teachers and scientists numerous fields to work on such as data bases, imperative programming, and automata to mention only three topics of similar urgency. These topics ought to be addressed from interested scientists.

References

1. Barth F., Federle R., Haller R., 1996: Algebra 8, Ehrenwirth Verlag
2. Dopfer G., Reimer R., 1995: Tabellenkalkulation im Mathematikunterricht, Klett-Verlag
3. Fuchs M. et al., 1994: Einführung in die Informatik, Klett-Verlag
4. Hubwieser P., 2004; Functional Modeling in Secondary Schools using Spreadsheets; in “Education an Information Technologies” of the Official Journal of the IFIP Technical Committee on Education, Vol. 9, Nr. 2
5. Schneider M., 2004; “An Empirical Study of Introductory Lectures in Informatics Based on Fundamental Concepts” in “Informatics and Student Assessment” Lecture Notes in Informatics, Vol. 1
6. Winter R., 1996: Grundlagen der formalen Logik, Verlag Harri Deutsch, 1996