Informatics - The Science of Minimal Systems with Maximal Complexity

Andreas Schwill
Institut für Informatik, Universität Potsdam, August-Bebel-Str. 89, D-14482 Potsdam, Germany

schwill@cs.uni-potsdam.de

Key words: Fundamental Ideas, Curriculum Research, Minimalism, Construction Kit

Abstract: It is a fundamental idea of computer science to search for, define, analyze, and operate with construction kits consisting of small sets of basic building blocks and a small number of operations to combine the building blocks to larger objects. While the construction kit is mostly simple, it often defines a vast, complex field that consists of all possible objects that can be built from the building blocks by using any (finite) sequence of combinations of operators. This idea affects and structures many areas of computer science. We present examples from several fields, including imperative and functional programming languages, computable functions, Turing and register machines, Boolean functions, data types, object-oriented programming, characterisations of formal languages along with examples from other disciplines. How can informatics lessons profit? If lessons are oriented towards a fundamental idea, the idea may explain, structure, and integrate many different informatics subjects and phenomena by a single recurring scheme. On the other hand, the construction kit principle belongs to the sphere of everyday thinking so students already have a basic intuition of the concept which may enhance their understanding when entering any field where the idea applies.

1. INTRODUCTION

In recent years we have elaborated Bruner's concept of fundamental ideas and made it accessible for informatics lessons (Bruner 1960). Here we

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35663-1_34

T. J. van Weert et al. (eds.), *Informatics and the Digital Society*

consider in detail a fundamental idea of computer science - orthogonalization - and show that it has a wide area of application and may guide many fields of school informatics.

By *orthogonalization* of a field Δ, following a term in linear algebra, we denote the definition of a number of basic elements Δ_e of the field along with a set K of operations ($K=\{K_1,...,K_n\}$, n small) on the basis each as small and simple as possible, such that every other object of the field may be generated by finitely many applications of operations on the basic elements (Figure 1). The result is a minimal generating system $B=(\Delta_e,K)$, consisting of the basis and the operations, that may be considered as a construction kit for the field.

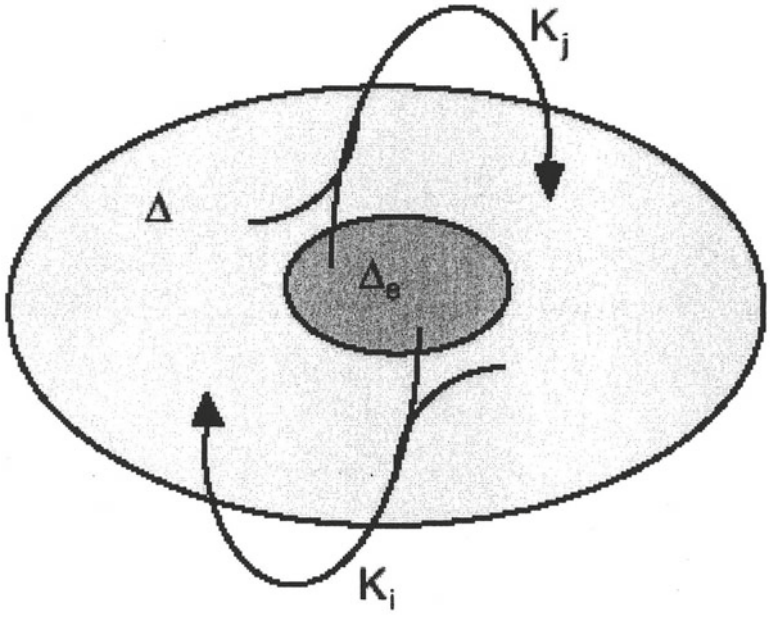

Figure 1. Principle of construction kits

We analyse orthogonalization with respect to didactic criteria and illustrate its relevance by presenting examples and applications in several areas of informatics.

We use the term *complex* to denote systems that are vast and diverse in their inner structure, while we call descriptions of systems *complicated* if they are vast and varied and hard to grasp. There is no direct relationship between the complexity of a system and the complication of its description. While the system, particularly a real life system, may be complex in nature,

it may have a simple, short description. We must avoid complicated descriptions if the systems are simple, and search for descriptions as minimal as possible if the systems are complex, to be able to understand, master, or manage them.

2. BACKGROUND

In 1960 Bruner formulated the teaching principle that lessons should predominantly orient towards the structure (the so-called *fundamental ideas*) of science. In recent years we have adopted the concept, made it and the relevant notions precise, and transferred it to informatics lessons by defining fundamental ideas of informatics (including algorithmization, structural dissection, (artificial) languages and orthogonalization). We have also proposed lessons suitable for teaching certain ideas in school (Schwill 1993; Schwill 1997).

We define a fundamental idea as a schema for thinking, acting, describing or explaining which satisfies the following criteria:

Horizontal Criterion. A fundamental idea is applicable or observable in multiple ways and in different areas of informatics. It organizes and integrates a wealth of phenomena.

Vertical Criterion. A fundamental idea may be demonstrated and taught on every intellectual level - "any subject can be taught effectively in some intellectually honest form to any child at any stage of development" (Bruner 1960). A central methodological means guiding the education of fundamental ideas on different levels of understanding is the *spiral principle.* This recommends three representations of concepts to be learned - *enactive* (lower level), *iconic* (medium level), and *symbolic* (highest level).

Criterion of Time. A fundamental idea can be clearly observed in the historical development of computer science and will stay relevant in the future. Importantly lessons based on fundamental ideas will not become dated as quickly as conventional lessons - a major advantage in teaching informatics which exhibits such dynamic evolution.

Criterion of Sense. A fundamental idea has meaning in everyday life and is related to ordinary language and thinking. Only a precise definition turns an idea "with sense" into an exact concept "without sense". When we teach a fundamental idea early in the student's schooling, we may give a first impression of the idea by using everyday situations as starting points for lessons.

3. MINIMAL SYSTEMS IN ARTS, SCIENCES, AND INDUSTRY

Why do minimal systems in the form of construction kits play such an important role in the arts, sciences, and industry? There may be two main reasons. Firstly ecological/economical causality. Nature and industry develop evolutionarily to derive the maximum result by spending a minimum of resources. The other reason might be the limited capacity of the human brain, particularly short-term memory, so concepts should have small descriptions to be manageable by humans. Miller's finding that short-term memory capacity is 7±2 chunks (or self-contained objects) may explain why many minimal systems comprise less than 10 basic elements and operations (Miller 1956).

The following minimal systems found in arts, sciences, society, and industry also verify the Criterion of Sense for orthogonalization.

3.1 Orthogonalization in industry

A modern example is Volkswagen's platform strategy announced in 1997. By using only four platforms that contain 60% of the car's parts, including the chassis, engine, brake system and gearbox, Volkswagen wishes to eventually produce 51 different models. Further examples of orthogonalization occur in concepts of lean management or lean production.

3.2 Orthogonalization in society

Orthogonalization may be found in public management (Biedenkopf 1994). The complexity of problems in developed industrial societies permanently increases - in health, pensions, tax, or unemployment systems. At the same time the complication of methods, processes, and legal regulations grows to manage this increase in complexity. This accelerates the waste of financial, personal, and natural resources and produces little or no visible benefit to the public. Biedenkopf favours a reduction of the complication of these methods and processes to a collection of a few simple and clear, yet powerful, social principles. These may be arbitrarily combined and flexibly tailored to address upcoming social problems.

3.3 Orthogonalization in arts

Minimal art, a movement and style that emerged in the 1960s, stresses the idea of reducing art to a minimum number of elementary objects or

primary structures, such as colours, shapes, lines and textures. In creative formal combination they produce a maximum of different pictures or sculptures. While traditional art uses an analytical approach, often attempting to represent real objects or experiences (faces and landscapes), minimal art works constructively only producing artwork that can be generated by its "construction kit".

3.4 Orthogonalization in music

Minimal music is likewise characterised by maximal simplicity and reduction of basic musical material - tones, rhythms, musical patterns, and compositorial means. The major operations to create music from the basic material are repetition of patterns, phase shifting, overlaying, stressing, adding single notes to slightly change rhythms, and sequences of tones over time. Although its underlying structure is simple, minimal music leads to a highly creative feeling of sound. It has spread worldwide in the form of techno music and its branches of trance, house, and ambient. These stress the beat and use repetitive sound patterns and rhythms to create hypnotic and ecstatic experiences.

3.5 Orthogonalization in linguistics

Umberto Eco wrote about a long-lasting search for the perfect language, a universal language in which every object, thought, idea, feeling may be unambiguously expressed (Eco 1995). A typical orthogonalization approach in the 17^{th} century concerned a "construction kit" of forty *categories* subdivided into 251 *differences*, in turn subdivided into 2030 *species*. Each category was assigned a two-letter syllable, each difference a consonant, and each species a vowel or diphthong. So de stood for the category "element"; deb for the first element or "fire"; deba for a part of the element, a flame.

3.6 Orthogonalization in mathematics

Simple co*nstruc*tion *kits* that generate highly complex fields arise in chaos theory. Consider Julia sets that are generated by iteration of very simple functions f in the complex plane C such as

f: C→C defined by $f(x)=x^2+c$ where c is a constant.

For certain x either f^n is bounded or unbounded and the Julia set associated to f is defined to be the set of complex values x where f^n lies on the boundary.

By introducing *Kolmogorov complexity* (Li and Vitányi 1997) mathematics has formalised the notions of complexity and complication of a system. For an object or system s we define the Kolmogorov complexity C(s) to be the length (in bits) of the shortest algorithmic description A of s. The algorithm A of length C(s) A produces s.

On the one hand, an object may be regarded simple if its Kolmogorov complexity is small, and systems are obviously complicated if their description is longer than necessary, i.e. longer than C(s). Random objects, a random sequence s of bits, say, are most complex because we are not able to find a description that is considerably shorter than the object s itself. Rather every algorithm A has to contain and enumerate each and every bit of the sequence and, thus, is about as long as the sequence itself, i.e. $C(s) \gg |s|$.

4. MINIMAL SYSTEMS IN INFORMATICS

In at least two approaches orthogonalization conquers the methodology of informatics. The first concerns the minimisation of resources: sometimes using very complicated ideas and constructions one tries to develop an algorithm that needs as little time and storage as possible to solve a problem. The problem itself may be simple in structure and easy to understand.

On the other hand the objective may be to minimise descriptions of systems. There are approaches in computer science that try to describe the most complex given structures by concepts as simple and minimal as possible or, vice versa, try to define very small orthogonal systems and then study the structures they generate. In the following we will focus on the descriptional aspects of minimalism.

We may consider informatics as a science that has developed, or adopted, the most beautiful minimal system of 0 and 1. All subjects of relevance are eventually mapped into a sequence of zeros and ones for execution by a digital computer.

There are many nice minimal systems. The following may be subdivided into two groups. One group contains minimal systems that define the executing machine or parts or models of it. The other covers systems for modelling the real world by a computer program consisting of data structures representing the static elements of the original and control structures realising its dynamic elements. For both parts of the model, depending on the underlying programming paradigm, informatics has defined fundamental minimal systems in the form of construction kits.

4.1 Orthogonalization in machines – register machines with 2 registers

The register machine (Figure 2) is an automaton whose memory contains a fixed number, say m, of registers each able to store an arbitrarily large natural number.

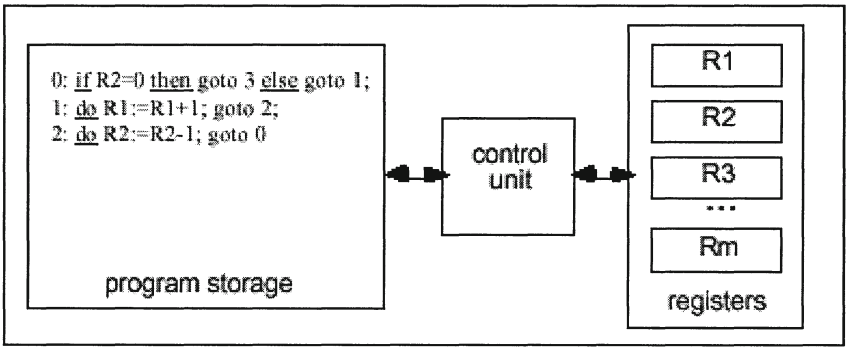

Figure 2. Register machine

The machine can perform only three operations on a register: addition of 1, subtraction of 1 if the register contains a number >1, and zero-test. A register machine program is a sequence of statements of either

i: do f; goto j
or i: if t then goto j else goto k

where i and j are labels, f is an operation on a register, and t is the zero-test on a register.

A register machine works as follows: At the beginning r≤m input values are stored in the first r registers. All other registers are set to 0. The machine executes the program beginning with statement labelled 0. The machine stops if it is to branch to an undefined label. The contents of the first s≤m registers are considered as the output. So the machine computes a function f: $IN_0^r \rightarrow IN_0^s$.

This simple machine model is extremely powerful: a register machine with only two registers can compute any computable function and so can simulate any computer. The key idea is clever coding of the storage into a single register. The coding is also an application of orthogonalization.

Orthogonalization does not stop here. A further step would be to reduce the number of instructions leading to a *single instruction computer* with equivalent power whose only instruction is

subtract 1 and jump to <label> if negative.

4.2 Orthogonalization in imperative programming – control structures

According to our observations above we can derive well-known construction kits for imperative control structures. Each of the following pairs of basic elements and operations forms a construction kit:

({assignment, goto}, {concatenation, if_then_else_fi})
or ({assignment}, {concatenation, while_do_od, if_then_else_fi})
but not ({assignment}, {concatenation, for_to_do_od, if_then_else_fi}).

The complexity of the system defined by these small construction kits may be exemplified by the following program (x is of type integer):

```
read(x);
while x>1 do
  if even(x) then x:=x/2 else x:=3*x+1 fi ;
od ;
write("I am done").
```

The program implements the Collatz function - the 3x+1- problem or Syracuse problem. For over 60 years it is not known if the program terminates for all inputs.

4.3 Orthogonalization in functional programming - λ-calculus

Functional programming is based on the λ-calculus, a mathematical formalism developed to define and use functions that are given by algorithms (Barendregt 1984). The λ-calculus is a classical minimal system that uses simple yet very powerful basic elements and operations.

Expressions or λ-terms, are defined inductively. Given a set of variables $X=\{x_1,x_2,x_3,...\}$, then it holds:

1. Each variable $x \in X$ is a λ-term (elementary λ-terms).

2. If M and N are λ-terms, then so is (MN). (MN) denotes the *application* of a λ-term M to a λ-term N, usually written M(N).

3. If $x \in X$ is a variable and M is a λ-term, then (λx.M) is a λ-term. This rule describes abstraction of a λ-term M to a function λx.M, where x is the formal parameter and M is the function body. λ stands for the keyword function in programming languages. The dot separates the function head and function body. In common with programming languages abstraction describes the parametrization of an expression like x+5 and transfer to a function f(x)=x+5. Note that a λ-term is not identified by a name and has to be completely written down wherever it is used.

Application of a λ-term to an argument is defined by the so-called β-rule which describes the replacement of formal parameters by actual ones and is defined by

$$((\lambda x.M)N)=M[x \leftarrow N],$$

where we apply the function (λx.M) with formal parameter x and body M to an actual parameter N. The rule says the entire term may be replaced by M[x←N], the term obtained by textually replacing parameter x by N everywhere in the body.

Despite its extreme simplicity λ-calculus is powerful enough to describe all computable functions, i.e. λ-calculus may be considered a fully functional programming language.

4.4 Orthogonalization in modelling – data types

The complex real world modelled by a software system usually consists of much data in many different forms and representations. Informatics has developed a method to cope with this unstructured data in a systematic way using simple construction kits. These allow any data to be described and eventually mapped into a computer.

The standard kit in programming languages is

({character, integer, real, boolean},
{aggregation, generalisation, recursion, functional spaces})

and consists of the elementary data types character, integer, real and boolean and a number of operations, so-called *constructors*, such as aggregation, generalisation, recursion, and construction of functional spaces (Figure 3).

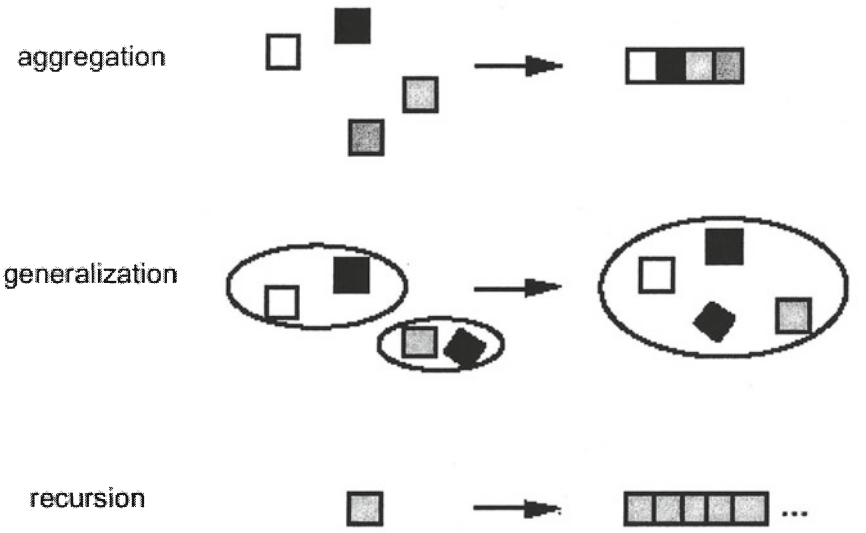

Figure 3. Effects of aggregation, generalisation and recursion

Further approaches of orthogonalization in informatics leading to minimal systems may be observed in the following:

- the universal Turing machine may be considered the one-element basis of the class of all Turing machines or even of the class of computable functions. The complexity of Turing machines or computable functions in relation to their minimal basis is best experienced by a Busy-Beaver-Turing machine;
- primitive-recursive and μ-recursive functions are defined inductively. There is a set of basic functions, e.g. constant function, successor function, and a set of operations, e.g. composition, μ-operator, that form a generating set for the class of primitive-recursive and μ-recursive functions, respectively;
- object-oriented programming attempts to develop a collection of basic reusable modules that may be configured for the concrete application area. Object-oriented design uses design patterns that form an abstract language to describe solutions to recurring object-oriented design problems;
- programming languages are often designed according to the orthogonalization principle - "When there is one good way to express something in Eiffel, there are often not two." (Meyer 1989)

- fundamental basis of the set of Boolean functions and technical foundation of computer systems are the function sets {and, or, not} or {and, not} or {nand} or {nor}. The functions of a set have to satisfy five simple criteria in order to define a generating set of all Boolean functions;
- construction kits consisting of tiny, highly modular, reusable components are being developed for operating systems. These components have well-defined operational behaviour that may be configured and functionally enriched by minimal extensions;
- homomorphisms, the bracket language and regular languages define a construction kit for the class of context-free languages such that any context-free language may be considered as a coding of correct bracket expressions with some regular languages in between;
- to prove that a system is not orthogonal one often uses the idea of *emulation*: Given a generating system, if one of the basic elements or operations may be realized by the others then the system is not minimal.

5. MINIMAL SYSTEMS IN INFORMATICS LESSONS

The above examples support the thesis that orthogonalization is a fundamental idea of informatics that satisfies both the Horizontal Criterion and the Criterion of Sense. If so the idea must be integrated into informatics lessons and taught in a way that clearly illustrates its property of linking different informatics subjects and integrating diverse phenomena and methods under a common concept.

To achieve this orthogonalization and minimal systems should be dealt with in schools according to the following guidelines:

- Orthogonalization should be introduced at every stage of intellectual development and, following the spiral principle, examined with increasing level of elaboration and formalization - starting in primary school.
- It must be made clear where exactly the idea should be introduced and applied in the informatics curriculum and what advantage its use offers for solving the problem.
- The appropriate minimal system, its basic elements and operations, and their inter-relationships must be explained in detail.
- The inner structure of the field defined by the minimal system must be analysed.

- It is important that students are shown how this approach and earlier applications of minimal systems in other subjects coincide or differ.

The key concept is the Vertical Criterion. That enables the idea to be taught on every school level "in some intellectually honest form". Some examples mentioned above may be reduced to the primary school level.

From the methodical point of view orthogonalization appears to be a valuable subject. Students usually have gained early experiences with construction kits in their everyday life e.g. LEGO. They may easily grasp small systems with few orthogonal basic elements and operations and, via constructivism, experiment with basis and operations and explore, possibly in project work, the space "spanned" by the construction kit and experience its vast complexity.

REFERENCES

Barendregt, H. P. (1984) *The Lambda calculus*. North-Holland.

Biedenkopf, K. (1994) Komplexität und Kompliziertheit. *Informatik Spektrum* 17, pp. 82-86.

Bruner, J. S. (1960) *The process of education*. Cambridge, Mass.

Eco, U. (1995) *The search for the perfect language*. Blackwell.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995) *Design patterns*. Addison Wesley.

Li, M. and Vitányi, P. (1997) *An introduction to Kolmogorov complexity and its applications*. Springer.

Meyer, B. (1989) From structured programming to object-oriented design: the road to Eiffel. *Structured Programming* 1, pp.19-39.

Miller, G. A. (1956) The magical number seven plus or minus two: Some limits on our capacity for processing information. *Psychological Review* 63, pp. 81-97.

Schwill, A. (1993) Fundamentale Ideen der Informatik. *Zentralblatt für Didaktik der Mathematik* 1, pp. 20-31.

Schwill, A. (1997) Computer science education based on fundamental ideas. In *Information Technology - Supporting change through teacher education*, D. Passey and B. Samways (eds.), Chapman Hall. pp. 285-291.