# Informatics - The Science of Minimal Systems with Maximal Complexity

**Andreas Schwill**
Institut für Informatik – Universität Potsdam
August-Bebel-Str. 89 – D-14482 Potsdam – Germany
Phone: +49-331-977-3100 – Fax: +49-331-977-3122
Web: www.informatikdidaktik.de
Email: schwill@cs.uni-potsdam.de

**Keywords**
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 and very 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, among them are imperative and functional programming languages, computable functions, Turing and register machines, Boolean functions, data types, object-oriented programming, characterizations of formal languages along with examples from other disciplines.
How can informatics lessons profit from this observation? On the one hand, 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, since an idea like the construction kit principle also belongs to the sphere of everyday thinking, students already have a basic intuition of the concept which may enhance their understanding when entering any of the fields where the idea applies.

## 1 Introduction

In recent years we have elaborated the concept of fundamental ideas (originally propagated by Bruner [Br60]) and made it accessible for informatics lessons. Here we consider in more detail a selected idea from the collection of fundamental ideas of computer science that we called 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 $\Delta_e$ of the field along with a set K of operations ($K=\{K_1,...,K_n\}$, n small) on the basis (s. Fig. 1), 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. So the result of orthogonalization is a minimal generating system $B=(\Delta_e,K)$, i.e. a pair consisting of the basis and the operations, that may be considered as a construction kit for the field.

We analyze the idea with respect to different didactic criteria and show the relevance of the idea by presenting examples and applications in several areas of informatics as well as in lessons.

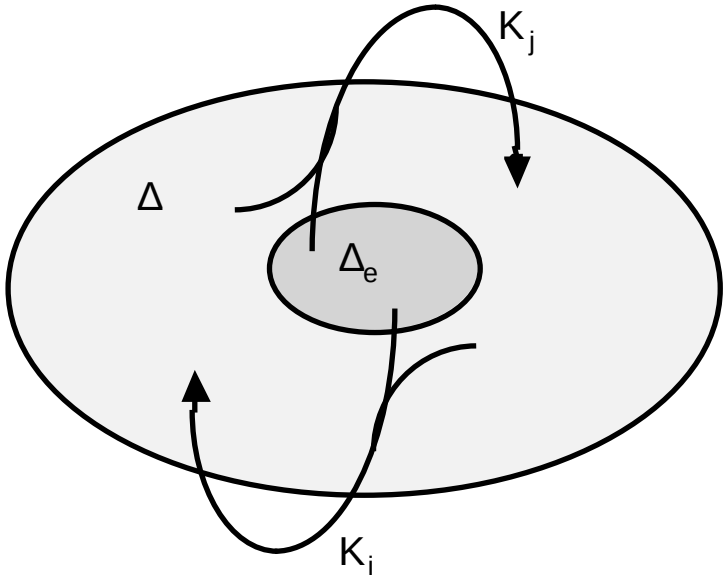


Fig. 1: Principle of construction kits

Throughout this paper 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. Obviously, there is no direct relation between the complexity of a system and the complication of its description. While the system, in particular, if it is a real life system, may be complex in nature, it may nevertheless have a simple and short description. What we have to avoid under any circumstances are complicated descriptions if the systems are simple, and what we have to search for are descriptions as minimal as possible if the systems are complex in order to be able to understand, master, or manage them.

## 2 Background

In 1960 J.S. Bruner [Br60] 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 (among them are algorithmization, structural dissection and (artificial) languages as well as orthogonalization) and proposing lessons suitable for teaching certain ideas in

school. Here we only give a rough overview of the concept. For details the reader is referred to [Sc93,Sc97].

We define a fundamental idea as: A schema for thinking, acting, describing or explaining which satisfies four different criteria.

*The 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. For orthogonalization this criterion is shown in Section 4.

*The Vertical Criterion*. A fundamental idea may be demonstrated and taught on every intellectual level following Bruner who said that "any subject can be taught effectively in some intellectually honest form to any child at any stage of development". A central methodical means guiding education of fundamental ideas on different levels is the *spiral principle* also recommending three representations of concepts to be learned: *enactive* on the lower level, *iconic* on a medium level, and *symbolic* on the highest level of understanding. For orthogonalization this criterion is shortly motivated in Section 5.

*The Criterion of Time*. A fundamental idea can be clearly observed in the historical development of computer science and will stay relevant in the future. This aspect is important since lessons based on fundamental ideas will not become antiquated as quickly as conventional lessons–a major advantage in teaching informatics with its dynamic evolution.

*The Criterion of Sense*. A fundamental idea also 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". So, whenever we have to teach a fundamental idea early in the student's schooling, we may give students a first impression of the idea by using everyday situations as starting points for lessons. For the idea of orthogonalization this criterion is made clear in Section 3.

## 3 Minimal Systems in Arts, Sciences, and Industry

Why do minimal systems in the form of construction kits play such an important role in all kinds of arts, sciences, and industry? There may be two main reasons. The first one is obviously an ecological/economical cause: Nature and industry develop evolutionary to get the maximum result by spending a minimum of resources. The other one might come from the limited capacity of the human brain, in particular of the short term memory, so that concepts one works with have to have small descriptions to be manageable by

humans. Miller’s result on the short-term memory capacity of 7±2 chunks [Mi56] might be a rough explanation for the fact that many minimal systems comprise less than 10 basic elements and operations.
In the following we shortly present some minimal systems in arts, sciences, society, and industry. These examples also verify the Criterion of Sense for orthogonalization.

**Orthogonalization in industry**

A modern example of industrial orthogonalization is Volkswagen's *platform strategy* announced in 1997 [Se97]. 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 occurrences of orthogonalization may be found in concepts of *lean management* or *lean production*.

**Orthogonalization in society**

An illustrating application of orthogonalization in public management is given by K. Biedenkopf [Bi84]. The complexity of problems in a developed industrial society permanently increases, e.g. in health, pension, tax, or unemployment systems. At the same time, often much faster, the complication of methods, processes, and law regulations grows in order to manage the increase in complexity. This leads to an accelerating waste of financial, personal, and natural resources which eventually produce almost no visible benefit for the public. Biedenkopf favors a reduction of the complication of these methods and processes to a collection of few simple and clear, yet powerful, social principles which may be arbitrarily combined and flexibly tailored to upcoming social problems.

**Orthogonalization in arts**

Minimal art, a movement and style that came up in the 1960s, stresses the idea of reducing art to a minimum number of elementary objects, such as colors, shapes, lines and textures (sometimes called "primary structures"), that in creative formal combination may produce a maximum of different pictures or sculptures. While traditional art uses an analytical approach often attempting to represent real objects or experiences, minimal

art works constructively only producing artwork that can be generated by its "construction kit".

**Orthogonalization in music**

Minimal music denotes a style that is likewise characterized by maximal simplicity and reduction of the basic musical material, i.e. 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 in order to slightly change rhythms and sequences of tones over time. Although simple in its underlying structure minimal music leads to a highly creative feeling of sound.

Minimal music has spread all over the world in the form of techno music and its branches trance, house, and ambient stressing the beat and using repeating sound patterns and rhythms to create hypnotic and ecstatic experiences.

**Orthogonalization in linguistics**

In his book U. Eco [Ec95] deals with the long-lasting search for a *perfect language*, a universal language in which every object, thought, idea, feeling may be unambiguously expressed. A typical orthogonalization approach is that of John Wilkins in the 17th century. His "construction kit" consists 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. For example: de stands for the category "element"; deb for the first of the elements, that is "fire"; deba for a part of the element fire, a flame.

**Orthogonalization in mathematics**

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

$f: \mathbb{C} \to \mathbb{C}$ 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* [LV97] mathematics has formalized 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. So by running 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. $C(s) \approx |s|$.

## 4 Minimal Systems in Informatics

There are at least two approaches where orthogonalization conquers the methods of informatics. The first one is the minimization 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, however, is often simple in structure and easy to understand. A typical example of this category is the union-find-problem [Me84].

On the other hand there is the objective to minimize descriptions of systems, i.e. there are approaches in computer science that either 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.

At first glance we may consider informatics as a science that has developed, or shall we say adopted, the most beautiful minimal system that we may think of: 0 and 1. All subjects of relevance are eventually mapped into a sequence of zeros and ones in order to be executed by a digital computer.

But there are many more nice minimal systems. The systems we consider in 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 group 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 realizing its dynamic elements. For both parts of the model, but depending on

the underlying programming paradigm, informatics has defined fundamental minimal systems in form of construction kits.

**Orthogonalization in machines – register machines with 2 registers**

The register machine (Fig. 2) is an automaton whose memory contains a fixed number, say m, of registers each able to store an arbitrarily large natural number. 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. Then the machine executes the program beginning with statement labelled 0. The machine stops if it is to branch to an undefined label. Then the contents of the first s≤m registers are considered as the output of the machine. So the machine computes a function f: $IN_0^r \rightarrow IN_0^s$.

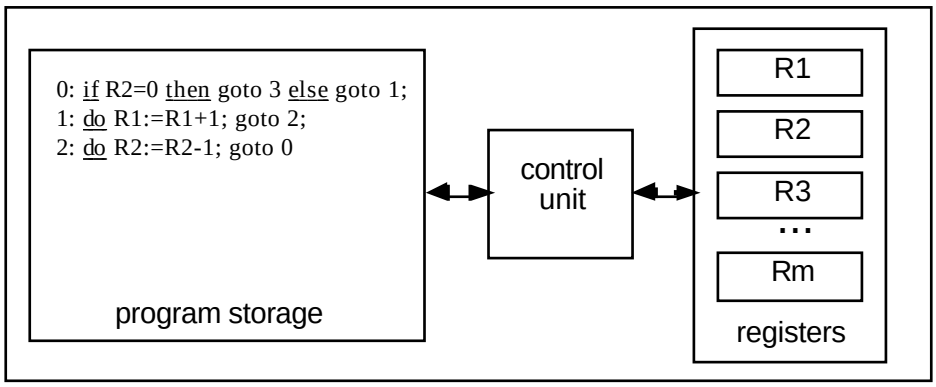


Fig. 2: Register machine

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

But 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.

### Orthogonalization in imperative programming – control structures

According to the 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}).

With the last system we cannot program conditional loops.

The complexity of the system defined by these small construction kits may be best experienced 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 (by L. Collatz, 1937), also known as 3x+1-problem or Syracuse problem. For more than 60 years now it is unknown whether or not the program terminates for all inputs.

### Orthogonalization in functional programming – λ-calculus

Functional programming (with LISP, ML, Haskell or other languages) is based on the λ-calculus, a mathematical formalism developed by A. Church in the 1930s to define and use functions that are given by algorithms [Ba84]. The λ-calculus is a classical minimal system that uses simple yet very powerful basic elements and operations.

Expressions in λ-calculus, so-called 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 in the form M(N).
3. If x 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 the keyword function in programming languages. The dot separates function head and function body. As of 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, so it 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 that the entire term may be replaced by M[x ← N], that is the term that is obtained by textually replacing parameter x by N everywhere in the body. This rule excludes situations where M itself contains λ-abstractions, but for our purposes here it is sufficient.

Examples:

1. (λx.z) defines a constant function, since for any y using the β-rule ((λx.z)y) may be evaluated to z.
2. (λx.x) defines the identity function, since for any y using the β-rule ((λx.x)y) may be evaluated to y.

Despite its extreme simplicity it has been shown by A. Church that λ-calculus is powerful enough to describe all computable functions, i.e. λ-calculus may be considered a fully-functional programming language. The proof is by simulation of register machines. For it one has to encode natural numbers into λ-terms, then simulate the register operators +1 and -1, etc. A natural approach is to identify

| | | | |
|---|---|---|---|
| 0 | by the λ-term | (λf.(λx.x)), | |
| 1 | by the λ-term | (λf.(λx.(fx))), | |
| 2 | by the λ-term | (λf.(λx.(f(fx)))), | |
| 3 | by the λ-term | (λf.(λx.(f(f(fx))))) | and so on. |

Then we can define arithmetic operations by λ-terms, e.g. the successor function +1 on natural numbers by

(λn.(λf.(λx.(f(n(fx)))))).

**Orthogonalization in modelling – data types**

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

The standard kit in programming languages is

({character, integer, real, boolean},

{aggregation, generalization, 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, generalization, recursion, and construction of functional spaces (Fig. 3).

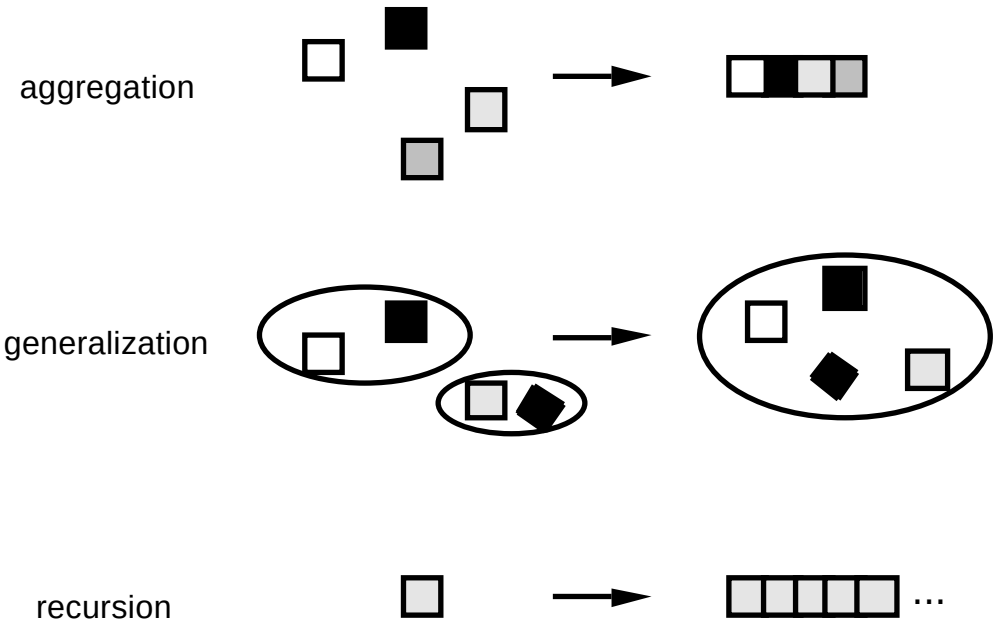


Fig. 3: Effects of aggregation, generalization, and recursion

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

· 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 (e.g. the model-view-controller pattern [Ga95]) that form an abstract language to describe solutions to recurring object-oriented design problems;
- programming languages are often designed according to the orthogonalization principle. Very illustrating is a quote of B. Meyer [Me89] for motivating his conception of the programming language Eiffel: "When there is one good way to express something in Eiffel, there are often not two" (p. 37);
- 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}. E.L. Post and S.V. Yablonsky have found five simple criteria that the functions of a set have to satisfy in order to define a generating set of all Boolean functions;
- it is attempted to develop construction kits for operating systems consisting of tiny, highly modular and reusable components with well-defined operational behaviour that may be configured and functionally enriched by minimal extensions according to the future application area.
- 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 (Theorem of Chomsky and Schützenberger).
- In order 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 from informatics and other disciplines 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 definitely integrated into informatics lessons and taught in a way that makes visible its property of linking together different informatics subjects and integrating diverse phenomena and methods under a common concept.

To achieve this it is necessary to deal in school with orthogonalization and minimal systems according to the following guidelines:

1. Orthogonalization has to be treated on every stage of intellectual development along the spiral principle with increasing level of elaboration and formalization starting in primary school.
2. One has to make clear where exactly the idea appears and applies in the concrete subject and what advantage it gives for solving the problem.
3. It has to be explained in detail what the current minimal system is, what its basic elements and operations are and how they work together.
4. One has to analyze in more or less detail the inner structure of the field the minimal system defines.
5. One has to show how this approach and earlier applications of the idea 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" and some examples mentioned above may be reduced to the primary school level.

Also from the methodical point of view orthogonalization appears to be a precious subject: Students usually have gained early experiences with construction kits in their everyday life ("LEGO") and may easily grasp small systems with few orthogonal basic elements and operations and, along constructivism, playfully experiment with basis and operations and explore, maybe in projects, the space "spanned" by the construction kit and experience its possibly vast complexity. Explanations of the teacher are hardly necessary.

### References

[Ba84] H.P. Barendregt: The Lambda calculus. North-Holland 1984
[Bi94] K. Biedenkopf: Komplexität und Kompliziertheit. Informatik Spektrum 17 (1994) 82-86
[Br60] J.S. Bruner: The process of education. Cambridge Mass. 1960

[Ec95] U. Eco: The search for the perfect language. Blackwell 1995
[Ga95] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design patterns. Addison Wesley (1995)
[LV97] M. Li, P. Vitányi: An introduction to Kolmogorov complexity and its applications. Springer 1997
[Me84] K. Mehlhorn: Data structures and algorithms 1: Sorting and searching. Springer 1984
[Me95] W. Mertens: American minimal music. Pro AmMusic Resources 1995
[Me89] B. Meyer: From structured programming to object-oriented design: the road to Eiffel. Structured Programming 1 (1989) 19-39
[Me01] J.S. Meyer: Minimalism: Art and polemics in the sixties. Yale Univ. Press 2001
[Mi56] G.A. Miller: The magical number seven plus or minus two: Some limits on our capacity for processing information. Psychological Review 63 (1956) 81-97
[Sc93] A. Schwill: Fundamentale Ideen der Informatik. Zentralblatt für Didaktik der Mathematik 1 (1993) 20-31
[Sc97] A. Schwill: Computer science education based on fundamental ideas. In: Information Technology - Supporting change through teacher education (D. Passey, B. Samways, eds.), Chapman Hall (1997) 285-291
[Se97] M.S. Serrill: Germany's new drive. Time Magazine 150,14 (1997)