Preparatory Knowledge: Propaedeutic in Informatics

Susanne Loidl[1], Jörg Mühlbacher[1], Helmut Schauer[2]

[1]Institute for Information Processing and Microprocessor Technology (FIM), Johannes Kepler University Linz, Altenbergerstr. 69, A-4040 Linz
`{loidl, muehlbacher}@fim.uni.linz.ac.at`
[2]Department of Informatics (IFI), University Zurich, Winterthurerstr. 190, CH-8057 Zurich
`schauer@ifi.unizh.ch`

Abstract. In the recent past a number of concepts have achieved prominence in the quest for basic principles of informatics with long-term validity. Particularly at schools providing an all-round education, it makes sense – and is necessary – to concentrate on basic concepts. The fact is that strictly product-related knowledge is inadequate, and in some cases already obsolete before pupils leave school. A more systematic grasp of these concepts and their interrelations is therefore not just desirable, but essential. Some of these "unchanging values" in informatics are briefly introduced here, and it is shown how they can be, first, made more comprehensible by means of applets, and second, put to work in teaching right now, in conjunction with eLearning.

1 Introduction

In the recent past a number of concepts have achieved prominence in the quest for basic principles of informatics with long-term validity – and these should be playing an increasing part in the curricula of schools providing an all-round education; the concepts in question are briefly introduced and discussed here (this paper is a shortened version of [6]; see also [3]). Examples of such concepts are: abstraction, particularly in connexion with modelling and recursion, differing forms of notation (with a clear distinction between syntactic and semantic aspects), or distinctions between static/dynamic and local/global aspects also appear important. Again, special properties of relations, such as transitivity, symmetry or reflexivity, and (say) the difference between identity and equivalence, are of primary significance in informatics. The examples selected and given below (see also [10] and [12]) are intended to show how and (especially) which concepts can be conveyed.

The issue of how far procedural or object-oriented programming should be included in formal education is not discussed here. While programming is an excellent training in algorithmic thinking, it does require a certain amount of practice. The latter counts as a skill, and its status and scope are bound to depend on the individual type of school and the educational goals the school pursues.

2 Examples of basic concepts in informatics

Let us consider a typical task in information processing: determining the mass x of an unknown object by means of a balance. To analyse this task, we start by constructing a **model** [2] (Fig.2).

2.1 Modelling, abstraction, states

Modelling involves **abstraction**: certain aspects of the task are deemed to be relevant, and are taken into account in the model, while other aspects are treated as irrelevant and thus ignored. What is deemed to be relevant or irrelevant is of fundamental importance, and depends on the purpose of modelling. Here we ignore the size, shape and colour of the unknown object, for instance, and consider the balance only at rest, with three possible results of weighing: the mass of the object in the left-hand pan can be less than, equal to or greater than the sum of the masses of all the weights in the right-hand pan. This last point illustrates the distinction between **static** and **dynamic** aspects of modelling and the concept of a **state** which a system can be in. A system of this kind, that can be in various defined states and that switches from one to another as a result of defined **events**, is called an **automaton**. If every subsequent state is uniquely determined by the current state and the event in question, the **automaton** is **deterministic** and its behaviour can be forecast. Gambling machines are typically non-deterministic. If we consider placing a weight in a pan as an event, our model of a balance is then a deterministic automaton. If we permit the removal of weights previously placed, a change of state can be reversed. Changes of state can thus be **reversible** or **irreversible**. In the case of computer applications, any action that can be reversed by means of **undo** is an example of a reversible change of state.

A further important aspect is the **accuracy** of a weighing procedure. For instance, we can decide in favour of a discrete model with integer weights, with which the mass of the unknown object can be ascertained only as a whole number, while leaving it open whether the weights are specified in grams, kilograms, etc. With the distinction between **discretely** and **continuously** variable values we have another concept basic to informatics.

Another key aspect of modelling is deciding what is rigid about the model and what can be altered. For example, a given set of weights could be prescribed, or the choice of weights could be left open. Again, the balance beam could be supported at its midpoint in all cases, or the point of support could be shiftable, to permit a free choice of leverage. Which the **parameters** of a model are, and which quantities are treated as **constant** and which as **variable**, are thus also fundamental issues. Alan Perlis [9] put this very neatly 30 years ago in the remark “One man’s constant is another man’s variable”.

One special aim in the balance example, going beyond modelling as a function of the level of abstraction selected, is a discussion about the purpose of the resulting model. For mathematicians the equation $x*a = g*b$ suffices, where g is the weight used to determine x and a and b are the beam lengths. This presupposes that the aim is to model a mechanical balance, for a weighing device in which (say) the extension of a spring as a function of the weight applied is used to measure mass involves a different equation.

Informatics specialists using integer weights (here their order of magnitude plays a part) need to take into account both the resulting "inaccuracy u" of the balance and the process of the balance coming to rest; in the model they will therefore start from an inequation $|x*a - g*b| < u$, or regard the equation as satisfied once the angle α is less than an ε adapted to the purpose of weighing. So a discussion of the balance example can well lead on in the classroom to a discussion of the difference between formal, mathematical modelling and the sort of modelling typical of the engineering sciences. Interpreting the findings obtained from a model will also need discussion.

The following key properties of models can thus be discussed: Completeness (in terms of the purpose intended), freedom from contradictions (consistency), fidelity to the original and the associated interpretation of the data provided by the model.

The metamodel for discussing modelling is shown in Fig 1.

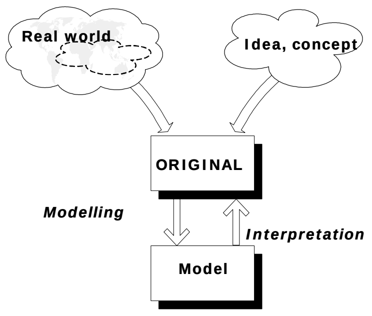

Fig. 1: Metamodel of the modelling process

On top of this, for informatics specialists the weighing procedure leads directly to the concept of an algorithm.

2.2 The concept of an algorithm

No doubt about it, the concept of an algorithm is fundamental to training in informatics. As indicated initially, we shall not comment on programming in a programming language, although programming is naturally the special procedure for informatics specialists to formulate algorithms. At this point we are more concerned with the concept of an algorithm independently of the software context.

Let us consider the sequence of actions that are performed when an object is weighed by means of a balance or a model that mirrors its behaviour. For instance, we can place weights in the pan or remove them completely at random, until the balance is in equilibrium. Apart from the fact that this procedure takes time, it comes to an end only if the mass x of the object can be represented as the sum of a subset of the available weights.

So let us search for a directed procedure that determines the mass x of the object in as small a number of weighing operations as possible. This leads us to the concept of an **algorithm**. If the individual steps are to be performed in a particular order, the algorithm is called **sequential**. Algorithms in which the individual steps can be performed in any order, or even simultaneously, are called **parallel**. For example, several weights can be placed or removed simultaneously, and thus parallel; on the other hand the individual weighing operations are performed sequentially. The distinction between sequential and parallel procedures is also of great importance in informatics – we need only think of data transfer via serial or parallel interfaces, say.

As formulated here, the examples belong to the class of **iterative** algorithms. Going further, we come to the issue of **recursion**. Often a recursive approach yields a simple solution to a problem. “Recursive” means “with self-reference”. Recursion occurs whenever something refers to itself.

Traditional examples of recursion, such as the Fibonacci series: Fib(n) = Fib (n-1)+ Fib(n-2) with Fib(1) = Fib(2) = 1 or n factorial: n! = n*(n-1)! with n>1 and 1! =1, are to be found both in informatics teaching and in mathematics teaching.

However, we are more concerned with recursive thinking, the recursive description of observations and the use of recursion to solve problems. Here comprehensible, concrete tasks and examples adapted to the year/level in question must be found.

An initial, straightforward example of recursion from everyday life is a tree. Let us imagine a cross-section through a tree-trunk and examine the growth rings that have formed around the central pith. A tree one year old has one annual ring surrounding the pith. In the general case the cross-section of a tree-trunk consists of the outermost ring surrounding the cross-section as it was one year earlier. And this recursive perspective continues until the “abort criterion” is satisfied, when the pith is reached!

The following example does cause a certain surprise in class (in our experience), when one explains a succinct way of describing a queue of people waiting ahead of a cash desk in a supermarket: a queue Q of persons P is either empty (an empty queue) or consists of a person P followed by a queue Q. If one points out at the same time that one ca abstract from a “person P” to any object, and introduces a non-existent “empty” object ε in analogy to the empty set, one gets the pure concept of a queue!

By comparison, describing a queue iteratively takes much more doing. It depends on the educational goals the school in question pursues, and on the year/level in question, whether one then decides to tackle the next step towards EBNF (Extended Backus Naur Form) by means of the following example, which is also excellent training in thinking: how do we describe how a train is put together?

To put it simply, a train consists of an engine E at the head, followed by at least one coach C. The recursive description focuses on "how is a train put together". We can write: train = ET and T = C | CT. For practice one can then derive train = ECC, train = ECCC, train = ECCT etc. and recall the situation with the queue for comparison: Q= ε|P|PQ.

2.3 Time complexity

Let us try to estimate the effort involved in our weighing algorithm. With a purely trial-and-error approach the mean number of weighing operations is proportional to the number of all possible subsets of the weights. If the number of weights is n, the number of all subsets of these weights is equal to 2^n, so the mean effort increases exponentially with n. The reason for this unnecessarily great effort is that with a purely trial-and-error approach weights are selected for the next operation independently of the unsuccessful previous tries. No use is made of the information whether the object was lighter or heavier than the sum of the weights selected for these tries! Actually, the largest weight value tested that was lighter than x forms the lower limit, and the smallest weight value tested that was larger than x forms the upper limit, of an interval that the value x to be found must lie within. The strategy behind an optimized weighing algorithm can only be to halve this interval at each weighing operation, by comparing x with the arithmetical mean of the interval limits. If x is lighter than this mean, the latter becomes the new upper limit; if x is heavier than this mean, the latter becomes the new lower limit. x has been found when it equals the mean or the interval has been reduced to 1. Since each weight is placed only once, the effort (number of operations required) is in **linear** proportion to n. This optimized algorithm is thus much more efficient than trial and error. In connexion with the time needed to perform an algorithm one speaks of **time complexity**, a fundamental concept in informatics. Other key issues in connexion with the concept of an algorithm include the question of whether an algorithm holds, whether a problem is decidable, computable, etc. We return to these questions later.

2.4 Number systems, coding

Since the number of weighing operations required is a function of the number n of weights, the question arises of whether the number of weights can be reduced without restricting the range of weight values that these can represent. With a conventional set of weights with the eight values 1, 1, 2, 5, 10, 10, 20, 50 for instance, all integer weight values within the interval 0 to 99 can be represented. This choice of weights is obviously inspired by the decimal number system. Interestingly, the sequence 1, 2, 5, 10, 20, etc. has the original property that for each pair of numbers in sequence the first value is the integer half of the second value (the sequence 1, 2, 5, 10, 20, etc. corresponds to the values (rounded to integers) in the European Standard sequence E3, which assigns three logarithmically roughly equidistant values to each decade). The sequence of powers of 2 1, 2, 4, 8, 16, 32, 64, etc. also adheres to this principle; with the corresponding binary set of weights with the seven values 1, 2, 4, 8, 16, 32, 64 all integer weight values within the interval 0 to 127 can be represented. Although a binary set of weights of this kind is not a standard product, it is superior to the decimal set of weights.

Fig. 2 shows the result of a weighing operation using a binary set of weights. The weights placed in the pan correspond exactly to the positions of the ones in the binary coding of x.

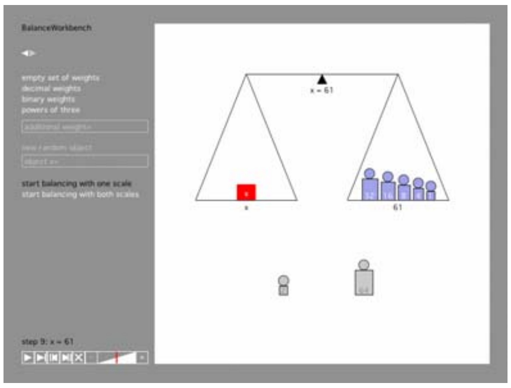

Fig.2. Picture of a model of a mechanical balance, as an example of modelling and binary coding (with the weights 32, 16, 8,4,2)

2.5 Decidability, computability, NP complete problems

In connexion with questions such as whether an algorithm holds, i.e. whether we are dealing with a decidable, computable problem, a tractable problem etc., the favourite objection is that such questions are far too complex, go beyond schools' educational targets and should be reserved for the sphere of tertiary education. In this section we want to show that simple examples that can be formulated intuitively really exist and can be used to introduce these topics in informatics in secondary schools.

At the same time there must be a strict requirement that informatics should be taught only by people with a relevant qualification! We accordingly take it that the discussed topics are already known, and concentrate on the issue of satisfactory didactic treatment.

We start by considering whether everything that occurs to one can be subjected to algorithmic treatment, and thus ultimately to programming.

The halting problem is a good example of a problem that is easy to grasp: can one define an algorithm that decides, for any algorithm whatever (!), whether it completes after a finite number of steps or not? Depending on what the pupils already know, this problem is fairly easy to describe verbally: imagine someone sitting at a PC, waiting some time for results and becoming increasingly worried about whether the program currently running just takes a considerable time or whether a bug has crept in and the best thing would be to abort it. This leads to the wish for a test program that can decide in advance whether the program in question will ever complete and provide results. One can then point out that theoretical informatics delivers the conclusion (which pupils might not have expected) that tasks do exist that are not computable, i.e. not programmable, and that the halting problem is an example of such a task. At the same time the pupils are confronted with a good reason why informatics investigates its own basis in theoretical informatics.

At the next stage it can be assumed that from now on only computable problems will be examined in detail. Here they are very simple, instantly comprehensible tasks such as sorting a finite set of numbers etc. At the same time the requirement should be to perform such tasks in the most efficient way possible, i.e. to search for good algorithms – “good” can be defined as minimizing run time. To illustrate what counts as a good or a less good algorithm, let us take n = 7 integers, order them graphically, first as a linear list and then a binary search tree, and now ask how many comparisons are needed to find out whether an integer x is *not* among the 7 numbers selected; this provides a preliminary justification for the subject “Algorithms and data structures”. If a link to mathematics is to be developed here and the pupils have the necessary basic knowledge, the binary search resulting from this example leads to logarithms to the base 2, $\log_2$ n. The next question is how the number of comparisons increases if one selects 2n rather than n numbers.

At the next stage a particularly impressive example is used to make it clear that time-consuming problems cannot be solved simply by technical progress – acquiring a faster computer. To illustrate this phenomenon, the puzzle problem discussed in detail below can be presented; it is easy to explain:

We consider a very small jigsaw puzzle, measuring 5 by 5 pieces. All the pieces are different, but should yield the picture intended, if they are put together correctly.

First of all one must ask the didactically central question whether the problem is soluble at all (computable), i.e. whether it can be solved with the 25 pieces given. If we recall that children can perform this task before they start going to school, there does not seem to be much of a problem. However, it is clear that a computer will need an algorithm: before tackling the puzzle problem, we must find out whether it is computable! A simple brute-force algorithm supplies a positive answer:

- Number the pieces from 1 to 25.
- Arrange all pieces in a sequence. We thus obtain all n! sequences of the n (= 25) numbers.
- For each resulting sequence, check whether it solves the puzzle.

In the worst case it takes n! tries to find the correct sequence!

At this point, faithful to the principle of interdisciplinary teaching, we can introduce the concept of permutation, and use a few examples to derive the number n! of permutations of n numbers, or even repeat the definition n! = n(n-1)! (with a glance back to recursion).

If we omit rotations – determining the number of possibilities could get us into didactic difficulties –and use a computer performing 1 billion checks per second, we get the following figures: Placing: 25! = 1,55*10^25 seconds, i.e. ~ 4,9*10^11 years. That is still 15 times as long as the time that has elapsed since the original big bang! It is didactically effective to get the students to give an intuitive estimate of the time required first.

Two lessons emerge from this: acquiring a faster computer does not help at all, and we need to start hunting for a better (good) algorithm.

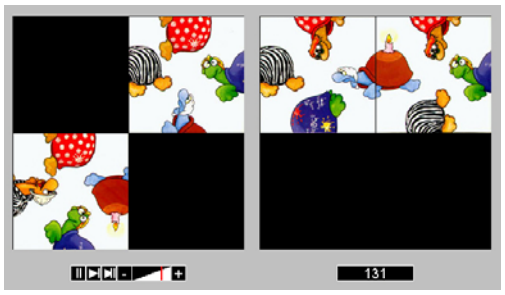

Fig. 3. Puzzle problem simplified with 2x2 pieces

At this point it is up to the teacher to convince the pupils that procedures for solving the puzzle problem within a realistic length of computing time are known, e.g. by using structural data about the edges of the individual pieces to get to a solution.

However, a discussion about this leads straight to the issue of NP complete problems, though we must be aware that this topic can be mentioned only verbally and in simplified form. But even at this level it is perfectly suitable for awakening pupils' curiosity, and thus getting them interested in the science of informatics.

The following selection of examples has worked well in practice: one starts with the Travelling-Salesman-Problem (visiting n towns without visiting any of them more than once), which can be explained graphically without difficulty. It is also easy to show that this problem is computable: the approach is to list all permutations of the n towns and to check for each permutation whether it satisfies the criterion for a round trip. In secondary education one then has no choice but to point out that, interestingly enough, (1) for large n no method of solving the problem in a realistic length of time has yet been found, and (2) theoretical informatics provides the following remarkable statements: (a) there is reason to suspect that no algorithm exists to solve the problem, and (b) according to the state of science it will never be possible to prove that the suspicion voiced in (a) is correct.

The next step is to remind the pupils that, if their school has a large number of classes and teachers, the timetable they get at the beginning of the school year is unlikely to be definitive – instead, it will be a compromise (method of successive approximation), since the task to be performed is defined as follows: obviously no teacher can teach in two classes simultaneously, but he or she should a continuous succession of lessons with no gaps, and the sequence of subjects per schoolday should make sense for each class.

The remarkable thing is that the same suppositions apply in the case of this so-called timetable problem as with the Travelling-Salesman-Problem: if n (the number of teachers) and m (the number of classes) are very large, trial and error will not lead to a satisfactory result. Oddly enough, though, if a good (polynomial time bounded) solution were found, it would follow that a good solution in the same sense existed for the timetable problem, and it would make sense to go on hunting for one. The converse also applies: if a proof of statement (a) were found for the Traveling-Salesman-Problem, we would know that no solution existed for the timetable problem, either. The argument also applies in the other direction: if it can be proved that no tractable solution exists for the timetable problem, then none exists for the Travelling-Salesman-Problem.

The following selection of examples has worked extremely well in the classroom: one presents the timetable problem verbally only, and then goes on to the so-called clique problem as a further instance of an NP complete problem. It is very easy to illustrate this by drawing a graph [11] with 5 nodes and 8 edges (Fig 4), with no need for previous knowledge in mathematics.

As with the Traveling-Salesman-Problem, there is an opportunity here to return to the concept of a model: here the nodes correspond to persons, and an edge is drawn if a special relationship exists between two persons. A subset of nodes and edges is called a clique if an edge exists between every pair of nodes.

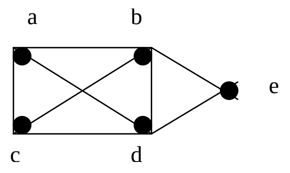

Fig. 4. Graph with 5 nodes and a clique of four defined by a, b, c, d

In our experience classwork is enhanced by a discussion of this issue, together with a reference to the fact that more than a thousand problems equivalent to the two presented here are known [1]. Here Informatics teachers are confronted with the same didactic problems as their colleagues in the natural sciences, who are obliged to draw attention in their teaching to any number of unresolved issues. In our view it is didactically worth-while to point out the limits of a discipline without explaining the underlying formal basic principles.

As a special aid in connexion with this topic, a study guide has been added to the eLearning version of the preparatory course in informatics [10]– see section 3.2.

As regards, first, the exact definition of "tractable" by means of "big" O notation with a polynomial to describe run-time complexity and, second, the definition of "computable, but intractable", we advise against tackling this in secondary education. Even if familiarity with polynomials can be assumed, the definition of O(f(n)) for time complexity is hard for pupils to grasp and should not be thrust upon them.

2. 6 Information, language, alphabet

The **representation of information** by a code, and the distinction between the form of this representation and its significance, i.e. between **syntax** and **semantics**, are further basic concepts in informatics and imply the definition of information in contrast to **data** and **knowledge**. The concept of **language** is closely linked to syntax and semantics; this includes programming languages, since syntax is a set of rules for constructing words, plus the rule that only words constructed in this way belong to the language. Semantics defines the meaning of these **words** in a language.

Then again, language involves the concept of an **alphabet**, since a language consists of a set of words over the alphabet, while an alphabet is defined as a set of symbols drawn from a supply of signs.

While presenting the concept of the syntax of a language, one is bound to raise the issue of how to describe syntax. This leads us on to “metalanguage”, and we recall that when we were discussing models we referred to a metamodel, as diagrammed in Fig. 1. And we also briefly referred, in our treatment of recursion, to EBNF, a concept of a metalanguage to describe syntax.

It seems clear, though, that while there are no didactic snags involved in presenting the concepts of an alphabet, a code and formal languages in secondary education, given their direct relevance to practical work (programming languages), one runs up against the limits of what is feasible in the case of metalanguages such as EBNF. If one decides to avoid programming languages altogether as instances of formal languages at this stage, possible alternatives are: the rules for writing syntactically correct mathematical formulae or musical notation. The latter is particularly suitable, inasmuch as it includes semantic annotations (volume: piano, forte; tempo: presto, etc.)!

2. 7 Relations

Of course a **classification** of data with respect to their properties, their **structure** and their **relations** belongs to the concepts of long-term validity with which properties such as **symmetry** or **equivalence**, and thus **equivalence classes**, can be explained.

The list of concepts given here is purely exemplary and anything but exhaustive; it is meant to encourage further discussion. However, our aim is to show that in the context of all-round education informatics teaching must be concerned not with technological artefacts, but with concepts of long-term validity, and can at the same time be organized to link up with other subjects (here with mathematics); this also applies in reverse.

3 Ways of putting the new media to work

From the various figures it is already clear to what extent the new media and eLearning can help to represent these “unchanging values” in informatics more effectively. At FIM and also at IFI eLearning has been an important issue for years now; at FIM the first steps in this direction were taken 20 years ago, when CBT (Computer-Based Training) courses (concerned with programming, operating systems etc.) were developed and offered as an enhancement of traditional teaching.

3.1 What has been developed

From the focus on eLearning several tools have taken shape; these have been in use in teaching for some years now. In particular, FIM has developed the WeLearn.Framework, which is constantly being enlarged in scope; it currently comprises the components, such as an open, easy-to-use eLearning environment (WeLearn) of universal applicability; didactic models for use at universities, in schools and in adult education; various tools and courses (in particular to implement our ideas about introducing students to informatics) to enhance teaching in the final years of secondary education.

Here we draw attention to [5],[8] and [10]. One study [7] has investigated how well the learning material and the learning environment provided were accepted.

3.2 "Propaedeutic in Informatics"

A key element in realizing our ideas about introducing informatics consists of specially prepared teaching and learning material available to students both via the WeLearn platform and on CD. "Propaedeutic in Informatics" is an introductory course for informatics students held by FIM at the JKU Linz. It regularly takes place in the winter semester; and involves blended learning [4] as a didactic model: here lectures and phases of self-organized study alternate. In the summer semester the subject matter is treated again, for the benefit of working students, other latecomers and interested pupils in the final years of secondary education (see below); in this case, though, the course consists of a kick-off meeting followed exclusively by distance learning.

This course is provided not only at the JKU, but also – with a different setting – at the University of Zurich, where students of business informatics are familiarized with the topics discussed here, using the same electronic material. Parts of it have also been successfully incorporated into an academically oriented course at the FH Vorarlberg.

The electronic material currently available comprises:

- A study guide: guidance for self-organized study and an explanation of parts of the subject matter, presented in the form of a dialogue between youngsters, and aimed particularly at pupils in the final years of secondary education
- The entire study material in the form of illustrated, partly interactive HTML pages
- The study material in full as text, also available as printed lecture notes
- The full set of transparencies for individual lectures
- Applets, on the basis of which students can carry out experiments and simulations and thus penetrate the subject matter. The applets discussed in chapter 2 are included here.

As regards teaching in secondary education, the following should be borne in mind:

Parallel to the above forms, the electronic material is also issued to secondary schools, where it can be used for teaching informatics/in preparation for Informatics A level (see below). Secondary-school teachers with a teaching qualification in informatics use the eLearning material (available on CD) in class, or have installed their own WeLearn server, via which they not only make the study material available but also help their pupils with queries, by means of newsgroups. Attention should be drawn to the following rule at the JKU Linz: Students commencing a degree course in informatics at the JKU after passing Informatics A level need not attend the preparatory course in informatics, provided that the subject matter for A level roughly corresponds to the scope of the basic principles presented in this paper.

4 Conclusion and outlook

People often say we live in a particularly fast-moving age – and this is especially true of the still young discipline of informatics. If we date the breakthrough in informatics to the 1960's, its history goes back less than 50 years, compared with a few thousand years in the case of mathematics. Informatics has developed extremely rapidly; particularly in the software field, the number of products available goes up by leaps and bounds, while their half-life diminishes dramatically. It thus seems logical and necessary to concentrate on the basic concepts, particularly in the field of secondary education. The fact is that purely product-related knowledge and skills in the narrow sense are inadequate, and in some cases already obsolete before pupils leave school. A more systematic grasp of these concepts and their interrelations is therefore not just desirable, but essential.

Literature

1. Garey M.R., Johnson.D.S.: Computers and Intractability: A guide to the Theory of NP-Completeness, W.H. Freeman, San Francisco, 1979
2. Hubwieser P.: Modellierung in der Schulinformatik. LOG IN 19, Heft 1. S.24-29, 1999
3. Informatik als Grundbildung; Informatik Spektrum, Band 27, Heft 2-4, 2004
4. Loidl, S.: The Beautiful but challenging World of Elearning. In Auer, M. E. and Auer, U., editors, International Conference on Interactive Computer Aided Learning, The Future of Learning, Villach Austria, Kassel university press 2004, ISBN 3-89958-089-3
5. Loidl, S., Sonntag, M.: Using metadata in creating offline views of e-learning content; in: Auer, M., Auer, U. (Ed.): ICL; Learning Objects and Reusability of Content, Kassel university press 2003
6. Loidl, S. Mühlbacher, J, Schauer, H.: Preparatory Knowledge: Propaedeutic in Informatics, Propädeutisches Informatikwissen, http://welearn-lavista.fim.uni-linz.ac.at/ (english/german), 2004
7. Mühlbacher, J., Mühlbacher, S.C., Loidl, S.: Learning Arrangements and Settings for Distance Teaching/Coaching/Learning: Best Practice Report. In Hofer, C., Chroust, G. (Ed.) IDIMT – 2002
8. Paramythis, A., Loidl, S.: Adaptive Learning Environments and e-Learning Standards; in: Roy Williams (Eds.): Proceedings of the 2nd European Conference on e-Learning, Glasgow, 2003
9. Perlis, A.J.: Epigrams on programming. SIGPLAN Notices, 17 (9),1982
10.Propädeutikum aus Informatik, http://welearn.fim.uni-linz.ac.at, 2004
11. Rosen, K.H.: Discrete Mathmatics and its Applications, 5th Edition, McGraw-Hill, 2003
12. Schauer, H.: Langlebige Standards in einer schnelllebigen Welt, CD Austria, 5/2004