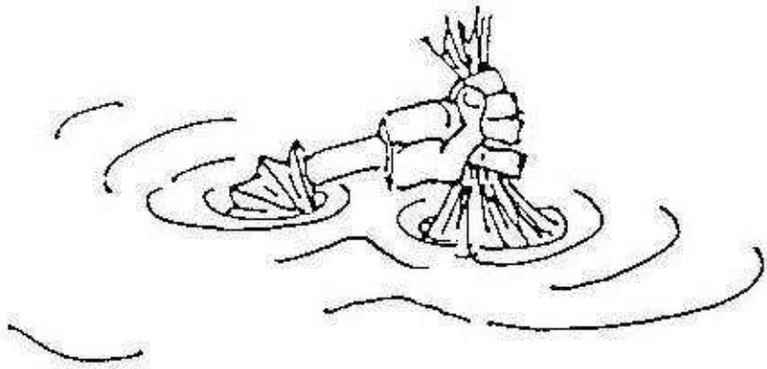
Evolutionary Principles in Self-Referential Learning (Diploma Thesis)

Jürgen Schmidhuber

Technische Universität München

May 14, 1987



Evolutionary Principles in Self-Referential Learning (Diploma Thesis)

Jürgen Schmidhuber
Technische Universität München

Abstract

There exists a number of algorithms which encapsulate parts of the behaviour we call learning. Programs have been written that do learning by chunking, generalization, certain kinds of analogical matching etc. These algorithms work nicely in certain well-chosen domains but fail in many others.

The fundamental statement of this thesis is that we can not capture the essence of learning by relying on a small number of algorithms. Rather on the contrary there is a need for a whole bunch of context-dependent learning strategies to acquire domain specific information by using information that is already available. Because of the complexity and richness of these strategies and their triggering conditions the obvious escape seems to be: Giving a system the ability to learn the methods how to learn, too. A system with such meta-learning capabilities should view every problem as consisting out of at least two problems: Solving it, and improving the strategies employed to solve it. Of course we do not want to stop at the first meta-level !

The only approach to achieve meta-capacity seems to be: 'Closing (feeding back) some initial strategies onto themselves' so that more complicated and better suited ones can evolve. This requires an initial representation of the system that allows it to introspect and manipulate all of its relevant parts. Furthermore some kind of evolutionary pressure is needed to force the system to organize itself in a way that makes sense in the environment it is living in. The fundamental role of the very general principle called evolution and its deep interrelations to the field of learning will be emphasized. Connections to v.Weizsäckers understanding of pragmatic information as well as to Piagets model of equilibration will be shown.

Two approaches to the goal of learning how to learn will be presented, both of them being inspired from seemingly rather different corners of artificial intelligence and cognitive science: One source of ideas is to be found in symbol-manipulative learning programs as EURISKO and CYRANO, the other one in work done on neuronal nets, associative networks, genetical algorithms and other 'weak' methods (an analogy to geometric fractals will be drawn). In this context it is argued that object oriented programming and neuronal nets have more things in common than is usually assumed.

The second approach which leads to the notion of self-referential associating learning mechanisms (SALMs, PSALMs) is illustrated by the implementation

of a simple self-referential and self-extending language and a few empirical results obtained by putting pressure on that language to organize itself. However, these results are not suited to show concrete cases of self-reference. It will be made obvious that the available machine capacity is clearly below the level that would be necessary to make the creation of 'semantic self-reference' likely (on the basis of the second approach and within a reasonable time). Thus this paper tends to having inspiring character rather than presenting a practical guidance to universal learning capabilities.

A table of contents is supplied at the end of this work.

Keywords

self-reference, introspection, learning, meta, evolution, associative nets, neuronal nets, genetical algorithm, bucket brigade, SALM, PSALM, EURISKO, fractals.

1. Introduction

Within the AI community there exists the general agreement that the most important part of intelligent behaviour at the same time is the one understood least at all: the ability to learn. Although systems have been built that are able to learn certain domain concepts or discover certain algorithms, the strong suspicion remains that the essential ingredients of self-reflexivity and full introspection still have to be discovered.

In some of his famous experiments Piaget has shown that children below the age of 5 do not dispose of the concept of generalization, let alone the qualification for logical thinking that is not distinct before the age of 11. But all the computer programs we create (including those we write to do learning) are based on logical thinking (are they ?). These programs are far away from showing such impressive performance in many domains as a little baby does, although babies tend to draw conclusions neither from general statements to more special ones (as a logician or an automatic theorem prover would do) nor from special statements to more general ones (as a physician would do), but from special ones to special ones (as everybody does but no one would admit) !

This special-to-special-thinking described by Piaget is closely related to the notions involved with the phenomenons often summarized under the diffuse name analogy. Some definitions of analogy in terms of first order logic can be found in [ECAI 86]. Nearly as many critical statements about these definitions can be found, too. The reason is that frequently good-looking definitions are too specific. They make sense in special cases of relations between some objects, but in many other cases they do not capture connections we intuitively would call analogical, too.

Most approaches to make programs learn are based on symbol-manipulative languages like LISP or PROLOG that do not formally distinguish between algorithmic and data structures. Usually some basic method like filing unknown or only partly known objects in isa-hierarchies (generalization, specialization) is the decisive factor for the performance of the system [Michalsky 84][Lenat 82a]. Sussmans program [Sussman 75] also learns by refining general procedural knowledge while "pushing it down the hierarchy".

Another method considered to be important is chunking (building macros): If sequences of actions often have proven to be successful in a certain domain they are packed into a new procedure that again may be used as part of a higher-level-chunk (Newell in [Michalsky 84]).

Some attempts have been made to do learning by analogy [ECAI 86], often by trying to find some kind of match between existing structures in predicate calculus. I adopt the widespread view that it is most important to find out how analogy works, but furthermore I want to argue that formal languages like predicate calculus are not very well suited to understand analogy. This opinion is compatible with Minsky's comment cited in [Wallich ?]: " Formalization [in AI] has been a disaster."

The main motivation behind the work presented here is the belief that there is a large number of ways for a learning system to make use of the things it

already has learned, call it learning by analogy or whatever you like. It is assumed that this number is indeed so large and that the set of strategies to acquire new information in a domain is so diverse and context–dependent that there is little chance in trying to supply the initial system with a few plausible algorithms and hoping that they will cover everything.

The strongest support for this assumption is provided by empirical results delivered by already existing programs. Especially the probably most outstanding effort to do symbol–manipulative learning, Lenat's discovery program EURISKO [Lenat 82a] which actually shows some kind of introspective behaviour by not only learning domain concepts but also the heuristics to acquire domain concepts, illustrates the need for systems that are able to learn how to learn in two different ways:

On one side EURISKO elicits the advantages of including the domain of creating heuristics in the set of domains to be explored by improving its performance within the other domains. Not just only by redefinition of its search space(s) depending on newly created concepts but also by change of the ways how to progress in these spaces EURISKO keeps up to date with its discoveries.

On the other side EURISKO's activities are constrained by the fundamental process of incorporating concepts (including heuristics) somewhere into the large generalization hierarchy. Simon, Bledsoe and Lenat interpreted the results and found something they called the 'shallow tree problem'. This term stands for the observation that the developing lattice of concepts and heuristics did not reflect very well the 'real' connections between them. In fact many of the heuristics found in higher levels of the lattice were in no way more general than others found below. Simon et al. were able to rearrange the generalization tree into an equivalent, very bushy one which had a small depth. But a great variety of other relations besides the inclusion held between certain nodes of the tree, thus 'structuring' the knowledge. It was concluded that not so much genl./spec. but analogy is the natural way to organize learned knowledge, and it was admitted that much more must be known about this form of organization [Lenat 82b].

Analogies in this context could be provided by every kind of link between concepts that is no isa–link. This of course holds only if we want to introduce a strict distinction between genl./spec. and analogies. But perhaps it is more natural to view genl./spec. as nothing else but a special case of analogy. Let me explain this.

A very broad informal definition would be : Learning by analogy is every kind of building 'senseful' [1] structures that is supported by already existing relations between already existing structures. (This view contrasts with more formal but also narrower approaches found for instance in [ECAI 86]. The disadvantage, of course, is that it is less clear how to realize such a view within an implementation). If you substitute the word 'relation' by the word 'isa–links' then you will get a nice informal definition of learning by generalization.

---

[1] More about the term 'senseful' in the chapter on evolution.

In order to avoid naming confusions and conflicts with definitions in the standard literature I would like to introduce the likewise very broad and informal notion of an 'informed structure' that comes close to my comprehension of the term analogy. A flexible learning system has to build informed structures (representations, methods modifying and making use of representations etc.) dependent on the hardware that is at its disposal and that provides the unchangeable framework for its development. An informed structure is any 'identifiable part'[2] of the system that supports the system in having success in its environment (which includes the construction of informed structures). The degree of 'informedness' a structure can have is given pragmatically by the contributions it gives to the success of the whole system. This view is strongly inspired by v.Weizsaeckers saying : 'Information is what generates information.' (See the chapter on evolution.)

Meta-capacity is considered to be important in large expert systems. Often heuristic meta-rules are introduced to select among a great amount of applicable rules proposed by such a system during a specific computation [Davis 80]. Meta-capacity probably is also essential for truly flexible learning systems. The more parts of a system are accessible by the sytem itself (in a non-destructive manner), the more senseful self-modification may take place. (EURISKO shows meta-capacity in a sense that is for instance constrained by its hierarchy.) Of course the self-accessibility should include the meta-rules, too.

A flexible approach to universal learning capabilities might be to define a system that (syntactically) allows the evolutionary creation of informed structures like the algorithmic methods mentioned above as well as the invention of new learning strategies that do not fall in any of the categories generalization, macro-building etc.. In the following two approaches to meta-learning will be specified. Since the history of the basic ideas reflects the reasons for the proceeding, I will stick to a more or less chronological order. [Knuth 74] advocates not only the presentation of the results but also the explanation of the faults that led to the formulation of the results (standing in a sharp contrast to the opinions of the giant mathematician C.F. Gauß). In Knuth's sense this paper describes the evolution of ideas dealing with evolution. (If you have read [Hofstadter 85] you might like it.)

The second chapter will present a first approach to meta-learning by making practical use of evolutionary principles. This approach will be criticized. Chapter 3 wants to gain a deeper understanding of the generality of the principles that cause evolution. The insights gained from these chapters will lead to a more natural second approach described in the fourth chapter. This second approach is accompanied by several implementations of the principles outlined. It will be seen, however, that purposive self-referential behaviour on the basis of these principles can not be expected, unless the machine power available is increased by some orders of magnitude. Thus the empirical results do not

[2] Identification presupposes an observer who is able to identify. The observer usually is a learning system, too. More about the problems involved with the term 'identifiable' in the chapter about evolution.

underpin the main thesis of this work, namely, that universal self-reference is the foundation of flexibility. Nevertheless the results indicate some interesting directions for future research.

Although the time sacrificed for the different concrete implementations exceeded the time needed for designing and writing this paper by an order of magnitude, the character of this work is inspirative rather than instructive. The important idea you should come to share is: The introduction of potential self-reference can be easy, if it is consequently supported from the beginning of the design of a self-developing system.

2. An Algorithm for Meta-Evolution.

2.1. Introduction.

This chapter proposes an algorithmic method to capture 'learning how to learn' based on a modified symbol-manipulative version of a genetic algorithm. To understand this approach a short review of the principles of genetic algorithms (GAs) follows.

2.1.1. Holland 's GAs.

Holland is considered as the father of GAs. He defined the finding of a solution for a problem posed in the context of a certain domain as a search. The search space has n dimensions and is put up by 2^n possible compositions of n relevant features that a solution can have or not [Holland 75].

In the beginning of the learning phase each member of a pool of randomly generated bitstrings of size n representing candidates for solutions is tested by a critic. The critic applies a domain dependent evaluation function and assigns a 'worthmeasure' (a real number) to each bitvector.

The next step is to select probabilistically one or two candidates from the pool, where the probability for the selection of a distinct bitvector is equal to its worth divided by the sum of the worths of all members of the pool.

If only one candidate has been selected, a mutation may occur. This means that a 1 may be changed to a 0 or vice versa somewhere in the bitstring, thus affecting the existence of some property of the candidate. Mutations should happen very rarely (see [Grefenstette 85] if you want to know why). The more interesting case occurs when two strings have been selected. Then a procedure called 'crossover' may take place which generates a new plan by exchanging parts of the genetic material of the two ancestors. This means that parts of the bit sequences of one string override the corresponding parts of the other one.

In any case the newly generated plan is tested by the critic which determines the new worthmeasure. If the latter is bigger than the worthmeasure of e.g. the worst candidate in the pool, this one may be replaced by the new one. (Alternative scenarios are thinkable, but common to all of them is some element of competition that in the long run leads to preference of the 'fit' plans.)

The cycle described in the last three paragraphs is repeated over and over again until some termination criterium is reached. This could be the appearance of a very highly rated candidate .

Some interesting properties of GAs have been proven that often make them the first choice if it has to come to a decision what kind of search method to apply to a given problem (see for instance [DeDong 75], and [Goldberg 85] for practical applications). In fact GAs have become so popular that the second international conference dealing with them is about to come up soon.

2.1.2. A Symbol–Manipulative GA.

In a practical course at TUM the author experimented with a modified version of Hollands GAs in order to explore ways to apply the principles explained above to the domain of automatic program synthesis [Dickmanns, Schmidhuber, Winklhofer 86]. The main difference to conventional GAs resided in representational issues.

Our candidates were (potentially) Turing equivalent programs represented as lists of arbitrary length. These lists contained statements written in a special language (a funny mixture of prolog and assembler) that allowed a controlled execution of programs being composed out of domain primitives. Control, of course, was delegated to a critic who attributed worthmeasures to the plans it tested, depending on their performance in the environment (domain)[3].

Some words about potential Turing equivalence. There is no criterion to decide whether a program written in a language that is 'mighty' enough will ever stop or not. So the only thing the critic can do is to break a program if it did not terminate within a given number of time–steps. Of course this is a restriction to Turing equivalence, but the degree of restriction can easily be modified (all there is to do is to change a variable). It is no more fundamental restriction than the one given by the finiteness of any storage device.

Because plans were not confined to have a fixed length, crossover was not so straightforward an operation as it is considered to be in the literature on the subject. Moreover the signs solidified that programming is a task that depends on better informed structures than the ones like crossover or the other genetic primitives are. Sometimes crossover proved to be useful by sensibly connecting sequences of actions gathered from two different plans. But often it was annoying to watch it for instance breaking a loop apart and leaving a plan that obviously made no sense. (The really annoying thing was to know that crossover never would change its silly behaviour by evolving to a more informed structure that avoids certain pitfalls. Why was that the case? The crossover algorithm was part of the 'hardware' (the unchangeable parts of a system) and so there was no possibility for other parts of the system to introspect and change it.)

Some additional primitives specific to our special form of GA were added, each of them improving the overall performance of the system a little bit. But soon new primitives reached their limits and led to a situation as unsatisfactory as the one depicted above.

The adhocness of the newly introduced primitives as well as the insight that really interesting domains like programming are unlikely to be treated successfully solely by some simple non–self–evolving methods led to the desire to enable the system to meta–learn the methods of how to learn. The notion of meta–meta–learning the methods how to meta–learn the methods etc. suggests itself, and a possible algorithmic framework for such a system with any number

[3] Something not dissimilar was done by [Cramer 85]. But here the programs were represented as bitstrings of a fixed small size (the conventional method) thus leaving no room for universality. The same is true for [Küchenhoff 86].

of meta-levels is shown next.

2.2. Meta-evolution.

Meta-evolution is a non-deterministic algorithmic scheme to develop algorithms making use of a few primitives that can be used to manipulate plans (programs). On the domain level we want to observe the development of plans that are useful in the domain. They are composed out of domain primitives and elements of the programming language chosen to structure and arrange the domain primitives in an algorithmic manner. These lowest-level plans can prove their adequacy by holding their own in the environment they are tested within.

On the level above the domain level the construction of plans is a new domain by itself. This means that operators like crossover (or more informed ones) are again represented as plans that can be atomized into elements of the programming language and plan manipulating primitives working on the lower level.

Because plan primitives are able to work on plans, and plan manipulating programs are represented as plans, there is no formal reason why the next level, the level of constructing plan manipulating plans, could not be tackled, too.

And so on.

Of course this proceeding requires plan primitives that are fundamental enough, so that compositions out of them can achieve any imaginable effect (Turing equivalence). One might think of simple plan editing primitives that are able to define some kind of 'current expression', to detect the 'end-of-line', to set markers, to compare elements of plans to be edited, to insert branchings dependent on such tests, to insert loops and pushs and pops (at least two stacks are necessary for well known reasons), to take two plans and insert parts of one plan into the other one or to delete certain parts. The primitives themselves should be trivial, however, their possible combinations building plans should be arbitrarily complex.

To start from scratch it is necessary that the hardware is able to generate syntactically correct plans for the initial phase. To create a first-order-plan by default means to intermix elements of the language and the set of domain primitives at haphazard or by any other default method but in a way that is constrained by the syntax of the language. To create an nth-order-plan where $n > 1$ is essentially the same with the exception that the set of plan primitives is included in that mixing process.

Here is the top level loop of **meta-evolution** written in a pseudo-algorithmic language that should be self-explanatory:

To do **meta-evolution** :

1. Set n = 1.

2. Forever do :

2.1. Call S(n) the set of nth-order-plans and set S(n) = {}.

2.2. While | S(n) | < maxpoolsize(n) do :
  2.2.1. Create a new nth-order-plan by default,
    give it a new name P.
  2.2.2. Set S(n) = S(n) ∪ {P}.
  2.2.3. **Test_and_criticize P.**

2.3. Set n = n+1.

As long as the pool of a certain level is not complete, it is enlarged. If a pool is filled, the pool corresponding to the level above is created. Pools of lower levels are changed by members of higher levels in a way that is hidden in the procedure **test_and_criticize** to which the main work is delegated.

To **test_and_criticize** a plan **P** out of S(**n**) :

1. If n=1
  then

    1.1. Transmit P to the domain critic who executes P in the environment and assigns a worthmeasure to it.

  else

    1.2. While no termination criterium is reached do :

      1.2.1. Select probabilistically some plans from S(n-1) and generate a new candidate P' by applying P to the selected plans.

      1.2.2. **Test_and_criticize P'**,
      treating it like a member out of S(**n-1**).

      1.2.3. update the current worthmeasure of P by using information about changes of performance gained by comparing the worthmeasure of P' and its ancestors.

2. Decide whether P displaces another member of S(n).

Test_and_criticize gains worthmeasures for the meta-plans it considers by applying itself recursively to the plans of lower levels generated by the meta-plans.

Termination of the while-loop may be caused by the observation that lower-level-plans did not improve for a long time.

The element of competition is introduced by the decision in step 2. which should of course favour highly rated plans. Competition takes place in every level below the highest meta-level, the members of the highest one do not (yet) have to participate in the struggle for live.

One should expect that in the long run more and more informed structures evolve in form of domain- and meta-plans. So default plans like e.g. a simple random crossover should be replaced by more methodical ones. One might imagine plans that represent information about how to build repeat-untils, perhaps by inserting conditions at 'plausible looking' places in endless loops of programs found on the level below. There is no limit to ones imagination if the set of initial primitives is chosen appropriately.

Probably the most interesting informed structures are those that aid to shorten the time to find new ones. Suppose the domain is the invention of plans that move a roboter through a large room. Suppose that the set of domain primitives includes simple actions like 'stepforth', 'stepright' etc. Then a good informed plan of the second level could be one that frames domain primitives with loops. This behaviour might often produce awkward results (imagine that the name of another domain primitive is 'grasp_object' !). But statistics might say that this principle is more promising in the environment than mixing primitives at random. This should result in a higher probability for survival for this informed structure.

Of course a plan from level 3 might profit by changing the level 2 plan mentioned above to an even more informed structure. This could be realized by inserting pieces of code that restrict the number of domain primitives the lower plan works upon.

Nobody said that random crossover is no informed structure. It is ! It somehow represents the very general heuristic saying that the world often is continuous and that it makes sense to create new information by somehow connecting information gathered from available structures[4].

Since the world is not only continuous but also manifold, general heuristics need refinement. (I will not state that they are to be specialized because specialization is only one part of the story as the shallow tree problem shows). The need for a universal refinement scheme represents a good deal of the justification of **meta-evolution**.

Reward is running bottom up. Effects in the domain may indirectly have an influence on the ratings of high-order plans, but the critic can not have a look into the whole system. All it can watch and recompense is the domain level.

Because of the cascade-recursive behaviour of **test_and_criticize** we can not expect the rapid creation of very high levels if the domain critic needs a noticeable time to do its work. Learning *is* a process that takes time.

---

[4] Crossover looks a little bit like the root of analogy !

Meta-evolution seems to be a way to learn the domain of learning. One can easily imagine evolved structures that practice genl./spec. or others that cannibalize certain plans in order to do learning by building macros. The good news is that there is no pledge to use a particular one of these popular methods. Anything could potentially evolve, depending only on the set of initial primitives. The bad news is that there are still several reasons for not considering **meta-evolution** as the best way to achieve our goal .

2.3. Critique of Meta-evolution.

The criticism presented here is not concerned with the question of the potential ability to learn. Church's thesis says that Turing equivalence catches everything that can be catched. It is the question of naturality that remains. (Computability versus Feasibility.)

How natural is the creation of meta-levels, meta-meta-levels and so on? At least human beings do not learn like this, instead we permanently mix levels. We do not use a counter that says: Now you are at level seventeen[5].

How natural is the general representation of knowledge in the meta-hierarchy ? Every type of knowledge (static or dynamic) must be represented somewhere in a plan. Access to knowledge ensues from actions executed by meta-plans. Often this kind of access might look unnatural, less like a straightforward inspection of a variable but more like a strange kind of search for some part of a program that some meta-plan interpretes as something. This is connected with the next question :

How natural is it to basically employ a genetic algorithm at all levels ? Isn't this too much bias ? Symbol-manipulative GAs seem to leave more room for development than e.g. strict genl./spec. frameworks. But should it not be possible that the GA some day may be abolished and replaced by some other kind of scheme, a scheme that makes better use of the capabilities of the physical machine it is running on ?

There are more arguments against the algorithmic scheme **meta-evolution**, arguments delivered by the field that provided the inspiration for GAs : The field of molecular biology. Although we can't exclude the possibility that some sort of higher-order evolution took and takes place on the molecular level (in fact the contrary is rather doubtful), the development of biological organisms itself indicates that information processing on this level has reached its limits. The evidence for this claim is given by the fact that biological GAs discovered a faster way of doing evolution: A possibility to evocuate the main part of information processing to the phenotype by providing it with some (probably rather unstructured) hardware (baby's brains) and some (probably sophisticated) software. The clue is that the software is rather a germ suited to acquire more software than a set of fixed programs[6].

---

[5] The number 17 is an hommage to Prof. Güntzer.

[6] And it is obvious why this is necessary : The DNA strings are just too short to carry all the information necessary for, say, an adult primate to survive. The number of neurons within a human brains is about 10^{10} [Schulten ?], and what seems to be of even greater importance, the number of connections between them is again larger by a factor of 1000. (To

The reflections in this section will lead to a second more natural approach to meta-learning. Evolutionary principles still will play the fundamental role, but appear in a garment that does not look similar to the principles of GAs any more. What is the essence of evolution ? The next section is intended to show that the notions behind evolution are much more general than the purely darwinistic conceptions influenced solely by ideas from biology.

---

speak with Schulten : It is the connections that carry the information.) But the maximal amount of syntactic info a DNA can carry is about 10^{10} bits, and most of them are used for things that do not have much to do with the brains. There are 10^{16}(?) other cells in a human body.

3. Evolution and Learning.

3.1. What is Evolution ?

> "Als Evolution bezeichnet man vorzugsweise die Herausbildung der Gestaltenfülle des organischen Lebens im Laufe der Erdgeschichte. Die Herausbildung einer Fülle von Gestalten ist freilich nicht auf den Gegenstandsbereich der Biologie beschränkt. Einerseits gibt es eine reiche spontane Gestaltenbildung im Anorganischen; heute unter den allgemeinen Kategorien der Synergetik mitumfaßt. Andererseits schafft auch die menschliche Kultur immer neue Gestalten. Evolution als Vorgang umfaßt also die ganze Wirklichkeit, die wir kennen. Sie bedarf also auch einer umfassenden Erklärung."

This quotation is an excerpt from [Weizsäcker 85].In the following Weizsäcker argues that the growth of entropy is identical to the growth of *Gestaltenfülle* (the plentyness of forms) if certain premises hold. With a little mathematical experiment he shows that the introduction of simple binding forces into a model (condensation model) similar to the kinetic gas model promotes the growth of *Gestaltenfülle*. (The kinetic gas model led to the formulation of the second law of thermodynamics). In a world like ours a row of binding forces exists. This makes it probable that the often cited 'warmth-death' of the universe does not result in an uniform distribution of atomar particles, as one might conclude naively by extrapolating the gas model. On the contrary the final state might rather resemble a "collection of complicated skeletons " [Weizsäcker 85].

So the phenomenon of evolution does not contradict the growth of entropy, as it is assumed. A frequently cited argument says that a decrease of entropy in one part of the world has to be compensated by an increase somewhere else. Without denying the existence of processes of this kind v.Weizsäcker says that the development of forms *does* result in an increase of entropy. He argues that the many ungood feelings relating thereto have their roots in a verbal or notional negligence. He shows that the definition of syntactic information $H = -\sum_k p(k)\ ldp(k)$ (where k disjoint events E(k) may occur with probability p(k)) is in substance the same as the one given for entropy, including the sign:

> "Man hat Information mit Wissen , Entropie mit Nichtwissen korreliert und folglich die Information als Negentropie bezeichnet. Dies ist aber eine begriffliche oder verbale Unklarheit. Shannons H ist auch dem Vorzeichen nach gleich der Entropie. H ist der Erwartungswert des Neuigkeitsgehalts eines noch nicht geschehenen Ereignisses, also ein Maß dessen was ich wissen könnte, aber zur Zeit nicht weiß. H ist ein Maß potentiellen Wissens und somit eine definierte Art von Nichtwissen. Genau dies gilt auch von der thermodynamischen Entropie. Sie ist ein Maß der Anzahl der Mikrozustände im Makrozustand. Sie mißt also, wieviel derjenige, der den Makrozustand kennt, noch wissen könnte, wenn er auch den Mikrozustand kennenlernte."

In the following v.Weizsäcker distinguishes between potential and actual Information. He regards actual information as negative entropy or as the information about a micro state that one possesses only by knowing the macro state. Potential information is what could be gained by knowing the micro state.

Entropy is potential information. Whether entropy is a measure of *Gestaltenfülle* or of disorder is only a differentiation between degrees of knowledge.

Evolution is sometimes viewed as the principle that generates order out of chaos. But according to v.Weizsäcker and to common sense order is something subjective. Consider figure 1 that shows a table of 9x9 partly colored fields. The colored fields are scattered chaotically as long as you do not know that every pth field is black (counting them by rows) iff p is a prime number.

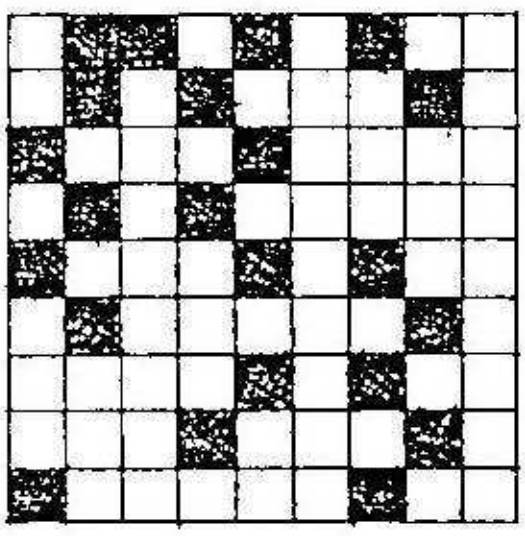


Abb. 1

Order depends on knowledge.

That is,order depends on knowledge. Following [Weizsäcker 85] we may conclude that the statement "disorder gets larger and larger" is a wrong conclusion out of the 2. theorem of thermodynamics. Entropy grows, but that does not mean that disorder grows.

In fact the world is becoming more and more ordered in the eyes of a learning observer, because he by himself provides the subjective scale of order. Simply because he is acquiring more and more knowledge about the world, the order of the world increases.

Weizsäcker argues that operational definitions of information and usefulness can be given, making both essentially identical. He pleads mathematically for the view to see information as a real function of usefulness for subjective probabilities. He describes evolution as the growing of potential syntactic information and shows that it is the most likely phenomenon.

3.2. Pragmatic Information.

The conventional every-day notion of information does not refer to the syntactically defined form of a message, but to "what is understood" ([Weizsäcker85], Thesis1). [Küppers 86] explains that " the objectivation of the semantical aspect of information is possible only if we include the pragmatic component of information ". This leads to the formulation of v.Weizsäckers Thesis 2: Information is only what generates information. This thesis is meant as a tightening up of the statement that pragmatic information is only what works, i.e. what is effective [Weizsäcker 85]. It is no circular definition.

[Weizsäcker E+C 72] introduce two variables that help to define pragmatic information : *Erstmaligkeit* (first occurence) and *Bestätigung* (confirmation). Useful information is only possible if some things are happening that are familiar to the information processing system (*Bestätigung*). But of equal importance is the appearance of unexpected events (*Erstmaligkeit*).

> "Nahe dem Grenzfall hundertprozentiger Bestätigung kann jede Neuigkeit registriert werden. [..] {die Verfasser} schlagen vor, in diesem Grenzfall die Erstmaligkeit direkt durch die Information im Sinne Shannons zu messen. [..] Nimmt aber der Bruchteil der Bestätigung ab, so kann nicht mehr jede Neuigkeit pragmatisch effektiv registriert werden."

If there is no *Bestätigung*, there is no useful information either.

> "Bloße Bestätigung entspricht der Karikatur des Spezialisten: er weiß alles über nichts; bloße Erstmaligkeit entspricht der Karikatur des Generalisten: er weiß nichts über alles."

(Quotations from [Weizsäcker 85]).

3.3. A Link to Piaget.

Erstmaligkeit and *Bestätigung* are connected with two notions introduced by Piaget: Accomodation and assimilation. (A good introduction to Piaget is given in [Ginsburg, Opper 75]). A learning child performs assimilation by giving existing schemes the chance to apply themselves to the environment. If the child already disposes of a structure that represents internally the falling of a ball on the ground it may apply that structure to other objects that are no balls, e.g. eggs. Assimilation describes the tendency of available patterns to apply themselves to the world (expectation driven programs assimilate, too). Assimilation relates to *Bestätigung*. Assimilation is justified because the world is not a random world but structured in a way that often allows *Bestätigung*.

On the other side the phenomenon of accommodation describes the forced creation of new structures within a learning system. The child may apply its scheme for 'falling objects' to a bird. But this scheme is not compatible with the real-world-event, because the bird is flying away. Now the child may accommodate structures that give room for objects that do not fall. It may, but it often won't. This depends on how important the accommodation of new structures is for the 'success' of the child, which again depends on the pragmatic context. *Erstmaligkeit* is related to accommodation in an obvious way.

We can regard assimilation as an oppression of the world by the brains, and accommodation as an oppression of the mind by the world. The alternate play of assimilation and accommodation is called equilibration [Piaget...]. The principles of equilibration can be found not only in the development of children but everywhere where expectation driven evolution takes place. Before we shall localize assimilation and accommodation in **meta-evolution** we want to gain a little more understanding of the nature of learning systems.

3.4. What is a Learning System?

To clear up this question it should be helpful to know what the term 'system' stands for. But the notion of a 'system' is something subjective and vague. [Weiss 77] tried to give an operational definition: For all material sub-comlexes s_i (i=1..n) of a complex S consider the cumulative balances v_i of fluctuations of physical and chemical parameters of s_i around some mean value. Let V be the variance of all identifiable properties of S. Then S is a system if $V \ll \sum_i v_i$.

> "Das wesentliche Merkmal eines Systems wird hier durch ein Stabilitätskriterium beschrieben, das die grundsätzliche Invarianz eines Systems gegenüber den Schwankungen in seinen Subsystemen zum Ausdruck bringt." [Küppers 86]

There remains a lot of room for subjective arbitrariness, most strikingly located in the sign '<<' and the word 'identifiable'. If there are difficulties with the term 'system', then how much more are there with the term 'learning system' ? Due to these problems the following discussion will have only informal character.

Intuitively we would say that a learning system is a system that (partly) consists of evolving sub-systems reflecting the outer world in a way that assists the entire system to survive. Within the system some sort of mini-evolution must take place in order to build informed structures that help the whole to hold its own in the world. Of course the identification of sub-systems is equally dependent on subjectiveness as the identification of systems in general.

Provisionally let us view a learning system as a set of informed structures being separated from the rest of the world it is existing in. The world should be interesting, which means it should allow 'arbitrarily complex' structures. The learning system should be connected to the world by effectors and receptors, because a system without connections can neither observe nor be observed. Effectors are structures belonging to the system that have an influence on structures belonging to the rest of the world. Receptors are structures belonging to the system that have an influence on other system structures dependent on structures belonging to the rest of the world. The notion of a 'structure' indulges in subjectivity - one could say 'sub-system' instead of 'structure'.

What does it mean to identify a learning system? Someone must be there who does the act of identification. This one will be called the observer. Difficulties arise with the fact that at least in our world the observers are evolving learning systems, too. Let's imagine that a learning observer who observes his

environment may try to discover other learning systems[7].

Because learning seems to enforce the modification of subsystems (informed structures) the observer will have even greater problems to discover structures that are permanent and can be regarded as the essence of some learning system. This is because only patterns that are familiar to the observer will be regarded. In other words: The observer who himself is a learning system can only identify something if the *Bestätigung* he gains by observing that something is above zero. Moreover it has to be so clearly above zero that he can do either assimilation or that he is at least motivated to do accommodation, in order to learn more about the complex in focus and which may turn out to be a learning system. The subjectiveness of the observer is determined by the knowledge he accommodated during his own development.

Obviously it is not easy for an observer to decide what is a closed learning system in our general sense and what is not. If the world is too complicated it is impossible. The point is that in an interesting world like ours systems making use of evolutionary principles are products of evolution again, as well as the observer himself. But the pragmatic contexts relevant to the different systems are different, in fact they might be too different. An observer watching his own evolving environment might not discover certain evolving learning systems (if the world is complicated enough and the observer is simple enough, which might be applicable in the case of the universe and human beings as observers). If the observer's knowledge about the world is limited, the world is only partially ordered in his eyes whilst other parts look chaotical. How can he recognize informed structures if he does not see any structures at all ? If he does not have any access to the pragmatic context that is relevant to these structures? He often will not be able to identify a set of structures and find the line that separates it from the rest of the world.

Surprising examples for structures holding their own in a certain environment come from chaos theory (see e.g. [Crutchfield, Farmer, Packard, Shaw87]). It took a long time for the learning system mankind to discover many of these structures. By having discovered them the order of the world increased (the observers are learning systems, too). But how much more can be found ? It seems that our informal definition given above often does not make too much sense.

The conclusion from this at all is nearly a platitude: Learning systems are learning systems only if they are considered as such by other learning systems. This requires enough 'pragmatic intersection' between observer and the system being observed, otherwise there simply is no other system in the eyes of the observer. In turn, he by himself probably will not be identified as such a system by 'the other side'.

---

[7] An example for a learning system in our broader sense (identified by biologists) is given by the set of all genotypes of a particular race. Informed structures are for instance the DNA strings themselves. The phenotypes are the effectors!

Of course we have rather clear notions about what a learning system implemented on a computer should learn, because by defining the world we provide the pragmatics. The pragmatical aspect of the world presented to a learning program should have a big intersection with the pragmatics of our world. Otherwise, if for instance some evaluation function of the critic in **meta-evolution** is not chosen appropriately, the system could escape into a direction we can not follow, and we will say the system failed to learn the task we posed. We will not consider it as a learning system, although it just followed the rules of evolution. But in fact we failed by not supplying the correct 'pragmatic pressure'.

Note that it is less clear how the system should learn its task, at least if the initial germ is flexible enough to be 'interesting'. Learning systems are identified being of such a kind only because they evolve in a way that is familiar to the observer. I consider my brother to be a learning system. But I am far away from knowing what is happening inside.

The glance into a learning system might be as meaningless as the observation of chaos (see the literature about neuronal nets). This somehow contrasts with the desire to understand everything that is happening within such a system. But the inability of learning systems to understand completely other equally complex ones might be fundamental.

In the theory of cellular automata (cellular automata can accomplish for instance certain pattern recognition tasks) many systems are known where certain initial states under certain conditions evolve chaotically. 'Chaotically' in this sense means that the shortest algorithm to compute the final state takes about as much expense as the complete simulation of the automaton. In other words, the automaton *represents* the shortest algorithm to compute the subsequent states.

It remains an open question whether these nearly unpredictable processes play a fundamental role in learning. But intuitively it seems to be plausible that 'interesting' learning systems some day will escape some initial (well understood) schemes that are provided by other learning systems as for instance human beings.

AM gives an example for escaping some initial algorithms, namely the name giving procedures. These were thought to construct new names for newly composed concepts out of the names of the elements. Soon these names were getting stranger and stranger [Lenat 82]. This may be a symptom of a more general law: Any (interesting) learning system will get in conflict with those initial algorithms that do not evolve. Of course the name giving must not be changed by the system, because it represents a part of the semantic interface to the human observers. On the other hand it should be changed because otherwise the observers also loose their ability to observe as they do not understand the new names any longer. In the case of AM the problem is solved by humans who interprete certain new concepts as something that is familiar to them (like primes) and who supply a related name.

But if a learning system gets significantly larger than AM there will be no chance for humans to find all 'good' concepts, simply because there will be too

many of them. Humans will not have enough time to look at every concept and decide whether it is related to something they know. Moreover certain concepts with a strange name may help the system in a way that has not been foreseen. A truly interesting system will find so many unforeseen ways that an observer will not be able to follow.

This has to do with another problem pointed out by Winston: The reasons for AM considering a concept as being of interest are often strikingly different from the reasons that cause a human to like the same concept. Why is that? In fact the concepts are not the same, but the names are.

3.5. Symbiosis Versus Parasitism.

A parasite usually is viewed as a structure that survives because it makes profit from the existence of other structures without contributing to their survival (a soon disappearing parasite would be one that destroys the structures it depends on).

Symbiosis takes place if two ore more structures complete each other and gain mutual advantages.

But the distinction between symbiosis and parasitism is as hard as the discovery of learning systems, for analogue reasons. Furthermore if something was identified as a parasitizing structure it may turn out to be part of a symbiosis if it is viewed from a different angle. The TBC germ was a parasite of the human race. But didn't it trigger important medicinal developments that helped mankind more than the TBC bacillus did damage to? From this point of view it was part of a symbiotic system, at least as long as it was not extincted by the development its appearance stimulated. The frontier between symbiosis and parasitism is an indistinct one and depends on subjective knowledge.

Symbiosis plays an important role in the evolution of complexity. Structures appear that are decomposable into many smaller structures that can not survive alone. A system that owes its existence to symbiotic principles is the human body with its masses of specialized cells. A larger one is the human society with its masses of specialized humans.

3.6. Erstmaligkeit, Bestätigung, Symbiosis and Meta–evolution.

To be able to handle *Erstmaligkeit* a learning system has to dispose of a principle to build structures that reflects the possible appearance of unexpected things in the real world. In GAs this principle is given by the possibilities of mutations, of random changes of already existing structures.

Pragmatic *Bestätigung* is introduced by the critic which establishes the connection to the world. So e.g. **meta–evolution** is the basis for an expectation driven system able to do accommodation if necessary.

Information generates information. The abstract quality called information is carried by something that I called informed structures in the introduction. So informed structures should generate informed structures. In **meta–evolution** the physical realizations of plans on a machine are examples for informed structures. Their degree of informedness is defined pragmatically by the advantages

they cause on the lowest level, the physical environment[8]. In fact the informedness of a plan is indicated by its probability to survive. Each plan's existence is justified only to the degree in which it is contributing to the success of the whole system in the 'real' world.

Remember that the motivation for the algorithm called **meta-evolution** came from biology. Biological evolution is a kind of evolution that was successful enough to become well-established. But it is only one manifestation of the general principle that may be expressed in its simplest form: Everything that survives, exists (if it is realized!). This principle does not only apply to physical individuals but also to ways of doing evolution as well as to other abstract ideas: The abstractum 'death', for instance, survived because the way of doing evolution by letting individuals multiply and die survived.

At the end of the last chapter it was mentioned that biological evolution led to a more 'advanced'[9] kind of evolution. By providing the phenotypes with a framework to do some sort of mini-evolution (learning) the genotypes found a way that put their own role into the background. Although the invention of sexuality (the first exchange of information located on DNA strings) led to dramatic evolutionary leaps [Eigen 86] some day this kind of evolution was relieved. DNA strings came into existence that supported an external information processing (external relative to the DNA) thus allowing more promising directions for development. Up to now this culminated in the evolution of human societies. The inventions of language, printing of books or computer nets are symptoms of this ongoing development that unties (in our subjective eyes) the main part of information processing from molecular structures ([Markl 86] : "Language is the sex of culture").

Let's return to our goal to make machines learn. Most computer scientists today depend on a von Neumann machine as the basis for their learning programs (so do I). Consequently the basic software germ whose task will be to acquire more software should be designed to allow structures that make use of what a von Neumann machine can do well (setting pointers, interpreting sequential programs ...). The development of 'natural' structures should be supported. Naturality is prescribed by the kind of data structures and algorithms a v.N. machine can process efficiently. Due to the reasons mentioned at the end of the last chapter **meta-evolution** does not seem to meet these criteria very well. Another argument against this algorithm is given here: **Meta-evolution** does not leave much room for symbiosis. Symbiosis and parasitism surely might be identified in 'chains' of plans from different levels. But an important idea behind the unchangeable algorithmic scheme is the parallel holding of information in a manner that supports competition but not so much the collaboration of plans. Although competition plays a fundamental role in symbiotic systems (in order to favour certain kinds of symbiosis), a system implemented on a computer

---

[8] Of course it does not matter if the 'world' again is simulated on some (probably the same) machine. Simulation pragmatically changes into reality if all aspects relevant for the critic are contained.

[9] The word 'advanced' means something very subjective.

should support the specialization of 'plans' and not try to make every plan a universal genius (as **meta-evolution** prefers to do by throwing less rated plans away although they might be very useful in certain situations).

In the next section another approach to self-referential learning is presented that seems to be better suited. It is based on ideas leaving more room for symbiosis[10].

4. Self-referential Associating Learning Mechanisms.

4.1. Introduction

In this chapter I want to propose another approach to meta-learning, where more care is given to principles of symbiosis and their realizations on v.Neumann machines. Originally this approach was inspired by another idea of Holland[11]: The bucket brigade (b.b.)[Holland 85][Holland 86]. B.b.s seem to be a way to handle a problem mentioned in [Minsky 81]: "The Basic Credit Assignment Problem for Complex Reinforcement Learning Systems".

4.1.1. Classifier Systems and the Bucket Brigade.

On a global message list messages in form of bitstrings of size n can be placed either by the environment or by entities called classifiers. Each classifier consists out of a condition part and an action part defining a message it might send to the message list. Both parts are strings out of $\{0,1,\_\}^n$ where the '_' serves as a 'don't care' if it appears in the condition part and as a 'pass-through' if it appears in the action part. A real number is associated with every classifier indicating its 'strength'.

During one cycle all the messages on the message list are compared to all classifiers of the system. Each matching classifier computes a bid by multiplying its specifity (the number of non-don't_cares in its condition part) with the product of its strength and a small factor. The highest bidding classifiers may place their message on the next list, but they have to pay with their bid which is distributed among the classifiers active during the last time step which set up the triggering conditions (this explains the name bucket brigade).

Certain messages result in an action within the environment (like moving a roboter one step). Because some of these actions may be criticized as 'useful' by an extern critic who can give payoff by increasing the strengths of the currently active classifiers, learning behaviour may take place.

Classifier systems seem to be simple, but they are potentially mighty. E.g. [Forest 85] shows that classifiers are well suited to implement semantic network structures (although Forest does not refer to the learning of such structures).

4.1.2. Symbiosis and the B.B.

In contrast to GAs the bucket brigades are subgoal-reward schemes [Westerdale 85]: in the long run only those classifiers become stronger that are 'setting the stage' for actions that lead to payoff. These classifiers have higher chances to assert themselves during the bidding phases, and sequences of useful actions

---

[11] Probably many schemes similar to b.b.s have already been in practical use before Hol-lands analysis, but they were not outlined explicitly. During a lecture on connectionist models held by Scott Fahlman in Munich in summer 86 he reported the advantages of the 'back-propagation method' developed by one of his students. I asked Fahlman whether back-propagation isn't essentially a b.b. scheme. After some hesitation he agreed.

triggered by messages from the environment evolve.

Only the teamwork of many little entities produces structures that survive. Although the competitive element is not abandoned (it is indispenseable), structures with symbiotic character are supported. Usually there are no classifiers that could survive alone.

4.1.3. Meta-capacity for the B.B.

After some time it is necessary to create new classifiers (if the environment is fastidious and not trivial). It is no wonder that Holland in his capacity as pope of GAs employs a genetical algorithm to solve this problem. Strong classifiers are preferably engaged in the process of exchanging 'genetic material' thus creating new ones to be tested.

The augmentation of the b.b. by a GA can be viewed as a simple form of meta-learning. But there is only one additional level above the basic level. There is no further means to augment the genetical algorithm. The system described above is not 'closed on itself', at least not explicitly.

Wouldn't it be more natural to apply the principles of the bucket brigade to the augmenting of itself? A possibility to do this would be the introduction of 'mental' primitives (an analogue to plan-primitves in **meta-evolution**) that can be used to analyze or to create classifiers. There should be no essential difference between the triggering of domain primitives and the triggering of mental ones. Both types should be able to appear as part of the evolving sequences of actions.

In the long run a b.b. scheme that is closed on itself in such a manner should develop 'good' heuristics (in form of appropriate classifier sequences) to create new classifiers. 'Good', of course, again is defined pragmatically by the environment.

The artificiality of the many levels in **meta-evolution** disappears, because all meta-levels collapse into a single one.[12]

Again the main argument for such an approach is the fundamental increase of flexibility. A system making use of such self-referential principles does not depend forever on certain initial algorithms. On the contrary, it should adapt itself to the increasing demands of an interesting (non-trivial) environment as well as to the increasing demands of its own internal representations. To avoid signs of stagnation such a system must be able to refine its methods if its representations are refined.

Experiences with stagnation have been made with AM, the predecessor of EURISKO. More than fifty initial heuristics referring to mathematics (and to mutations of lisp expressions representing mathematical functions) caused AM to 'discover' and to name new concepts from maths. Starting with a few concepts from set theory it created many more related to numbers, multiplication, primes, and conjectures like the famous one by Goldbach.

---

[12] This is probably closer to the way human beings handle meta - knowledge.

After some time AM's hit rate sank, i.e. the number of concepts considered as senseful by human observers decreased (rapidly). One reason for this was localized in the fact that the initial heuristics applied well to the initial concepts from set theory (and their representations), but not to more advanced mathematical concepts created by AM. These observations led to the desire to include the field of heuretics into the learning process, and EURISKO [Lenat 83] was born.

EURISKO and CYRANO (a 'thoughtful reimplementation of EURISKO' by Kenneth W.Haase jr. [Haase jr. 86]) are the only systems I know that somehow are explicitly closed on themselves. EURISKO is potentially able to introspect and modify all of its parts, because it is written in a language that makes everything explicit [Greiner 80]. Even the lisp interpreter building the basis for the system is represented explicitly.

But there is a great difference between introspection and potential introspective abilities [Maes 86].Certain parts of EURISKO are accessible in a natural way, for instance the concepts that represent the current domain, and the heuristics that work on domain concepts. But e.g. the fundamental modification of the organization of heuristics would almost certainly result in a failure of the whole system. This is because EURISKO's successes are largely dependent on its generalization hierarchy. If something happened that damages this hierarchy it would be extremely unlikely that an equally or better suited form of organization evolves at the same moment. The deeper reason for this is that big parts of EURISKO as for instance the truth maintenance system depend on the hierarchy, and that these parts are not very well described in the eyes of the system. It is not explicit which kind of changes to these parts are harmless or perhaps catastrophic. The system may want to find it out of its own accord by using its learning capabilities, but it may be the last thing it finds out. Changing fundamental parts of its behaviour may be the last self-modification the system executes. An example: lisp programming was included as a domain to be explored by EURISKO. When starting to modify its own lisp code EURISKO soon ran into bugs [Lenat 82a].

There seems to be no obvious way how EURISKO could reconfigure itself into a system that represents its knowledge in an e.g. more analogical form. But the shallow tree problem mentioned in the introduction indicates that most relations between objects do have a more analogue nature, and that the isa-link is only one of many important links.

So called low-level methods as GAs, b.b.s and neuronal nets seem to be better suited to make use of the giant field of analogous connections. An initially simple but potentially mighty system should allow the creation of generalization trees as well as the development of the many other methods to organize knowledge. Bucket brigades are powerful enough to let default hierarchies emerge [Holland 80]. Are they simple enough to allow anything else one might imagine? And in a 'natural' way ? If the answers are yes, then closing the b.b. on itself should be an exciting experiment. But at least on a v.N. machine the answer seems to be no: the massively parallel matching of messages and

condition parts simply takes too much time. (Holland makes use of a special hardware, [Holland 86]).

To build a learning germ for a v.N. machine we will follow a different approach: SALMs.

4.2. SALMs, PSALMs.

The word SALM is an acronym for 'self-referential associating learning mechanism'. This term stands for a domain independent mechanism that provides a simple but broad framework for the further development of a software germ defining the initial state of a learning system.

Why associating? Because the basic action of conventional association is setting or following a pointer, which is easy for v.N. machines. (This is the reason why many languages essentially doing pointer manipulation are implemented on such machines). Association will provide the basis for storing any kind of information as well as for the execution of sequential programs. In both cases the machine has to do the things it has been constructed for: following addresses in the storage.

Why self-referential? After all that has been said before the reasons should be clear.

Some prototypical SALMs (called PSALMs, of course) have been implemented.

4.3. What all PSALMs Have in Common.

Pressure to learn some behaviour in some domain is supplied by the way the hardware (the unchangeable part of a certain PSALM) interpretes the entities that collectively make up the system. Under certain conditions certain entities may trigger certain actions in an arbitrary environment. Furthermore every interpreted entity can have an influence on the decision which entities are to be interpreted next. This is meant to allow some flow of control driven by the system.

A critic of executed actions is getting active from time to time and may give payoff to the system. The pragmatics of the world (partly) are given by the evaluation functions the critic uses to determine the amount of payoff it spends.

Payoff may be used by the system to extend itself by creating new entities or associating old ones in some way or just to strengthen certain parts of itself.

All entities that are interpreted by the hardware have to pay with a part of their strength for that privilege. On the other side the hardware prefers to consider strong entities. So in the long run the hardware tends to support sequences of actions that lead to success in the world by decreasing every 'senseless' entity's probability to be interpreted. One could draw an analogy to the metabolism of biological individuals: Being active requires resources, but resources are limited. (—› Competition!)

A primitive is an action that can not be decomposed by the system. Entities can be associated with actions that may be domain primitives or 'mental' primitives. Those that are associated with domain primitives represent the outgoing connection to the world (effectors). In turn the system can receive messages from the environment (perceptions) which are also represented as entities.

The critic can notice only actions performed within the domain. The system is free to use payoff as it likes, but it can not create payoff (otherwise it could

escape the pragmatic pressure). This does not imply that it cannot transport payoff to entities it considers to be adequate.

The transport of payoff, the creation of new entities and the building of associations can be done by entities associated with some mental primitives. Sequences or clusters of mental primitives also can (must !) work on themselves to organize the way they organize the domain knowledge.

4.4. PSALM 1.

The experiences with PSALM 1 led to a stricter distinction between the language a PSALM uses and the pressure that forces the language to organize itself. It may be helpful to follow this evolutionary process (the author was part of it) chronologically.

PSALM 1 was a straightforward implementation of a self-modifying associative net based on weighted links. The basic structure was called an entity. At time t each entity e could be associated with an action (an undecomposeable functional representing a domain primitive or a mental primitive) and a set assocs(e,t) of ordered pairs out of SE(t)x[0;1], where SE(t) denotes the set of all entities existing at time step t. Each pair represents an association between e and some other entity, as well as the 'strength' of that association. This linkstrength between two entities e and e' at time t is defined by

$$s(e,e',t) := x \text{ if } (e';x) \in assocs(e,t), \text{ and } 0 \text{ else.}$$

Let the agenda A(t) be the set of all entities interpreted in parallel by the hardware at time step t. Then it is possible to define an order 'moreinteresting(t)' on SE(t):

$$e1\ moreinteresting(t)\ e2 <> \sum_{e \in A(t)} s(e,e1,t) > \sum_{e \in A(t)} s(e,e2,t).$$

The agenda A(t) can be sorted by the law of order defined by itself. Taking one entity from or adding one entity to the agenda may result in a completely different order of the elements it is consisting of.

At time 0 the payoff value is initialized with zero. At time t the hardware interpretes all (or the most interesting) entities of A(t) by inspecting whether they are associated with some action that is executed if there is the need. Because some domain dependent entities may cause the extern critic to give reward, payoff in form of a real number might be added to the old payoff value.

At each time step t>0 all links from A(t-1) to entities $\in$ A(t) are punished or rewarded (proportional to their old values):
$e1 \in A(t)$ *implies* for *all* $e \in A(t-1)$:

$$s(e,e1,t) := c*s(e,e1,t-1) + P*\frac{s(e,e1,t-1)}{\sum_{e' \in A(t-1)} \sum_{e'' \in A(t)} s(e',e'',t-1)}.$$

P denotes a number that is gained by decreasing the payoff by some default method, c is a constant out of [0;1] close to 1. A(t+1) becomes the set of the most interesting entities defined by A(t) unified with the set of perceptions that may have appeared during time step t. If there are no interesting entities (with a strength greater than a given threshold), some random selection takes place.

To explicitly close the system on itself on a very low level some mental primitives like the following ones were introduced:

- create_entity: This one creates and initializes a new entity and associates it with the whole agenda at the same time.

- create_links: creates or strengthens links between the most interesting entity and the rest of the agenda (if there is enough payoff to do so).

- shift_payoff: Takes payoff if available and distributes it on the links leading from the agenda to the 'outside'.

The goal of PSALM 1 was to to avoid the massive parallelism of bucket brigade schemes hoping that the system will develop strategies composed out of primitives that always place the 'right' entities in the agenda. It was implemented in INTERLISP and consisted of about 60 k of code most of which were concerned with gaining some efficiency by using hashing techniques which are not supported by INTERLISP [INTERLISP 85].

PSALM 1 was a flop. During the (very limited) times of observation no kind of structure evolved that would be worth mentioning. Although a rate of 5 agenda cycles per second is clearly too slow this is not the only reason for PSALM 1's failure.

4.5. Lessons Learned from PSALM 1.

There seemed to be nothing wrong with the basic hardware parts of the system. But nearly all of the mental primitives implicitely represented some unstated heuristics like : "If a new entity is created then it should make sense to associate some other entities with the new one, otherwise the new one will be lost soon and become garbage." Of course this heuristic may be helpful sometimes, but often it will cause trouble among active entities. The point is that the system has to build unnatural constructs to escape such trouble makers without loosing their advantages. But it should discover special procedures for special situations instead of general ones.

> Lesson 1. Make your primitives as primitive as possible, do not overload them with heuristics that may help a little bit in the starting phase but may be the reasons for awkward and artificial constructs in the many situations you have not foreseen.

Related to this is the 'parameter problem': If we want to introduce self-referential primitives that are able to associate other entities it must be clear *which* entities are taken as arguments. In PSALM 1 this was handled by using default arguments like 'the most interesting entity within the agenda' etc. In order to occasion the hardware to consider some distinct entity as a parameter for the action create_links, this entity had to be marked as the most interesting one by references from other parts of the agenda ..., what again resulted in very artificial constructs.

> Lesson 2. Become aware of the problems involved with parameter handling and provide the initial system with the potential to solve it naturally.

The parameter problem seems to be solveable by the introduction of more mental primitives that set global variables, which by default serve as arguments for the currently active entities. Two global stacks seem to be enough for such an 'extern' parameter handling. But global variables like stacks have another disadvantage, namely that they are global. This means that a program depending on such stacks has to be very careful. Misinterpreting one element popped

from top of a stack may cause the failure of the whole program (think about the consequences if an interpreter who evaluates a recursive function did not interpret some value on the stack correctly as the old base-pointer but as the value of the last incarnation).

But learning systems often 'misinterpret' (remember the fundamental processes of assimilation and accommodation). Although potential Turing equivalence is achieveable with two stacks and little more, the dangerous and unstructured programs running on such simple devices are far away from being natural (and from being similar to our way of thinking). A possibility to smoothe the effects of misinterpretation is to keep all kinds of information locally instead of globally.

4.6. PSALM 2.

PSALM 2 took more care of parameter handling. Every (potentially active) entity could wander through a number of states. The current state depended on how many of the needed parameters already were instantiated. Every entity 'collected' by some primitive action was interpreted as a missing parameter, and the primitive changed its state correspondingly.

Still it was felt that the handling of instantiations was not explicit enough to be natural. The desire grew to allow purposive alteration of parameter instantiations.

The notions mentioned above led to a kind of (minimal) object oriented approach. Each entity should 'know' its parameters, send messages in form of entities, perhaps interpret messages received from other entities, serve as a variable etc. In order to implement these ideas it was helpful to separate the problem of meta-learning into two logically rather different parts: The language a learning system is based on, and the pressure given by the hardware (and the environment) that forces the language to organize itself. (Remember, the hardware is the unchangeable part of the system.)

The next section describes PSALM 3. PSALM 3 allows rather straightforward implementations of programs that maintain the spirit of 'distributed programming'.

4.7. PSALM 3.

4.7.1. The Language.

Most programs (including most AI-programs) show a rather strict separation between procedural and declarative knowledge. This also holds for the many systems written in potentially (syntactically) self-referential languages like PROLOG. The conventional proceeding is to let some comparatively universal algorithms work on an amount of somehow structured data (trees, relations, production rules ...). Even if both the data and the programs are represented in the same syntactic form, say LISP-lists, the semantic frontier usually remains sharp.

A proceeding more similar to the human way to handle information is to let each piece of data 'know' what it is about, and to provide in an associative manner the algorithms that are needed. This leads to concepts like object-orientation, demons etc. 'Real' object-orientation seems to be achieved when all the (non-primitive) parts of an algorithm again 'know' what they are about, what it means to be called by another algorithm a.s.o. Thus the frontier between algorithms and data gets blurred. Data may be informed about the algorithms they are suited to, and the algorithms might know which data to process.

PSALM 3's language is designed to achieve such an indistinctness. A piece of data may sometimes just serve as information for some algorithm, in another context it may be part of a program, or it may trigger one.

To allow self-referential structures in a form that might resemble to sequential assembler code as well as to semantic networks the following claims to the language of PSALM 3 were enforced:

- An entity associated with an action should be able to define at least one 'exit' to another entity, in order to enable the formation of ordinary sequential programs. The exit may be viewed simply as the address of the next instruction.
- In order to create or change sequential programs the system must have a possibility to define or redefine some entity's exit(s). So at least one primitive action that takes two entities as parameters and makes a 'program' out of them has to be incorporated in the language.
- Primitives that can be used for parameter setting must be available.
- Any entity should be associateable in an annotated manner with any number of other entities. 'Annotated' means that not only simple links between entities are allowed, but that there can be additional information about what a link means. This requirement supports a decentralized management of information. It somehow provides the basis for object-orientedness. It also makes it easy to supply semantic comments to certain parts of a program, in order to indicate to other analyzing programs what e.g. some variable is about. Comments are very popular in the field of automatic program synthesis, see e.g. [Dershowitz 83],[Sussman 75].

- The language needs the capability to extend itself (e.g. in order to write new programs that cannot be constructed out of already existing ones, or in order to define new kinds of links ...). So at least one primitive action that is able to create new entities is necessary. Others are needed to put new entities to places where they may get a semantic interpretation.
- The language has to provide a way to inspect any entity that is part of the language. So primitives are necessary that take other entities as arguments and find out whether they are associated with some action, with which one, how the parameters of the action are instantiated, which exit is defined by some entity, etc. Furthermore there must be a way to change behaviour depending on such tests. At least one 'branching' primitive has to be incorporated.
- The language should be endowed with a potential way to follow sequences of actions back into time, in order to allow reflections about the history that led to the current state.
- All the primitives mentioned above as well as their parameters have to be represented as entities in order to close the language on itself.

To handle all primitives and the other entities in a homogeneous way they were implemented in a frame-style manner:

The fundamental structure of PSALM 3 is called a token. In the initial version of PSALM 3's language any token was described by its name, eventually some functional or procedural definition, and a set of ordered pairs of tokens that represented the associations. Such an ordered pair can be viewed as a slot and its filler in the sense of [Minsky 81], or as a subject and an object (in the sense of [Steels 86]), or as "component of the beta-structure" (Newell), or simply as a property and its value (lisp).

[Winograd 75] points out that the notion of a frame is a vague one, and that different people do have different requests to a frame system. An inheritance mechanism along isa-links often is considered to be important. But built-in inheritance is exactly what PSALM 3 wants to avoid (remember the shallow tree problem). If inheritance and isa-hierarchies are important (and certainly they often are so) then a possibility should be given to construct them explicitly. However, the potential possibility to escape the isa-links has to be provided, too. Of course we want the system to learn when to switch between different kinds of using available information.

Frames are often considered as a means to close the gap between declarative and procedural knowledge. A common proceeding is to let frames inherit algorithms. The most uniform (and thus the most beautiful) frame-oriented languages are those where not only the fillers of the slots but also the slots themselves are represented as frames. The slots should 'know' what it means to be asked for a value that is (perhaps virtually) stored on them. Because slots are frames, they again have to be described in form of slots and fillers. Although this looks rather circular, there is no big problem in constructing such self-describing languages. This has been shown in [Greiner 80] where RLL-1 is explained, a self-modifying language that became the basis of EURISKO. A

reimplementation of RLL–1 in Prolog that has been done as a fopra at TUM is described in [Stolcke 87]. Probably the most well-reasoned 'self'-language is KRS [Steels 86] which is based on the insights of intensional logic. KRS also has been used to do learning by discovery, see [Jonckers 86].

PSALM 3's fillers *and* slots again are tokens that can be associated with slots and fillers.

Some of PSALM 3's tokens have a fixed slot interpretation. If for instance a slot called 'action' is filled with a token that has a functional definition the hardware may apply that executeable definition to the fillers of some slots that are interpreted as the parameters. (The hardware is essentially given by one of the top-level–loops described below.)

Of course every primitive action has to be implemented carefully, because it must also handle the cases where some parameters are not correctly instantiated. In such cases the system (in a very advanced stage) should be (potentially) able to find out what went wrong. Perceptions (that also might be triggered by the environment) are used to make information about errors explicit. Of course, perceptions are represented as tokens, too, because only tokens are accessible for the language.

We are looking for a simple 'orthogonal' basis of primitive actions for a Turing equivalent language that allows explicit self–reference. This basis can be rated as an analogon to the set of fundamental lisp–primitives: {car, cdr, cons, (cond)eq}.

Due to the uniform representation of the language the number of initial mental primitives can be heavily reduced. The most simple basis might be given by the three tokens described below. Each has an even number of parameters, because two values are needed to access some distinct entity: A token, and a slot stored on that token, where the filler is regarded as the desired entity. The three basic tokens with a functional definition are:

- *Copy.Copy* takes four parameters (also represented as slots among the associations of the token whose action–slot is filled with the action 'copy'), and interpretes them as follows: The second one is viewed as the name of a slot of the token given by the first parameter. The filler of this slot (if existing) is copied onto the token given by the third parameter where it is stored under the name of a slot given by the fourth. *Copy* can be used to redefine actions, exits, or any other kind of slot.
- *Condeq*. This is the fundamental token for testing and branching. *Condeq* needs six parameters. The first four are taken to determine the fillers of two slots hopefully stored on two distinct tokens. These fillers are tested on equality. The remaining two parameters represent exits that are triggered depending on the result of the comparison.
- *Create_token* (two parameters). This one creates a new naked token that is not yet associated with any other token. The new entity is sent to a token given by the first parameter and stored under a slot given by the second. *Create_token* is the fundamental primitive for self–extension. Newly created tokens can be used as slots or fillers or both. They might get

'loaded' with semantics, when contributing to the 'pragmatic success' of the whole system.

Copy, *condeq*, and *create-token* build the basis for the self-referential behaviour of the language. Manipulation of already existing programs witten in the language can be done in a straightforward way by using these primitive actions. The manipulation of the manipulating programs of course is equally easy, because the 'meta-programs' are also written in the same language.

Another primitive that is able to manipulate tokens was considered to be important:

- *Remove*. *Remove* takes two parameters and is able to erase some slot of a token. (*Copy* overrides.) Using *remove* is the language's only way to reduce the number of entries a token has. This should be important, because the efficiency of access to some token is subject to the 'size' of the token (at least in the current implementation). Because the time of access to some information has an influence on the success of the whole system *remove* is justified.

Some kinds of information the language should be able to gain about itself do have an inherent set-nature. This is true for the set of slots a token can have (there is a primitive called *get_slots* which determines that). Another example can be given if the hardware does a parallel interpretation of tokens. In order to fulfill the requirement of potential looking back into time it seems to be natural to store on every token accessible information about the set of tokens that triggered it.

Of course the previous capabilities of the language are sufficient to represent sets or lists. Yet for reasons of efficiency the symmetry between slots and fillers has been broken a little bit. The implementation in question allowed fillers of slots that were no tokens but lists of tokens.

In order to handle lists the language has to be enlarged by primitives that enable list manipulation. Two of them seemed to be enough:

- *push_token* (four parameters) works analogue to *copy*. The only difference is that no overriding takes place, instead the object that is copied is pushed on the 'stack' defined by the last two parameters.
- *pop_token* (four parameters) splits a list (which is given by two parameters, a token and a slot) into its head and its tail leaving the tail where the old list was and sending the head to a place defined by the other two parameters.

Here is one example of a possible token:

token17:

slots	fillers
action	copy
par1	Clyde[13]

[13] You remember that Clyde is an elephant, and that every elephant is a mammal, and that every mammal is an animal, and that every animal ...

par2	isa
par3	token17
par4	par1
.	
.	
.	
slot354	token2347
support	(token17)
.	
.	

Token17 follows (strict) isa-hierarchies. Why holds that? Token17 works directly on itself (because its third parameter again is filled with token17), and it is resetting its own first parameter with the isa-generalization of its old instantiation (if such an isa-hierarchy exists). Because the next token to be considered by the hardware is proposed by the support-slot to be again token17, a loop structure can be identified. Dependent on the way the hardware is interpreting tokens some perception may terminate the loop.

But how did token17 come into existence? It was created by another token, say token12, that is or was associated with the action *create_token*. Token17 was sent to a place where it was interpreted as a parameter that should be associated with the action *copy* (what again may have been done by some *copy* ...).

Token17 easily can be changed (by some other token associated with a token manipulating action) to a little program that follows neighbour-hierarchies. All there is to do is to replace the filler of the slot par2 (which currently is 'isa') by the token 'is-neighbour', presupposed that this token exists and that the corresponding relation is represented appropriately. The token that would do this change would act like a generic function generator.

The more interesting cases, of course, may be those where many sequences of tokens work on many other sequences of tokens in parallel ... .

Obviously any distiction between different meta-levels seems to be abolished. Circumstances permitting the same sequences of actions may work on the domain level as well as on any other level. PSALM 3's language is well suited to handle ideas presented in [Perlis 85], where a theory of quotation is described with the aim to stay in first-order logic and to avoid antinomies involved within higher-order logics.

4.7.2. Some Words About Garbage Collection.

If the system frequently creates new tokens in an 'uninformed' way it often may happen that tokens get lost. After some time of development there might be no way to access some distinct token by following associations from the set of initial tokens or from the current agenda. Tokens that have become garbage may occupy large parts of the storage (especially during the starting phase when the system does a lot of silly actions), and some kind of garbage collection must take place.

In the case of PSALM 3 garbage collection is not so straightforward as it was in the case of PSALM 1, where a conventional algorithm was used. What does it mean if a token is accessible by PSALM 3's language? Let SA(t) denote the union of the set of initial tokens, the set of perceptions that emerged up until time t, and the current agenda (if the hardware employs an agenda mechanism). We are interested in the set $SA^{\bullet}(t)$ of all tokens that are 'accessible' from SA(t). Informally a token is accessible if a chain of associations starting in SA(t) and leading to that token can be built where all referencing slots in that chain are accessible tokens, too. Some distinct token can be referenced only via slots, and this is possible only if the slots themselves are referenceable by the language.

It seems as if we ought to know the set of accessible tokens before we can define it. This is not true,as there is a way to compute the desired set by computing a fixed point like this:

1.Set $SA_0^{\bullet}(t)$ equal to the set T(t) of all tokens existing at time t. Set n := 0.

2. Repeat:

2.1. Set n := n+1.

2.1. Let $SA_n^{\bullet}(t)$ be equal to $\bigcup_{e \in SA(t)} sa_n^{\bullet}(e,t)$ where $sa_n^{\bullet}(e,t)$ is defined recursively as $e \cup \bigcup_{e' \in C(e,n-1,t)} sa_n^{\bullet}(e',t)$ and where C(e,k,t) is the set of all fillers of e at time t where the fillers and the corresponding slots are in $SA^{\bullet}{}_k(t)$.

until $SA_n^{\bullet}(t) = SA^{\bullet}{}_{n-1}(t)$.

3. Set $SA^{\bullet}(t)$ equal to $SA_n^{\bullet}(t)$. The garbage is given by T(t) – $SA^{\bullet}(t)$.

Informally : Repeat conventional garbage collection thus reducing the number of accessible slots until this number can not be reduced any more and a fixed point $SA^{\bullet}(t)$ is reached. At the end of that procedure only tokens that are accessible via accessible slots remain.

The actual implementation performs only partial garbage collection from time to time, due to reasons of efficiency. This does not mean a fundamental restriction. Now or then all lost tokens are recollected. The only restriction exists in the fact that not all garbage tokens are recollected at the same time. In order to save time the garbage collector avoids to compute the complete fixed point at time t, but it removes garbage slots whereever it is possible to recognize them during the first and only cycle of recollection. So the next time the collector is triggered it may recollect tokens that were not recognized to be lost during the previous time, in addition to new garbage tokens.

4.7.3. A Link to Geometric Fractals.

A geometric fractal can be defined with the help of an initiator and a generator. Informally: the initiator is modified in a way determined by the generator. In the case of figure 2 the initiator is a line. This line is altered, its middle third is replaced by a geometrical figure similar to the generator consisting out of three of the four lines defining a square.

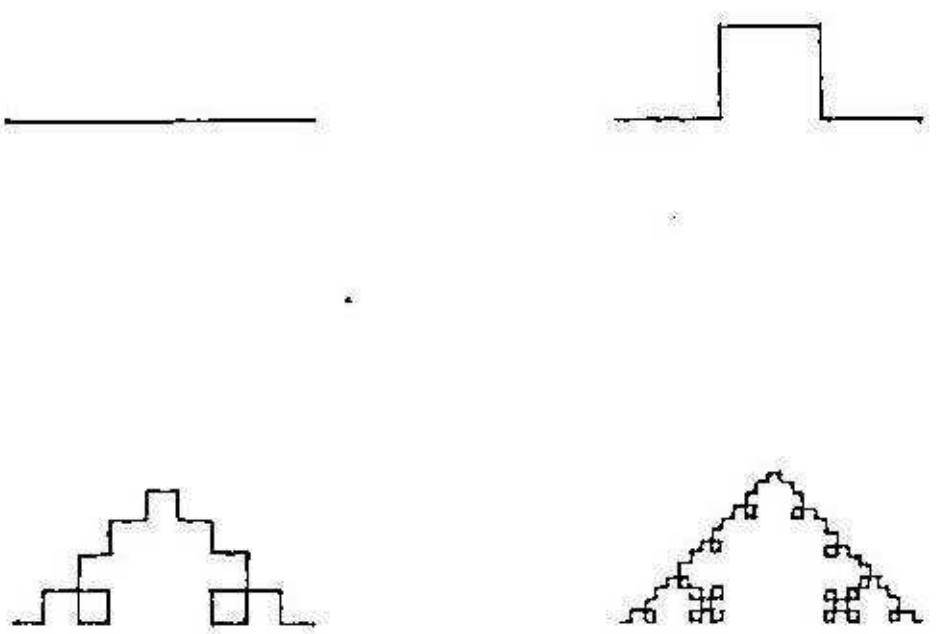


Abb. 2

Thus five new lines can be identified, and to each of them the procedure depicted above is applied again. If such a proceeding is repeated an infinite number of times, a self similar structure emerges: The whole is similar to an infinite number of its parts.

There is no need to follow such a stiff and deterministic scheme. The structures similar to the generator may be inserted into the developing initiators according to some random distribution. Certain distributions, analogically applied to the construction of 3-dimensional fractals, produce results strikingly similar to for instance stone formations, plants, crystals etc. ([Mandelbrot 83]). In such cases the evolving structures are only 'nearly' self-similar.

An example is the growing of ice-crystals on a window, producing fractal forms. This growing is constrained by the shape of the window as well as by the already existing crystals: If two different parts of the evolving structure meet, they may not overlap.

Let's draw an analogy to self-referential programming. Let us view the initiator as a program, and the generator as another program that is able to do program modifications. Then the generator could be applied to parts of the initiator, a different and more complicated program may be the result, and so on.

It becomes interesting when the initiator and the generator are equal to each other. This would mean that the initiators do not have to be changed always nearly in the same way, because by being changed the generator changes, too. I would like to call this the development of an 'algorithmic meta-fractal', because a transformation constructing a new stage of the 'algorithmic' fractal is in general also transformed at the same time it works.

The analogy to the window is the frame for development given by the world to which the meta-fractal may be coupled by effectors and receptors. The pragmatics of the world is like the window, but it has a very complicated, multidimensional, also fractally broken frame. If the initiator (and thus the generator) can take the form of any Turing computable program, the algorithmic fractal should grow and refine itself thus more and more filling the window ...

Let's have a closer look now on possibilities to introduce pressure, in order to force the initiator-generator to fill its window.

4.7.4. The Pressure

A row of mechanisms are thinkable to force the language described above to organize itself. Common to all of them is some pragmatic pressure that has to be established by the domain critic: It is the critic who states whether certain actions performed within the domain are 'useful' or not. Dependent on the utterances of the critic the hardware should favour a development of the language that results in successes within the environment. The semantic interface between our conceptions and the language is given by the evaluation functions the critic uses and the way the hardware translates criticism into 'probabilities for informed structures to survive'.

During all the tests that were executed with PSALMs the critic spent payoff in form of numbers. High numbers were given if a problem was solved well, low numbers in other cases. It is not true that a lot of information is lost by reducing perhaps complicated problems to simple numbers. This is because the different contexts that lead to the donation of high or low numbers, respectively, represent a lot of information, too. This information somehow should be reflected by the token language in form of informed structures.

To introduce competition every token can have a strength. Strength is measured with the same basic unit as payoff (natural numbers were chosen to represent payoff and the strengths in the actual implementation). The hardware has to favour tokens with high strength, and a token may have a high strength only if the amount of payoff has been reduced adequately some time before.

The way strengths are handled links the language to the world. Naturally the strength of some token is stored as an association in form of an accessible slot. In order to maintain the philosophy of the language, namely that everything should be explicit, the number representing the strength of some token also is represented as a token. So any number can be associated with information in the same way as any other token.

The explicitness of the strengths implies as a consequence that actions like *copy* need to be restricted. This is because it must be prohibited that the language *copys* high strengths to tokens that do not deserve it. More precisely: The slot called 'strength' may be filled with a token representing a number only if there is enough payoff to do so. Every time the language fills the strength–slot of some token the amount of payoff is reduced correspondingly.

Another mental primitive was introduced that is able to do the inverse action: *decrease_strength* can increase the amount of payoff by decreasing some token's strength.

The idea behind this proceeding was: The sum of payoff plus the strengths of all tokens may be increased only by the domain critic. What is happening to the payoff is hidden from the eyes of the critic. But it must be guaranteed that the system can not escape the pragmatic pressure forever by creating payoff or strengths out of nothing. Of course this does not mean that the language is not potentially able to reason about strengths and numbers. But it is well advised to create some new slots to support such reasoning processes, because strength–slots are treated specially.

A difference to PSALM 1 is that the associations between tokens are not weighted any longer. This has mainly aesthetic reasons: On the one side it is not clear what for instance a weighted link to the actual parameter of some action should mean. A parameter is instantiated, or it is not. This kind of all-or-nothing nature seems to be inherent to most of the initial slots. On the other side there are slots where a weighting could be justified by experiments carried out with neuronal nets [Rumelhart, Zipper 85][Buhmann,Schulten ?]. This includes the 'support'-slot which is used to propose one or more tokens to be interpreted next. But the introduction of special weighted links and their special treatment (by a row of adapted mental primitives) would grievously break the symmetry and beauty of the language's previous form.

Instead one simple number indicates the worthiness of the whole complex called a token, including the many all-or-nothing associations that may belong to it. [Lenat 77] gives more justification for a similar proceeding in AM.

4.7.4.1. Whistling.

How can PSALM 3's language start to develop? Note that in the beginning it does not even know the difference between domain and mental primitives. No token is associated with any kind of information in the initial state of the system.

Schulten emphasizes the importance of 'random whistle' for learning systems based on a neuronal net architecture [Buhmann, Schulten ?]. I want to take possession of these ideas and transform them into a symbol-manipulative analogon.

PSALM 3's hardware whistles, too: If a token that is to weak or that does not make any 'syntactic sense' (which should be easy to find out) is considered by the hardware, that token is associated randomly in a way that makes sense at least on the syntactic level. This is true especially for tokens associated with an action but also with clearly wrong parameters. 'Clearly wrong' means that the decision whether something is wrong can be taken on a pure syntactic basis.

If the syntax makes sense the hardware executes the action (what may trigger perceptions that indicate semantic errors), and usually one or more new tokens are proposed to be executed in a controlled manner: control suppresses the symbolic whistle.

Whistling somehow represents PSALM 3's fundamental principle to handle *Erstmaligkeit*, and to do accommodation (although accommodation in an advanced stage of development should be mainly done by more informed structures). The random element introduced by whistling reflects the fundamental fact that unforeseen things may happen in the 'outer' world. Its analogon in **meta-evolution** is random crossover.

4.7.4.2. Equilibration and PSALM 3.

Assimilation takes place whenever some perception or a pattern of perceptions is triggered that again triggers 'stabilized' (sequences of) actions. [Ginsburg,Opper 75] criticize Piaget because he did not make clear how and under which

circumstances equilibration takes place. But the up and down of assimilation (expectation driveness) and accommodation can not be defined in detail. It is the pragmatic context that has to bring on an equilibrium. Simple models suited to introduce equilibration are proposed next.

4.7.4.3. Possible Top–Level–Loops for PSALM 3.

Different kinds of agenda mechanisms (similar to the one used by PSALM 1) have been tested. Differences were given e.g. by the way competition was enforced. One way to determine the 'power' of a token at time t is to compute a bid by multiplying its strength with the number of tokens that proposed it during the last time step. Another way is to consider the strength of the proposing tokens, too.

The highest bidders should win. What does this mean? One could introduce a maximal lenght *l* that must not be exceeded by the length of the 'active' agenda. At time t the agenda A(t) is sorted by the 'power' of the tokens it consists of. Not more than the first *l* tokens of the sorted agenda may be interpreted in parallel. Every token that got a chance to be active is thrown from the agenda, and its strength may be decreased if there is no payoff available or increased otherwise[14]. If the strength is below a certain threshold, the token may be 'over–whistled'.

Should some tokens out of the non–active rest of A(t) become part of A(t+1), building some kind of short term memory? Or should A(t+1) solely be built out of tokens proposed by the active part of A(t), in addition to some eventually appearing perceptions? Should perceptions really be part of the agenda, or should they be kept on a special perception list where they can make proposals for A(t+1) ? Should perhaps only one token be interpreted at each time step? Or three? Or 231 ? Should a bucket brigade scheme[15] be employed by default (every token has to pay a part of its bid to its proposers) ?

Obviously there are many dimensions along which one could vary the central cycle which is changing the system's state at each time step. All the possibilities mentioned above have been investigated, but not in an exhaustive manner. A systematic examination of the probably very complex interdependencies of combinations of schemes miscarried because of the large number of possible combinations.

Furthermore the experiments indicated that the basic structure of the top–level–loop may be not so important at all. Before interpreting some results let us have a look through the eyes of statistics to find out what we can expect if PSALM 3 starts with nothing.

---

[14] The strength has to be decreased in order to prevent forever lasting loops that do not contribute something meaningful. On the other side there is no principal need to increase strengths by default, because the system could do the payoff management on its own: Whenever there is payoff the system may take over the task of sending payoff to tokens it considers to be adequate. Payoff management is one of the processes which have to be adaptable to changing situations.

[15] B.b. schemes also may not remain *le dernier cri* during the ongoing development of the language. But they may be helpful during the initial phase. (See the footnote above.)

4.7.4.4. Statistical Expectations.

Statistical prophecies about PSALM 3's behaviour can be made only if they refer to the initial phase. It is easy to predict how often certain constellations of tokens will appear during the phase where most associations between tokens are done in a random manner by the symbol-manipulative whistle.

But it is practically impossible (in general) to predict what will happen if the whistle is more and more suppressed by (self-referential) control. This is no reason to become unhappy, the contrary is true. If we could prophesy all details of further development there would be no need for a meta-learning system. In the chapter about evolution I expressed the supposition that the unpredictability of meaningful states of learning systems might be fundamental, and that there often may be no algorithm to compute such a 'relevant' state where the algorithm is clearly cheaper than doing the whole simulation of the system. Statistics may be no means to deal with 'chaotical' systems, not even with the relevant features of chaotical states (where 'relevant' is defined pragmatically, of course).

But statistics is well suited for the ignition phase. Initially PSALM 3's disposes of a set of n initial tokens. What is the probability for the spontaneous instantiation of the four parameters a token associated with the action *copy* can have? If we determine 4 actual parameters out of the set of initial tokens this probability is $\frac{1}{n^4}$. The probability for the spontaneous creation of a sequence of two *copies* where the support-slot also has to be instantiated correctly is $\frac{1}{n^9}$, and so on[16].

These numbers are not quite as horrible as they seem to be because usually there are many syntactic ways to achieve a semantic goal. But obviously tokens that need a lot of parameters are handicapped compared to simpler ones.

A fundamental dilemma of the ignition phase is: if n is too large we probably will not observe very exciting effects for a long time. If n is too small the language looses its self-analyzing capabilities, because there is a need for a certain amount of slots in order to describe the language explicitly.

It should be emphasized that this problem really belongs to the ignition phase. If there already is a lot of knowledge about how to set parameters and 'exits' in a meaningful way statistics plays a minor role. But I do not see any plausible way to jump over the initial stage, because I do not know how knowledge naturally is represented in a developing language like the one of PSALM 3. PSALM 3 was *designed* to find out how the many kinds of using available information like analogical connections could look like. If I had known this before there would have been no motive for the construction of PSALM 3. But all I can say after watching it at work is that the representation of

[16] This holds only if whistling is restricted to the initial tokens, which is the case in the actual implementation. So programs built out of initial tokens are rather 'unsafe' compared to programs consisting mainly out of newly created tokens. A clever system should notice this and act adequately.

knowledge probably might look rather different from the representations employed by conventional (AI) programs, as will be seen next.

4.7.4.5. Some Empirical Results.

The domain of moving a robby through a room had been included into the system (robbies are very popular among machine learners, Holland e.g. tested the b.b. with a simple robot, too [Holland 84]). The term 'included' means that some domain actions like 'step–upward', 'step–left' etc. were incorporated into the vocabulary of the language. Furthermore some new perceptions were allowed to be triggered in adequate situations, like 'error_there_is_a_wall'. A critic spent the more payoff the closer a robby (which was directed by PSALM 3) came to the place it should walk to.

Actually certain sequences (better 'clusters') of actions evolved that led to stable states 'fulfilling' what the simple evaluation functions of the critic requested. Stable states in this context mean situations where the whistling is mainly suppressed and the same actions are triggered again and again in answer to some perception like 'start_to_go_to_the_left_upper_corner'. Usually all tokens participating in a stable state have maximal strength and are very unlikely to be destroyed by the whistle some time.

Of course stable states depend on the generosity of the critic. If the critic gives enough payoff for middling solutions then stable states may evolve that do not have much to do with the pragmatic context we had in mind when posing the problem. Under such circumstances it was often observed that tokens were playing with each other throwing slots around or defining some stacks on other tokens thus pumping them up. Usually the system executes enough 'good' domain actions to satisfy the evaluation functions, but its 'free' time is often spent with behaviour that does not make sense in human eyes. Yet it is just following the simple but general rules of evolution. It is our problem to define the adeqate 'pragmatic intersection'.

Sharpening the evaluation criteria causes the unlearning of behaviour that is not suited to solve the task. Then it usually takes some time until better suited stable states evolve that have to do more with what the observer had in mind.

It was interesting to observe that constraints shortening the agenda led to more sequential forms of behaviour while a large agenda supporting parallelism led to the 'scattering' of information in clusters:
In the first case often small programs could be identified representing loops suitable to make the robby step into the same direction for a while.
In the second case such discoveries were rare: Many tokens were associated with some often meaningful action, but they were triggered in a way that remained opaque. Although Robby usually came close to his goal the ways he choose to achieve it were quite different during consecutive criticism–periods. The parallel version left the impression that perhaps some 'general notion' about how to approach the goal made the robby run.

Furthermore little artificial damages to the token structures were swallowed much easier by the parallel PSALM. This redundancy comes closer to things

observed in neuronal nets and human brains. More about redundancy in a later section.

Introducing a bucket brigade scheme did not improve the performance significantly, at least not within the choosen domain. But in general simple schemes like b.b.s should be interesting only during 'ignition' (see the last footnote).

Unfortunately the advantages of explicit possibilities to act self–referentially could not be underpinned empirically. This is due to the inherent time problems involved with the ignition phase. Although PSALM 3 has learned a few things it still is very far away from leaving the ignition phase behind. In fact all the domain dependent little sequences (clusters) of actions it has learned owe their existence mainly to the whistle.

4.7.4.6. Some First Reflections on the Empirical Results.

Every really self–referential evolving system should accelerate its evolution (hopefully exponentially or even faster – it depends on the complexity that the environment allows). But in the beginning the flat part of the 'informedness–curve' may stretch over a very long ignition phase. This is the case with PSALM 3: A language that starts from nothing certainly will need a long time before leaving the impression of being well informed[17]. PSALM 3 in its current implementation allowed the interpretation of about 30 tokens per second (this value depends on the agenda mechanism), which is magnitudes below what might be needed.

Although human neurons are certainly very different from PSALM 3's tokens, it is inspiring to play with numbers a little bit: Many of the 10^9 to 10^{11} neurons within the brains can be active simultaneously, probably clearly more than 10^9 per second. These numbers represent magnitudes that also seem to be desireable in the case of PSALM 3, because under such preconditions many little meaningful self–referential sequences are thinkable that should evolve within a few seconds spontaneously, building the basis for more complicated ones.

Since all I dispose of is a comparatively slow machine, I might want to jump over the initial phase by providing all well–suited little sequences from the beginning. But which are the well–suited ones? How many different ones are there? In what form do they appear (sequences or redundant clusters (how does in general a redundant cluster look like)) ..? One could introduce explicit isa–hierarchies, written in the token–language. A problem arises: programs as we would write them are not very redundant. So if there appeared a task where the generalization–methods do not work well, parts of the hierarchy probably will be 'unlearned'. The unlearning may affect fundamental parts like some essential climbing algorithms which are supplied in a non–redundant form by human programmers. Under such circumstances the whole hierarchy may be

[17] Biological evoulution shows another case of a long ignition phase. It took at least 3 billion years to bridge the gap between the first organisms and the first users of tools. It took another few million years to do the step to division of labour. It took another few thousand years to build a computer. The first human being who used a tool to build another tool already had one foot on the moon, from a cosmical point of view.

lost. The solution is not to put genl./spec. into the undestroyable hardware, because then we would be there where we started. Exactly because it became clear that flexible learning can not rely on a few things like isa-hierarchies, the language of PSALM 3 was designed to be flexible enough to learn new methods (most of which probably would be called analogical). The motive behind PSALM 3 was the belief that the number of important methods is too large and that the methods themselves are too opaque and context-dependent to be programmable.

The fundamental idea behind this work is 'explicitly closing an initial system on itself'. But fundamental ideas should be questioned, too. Of course the potential to act self-referentially must be provided in a learning system. But is it necessary or only natural to introduce this potential explicitly, as e.g. in PSALM 3's language? The human neuronal system does not seem to be closed on itself in a way similar to PSALM 3's, still it obviously allows structures that are self-referential. Something like a token might be represented as a (very redundant) cluster of neurons in our brains, and this might have unknown advantages.

Could it be that systems based on the simultaneous working of a great many of very simple devices can leave the ignition phase behind in a clearly shorter time? It might be possible that systems supporting and making use of conventional concepts like parameters, functions, if-then-else etc. are not suited to manage the fuzziness of the world, at least not within a reasonable time-interval. Although PSALM 3 represents a hybrid between a neuronal net and a conventional object-oriented system, there might be too much influence from the shore of symbol-manipulation. The parallel versions of PSALM 3 tend to scatter information, and to be redundant. But the tokens of PSALM 3 seem to be not so adequate for the distribution of information, as for instance methods known from the theory of associative memories are [Kohonen 77,80]. From watching the behaviour of parallel SALMs one might get the impression that the very important principle of redundancy oppresses the basic token structures. The principle of error-toleration might be so important that there can't be a powerful learning system without it. So this principle takes what it gets, and in the case of PSALM 3 it gets tokens. But other methods might provide a much better frame for the development of redundancy (see the chapter about future research). It is the old problem: Potential Turing equivalence is easy, but how to exploit it naturally after having achieved it?.

5. Future Research.

5.1. PSALM 3-Specific.

Different kinds of pressure on PSALM 3's language are imaginable that may lead to significant improvements under certain circumstances. Up until now only 'positive', excitatory support has been given to competiting tokens, for example. But results obtained by [Dell 85] and [Buhmann, Schulten ?] as well as observations made by neuro-physiologists indicate that inhibition may be as important as excitation. The natural way to introduce inhibition into PSALM 3 is to define an accessible slot 'inhibit' with a special hardware interpretation contrasting the interpretation of the 'support'-slot.

Promising directions like inhibition have not yet been investigated. An important reason for this is the assumption that only 'rich' domains will provide enough structure to show the advantages of such newly introduced concepts. ('Rich' means e.g. the inclusion of pretentious pattern recognition tasks.) But complicated domains usually imply complicated critic-actions, too. This means: More time will be needed, and time is something that is scarce at the moment. In general I believe that my machine should be at least 1000 times faster to show really interesting effects (the most interesting of which is self-reference, of course). I would love to see an analogue to PSALM 3 being implemented on the Connection Machine [Hillis 86] (more than 65 000 processors working in parallel, building connections among themselves, every processor e.g. being used by one active token, and the whole coupled with some interesting domains involving pattern recognition tasks as well as so-called higher-level problems).

Since PSALM 3 is settled somewhere between symbol-manipulation and connectionist models there are at least two directions one might naturally follow in order to abridge the ignition phase, if it is abridgeable at all. Of course, one way leads to more symbol-manipulation, the other one to less.

5.2. Self-Reference and Associative Memories?

In 4.7.4.6. I expressed the supposition that an essential foundation of learning, namely redundancy and the toleration of errors, may be achieved more consequently with methods inspired from the theory of associative memories. The typical feature of such memories is that information is accessed by its content rather by its address. A popular related software method is hashing, but conventional hashing is not suited for the recollection of 'fuzzy' knowledge accessed e.g. via incomplete keys. A nice content-addressable memory (CAM) should allow the recollection of data if for instance only some constraining conditions are specified. It should swallow noisy inputs and still find the most 'adequate' output. Information should be retrieveable even if the keys are damaged or incomplete (--> autoassociative memories).

A classic method [Kohonen 77, 80] is to represent keys as vectors out of $\mathbf{R}^n$ and map them by e.g. a linear transformation to some output vector $\in \mathbf{R}^m$ where the matrix doing the linear operation stores the 'correlations' between the patterns rather than the pattern themselves. [Kohonen 77] also describes

properties of an optimal (in the sense of least squares) adaptive process that transforms the transformation matrix thus achieving a learning effect.

[Geiger 87] takes input vectors out of $\{0,1\}^n$ representing objects where a 1 or a 0 indicate the presence or absence of some particular property an object may possess. A simple adaptive algorithm makes the system learn new input vectors and the corresponding outputs.

[Kohonen 77,80] shows how activities stretching over a longer time interval (programs!) can be implemented in CAMs by using feedback in a straightforward way.

The methods mentioned above are reminiscent to the way a hologram stores information: Essentially a two-dimensional fourier transformation 'scatters' knowledge contained e.g. in a photograph over a large area, where every small part of the hologram carries information about every part of the photograph. Recollection does not take place by considering a small part of the hologram but by applying a re-transformation to the whole, or at least to a larger part of it.

The scattering of information leads to the toleration of errors. Damaged or incomplete inputs may be harmless as long as the degree of damage is sensibly constrained. The allowed degree of 'fuzziness' is limited e.g. by the numbers of items 'super-imposed' within a CAM: The more items, the noisier the outputs. Neuronal nets also tend to scatter information, and so did the parallel versions of PSALM 3. But may be that tokens are still too 'localizeable', too symbol-manipulative, too much 'all-or-nothing'. Redundancy and the tolerance of errors is possible by the creation of sequences running in parallel while having more or less the same semantic effect. But could it be that a token already is too bulky a primitive? That the lesson 1 from PSALM 1 still holds? Could it be that structures that can do what a token can do should not be introduced explicitly, but evolve in a perhaps much fuzzier way? Could this lead to a faster and more natural ignition phase?

How could e.g. a system based on fuzzy triggering conditions determined by associative memories be 'closed onto itself'? I am very interested in this question, and it will have a high prority among the things I want to examine next.

5.3. The Symbol-Oriented Way.

Are there some (formally expressible) principles that have not yet been considered and that may be sufficient to explore the giant field of informed structures (esp. analogies) in an EURISKO-like manner? Is there a way to enable a system based on genl./spec. to change its basis without destroying itself? EURISKO builds a hierarchy of heuristics, which leads to the reduction of the number of heuristics applied in a particular context ($n \dashrightarrow \log^* n$). But often it would be wiser to leave the hierarchy and sidestep into another related domain, instead of fixing oneself to the one-dimensional field of genl./spec. (shallow tree). When to do this? Trial-and-error will help to find out, and the insights have to be placed appropriately as new heuristics indicating how to move in the

space of analogue heuristics, when to leave the isa-links, etc. The result may again be a distributed system (graph, network), where each node of the graph might 'know' good directions for continuing a search. So one might imagine that the isa-hierarchy and its specific algorithms become overgrown by more analogical methods. This imagination leads to problems exemplified next:

From time to time EURISKO invents new relations (slots) by specializing or generalizing old ones. (This is eased by RLL-1 [Greiner 80] which takes over the problems involved with consistency maintenance.) If we want to escape strict genl./spec. then we should consequently provide a possibility to define e.g. slots that are analogue to available slots. (Recursively, please. The part of the system that invents definitions should be able to work on itself, of course.)

But what does consistency maintenance mean then? Can a useful definition of consistency been given in such a case? What about the fuzziness involved with analogy? Analogies often help without being consistent at all.

It seems as if the pure symbol-oriented way is blocked with some obstacles, too. But perhaps the most promising attempts to understand learning will be inspired from both the high-level *and* the low-level shores of the ocean of research on knowledge representation.

5.4. Common Features of Object-Orientation and Connectionist-Models.

From my point of view the fields of neuronal nets, CAMs etc. on one side and object-oriented programming on the other side somehow converge. The common element that is more and more emphasized on both sides is the decentralization of information. One could argue that in object-oriented languages (OOLs) information is distributed in order to gain transparency while e.g. connectionist models make the distribution very unclear. However, this argument does not take care of the different starting-points: Neuronal nets are usually forced to organize themselves, OOLs are usually organized by programmers who had something in mind. No wonder they understand what they have programmed (although there are cases where this is doubtful). In the first case the semantics has to develop, in the second case it is directly 'put' into the system and labeled with names that make sense to the programmer.

The hybrid PSALM 3 soon gets (nearly) as untransparent as a neuronal net. The names it defines do not have any relation to human thoughts, still the corresponding tokens do carry semantic information, which can be objectively measured by including the pragmatic aspect of information.

A difference may live in the concept of physical neighbourhood introduced in models of neuronal nets. Within PSALM 3 there is no notion of spatial 'distance': Tokens may manipulate each other although their physical locations in the storage are separated by miles.

But sequences of tokens may alter other tokens only if they 'know' their names, which means that there must be some slots which might serve as an entrance to other areas of the storage. So one might like to think of a 'knowledge distance' between certain clusters. But this notion stays informal, any sensible definition of a metric based on such a distance misleads. This is because there is an

infinity of unforeseen ways to transport information, some of them more 'effective', others less.

Another important difference to neuronal nets may be the availability of annotated (and thus relational) information. Or is this a fallacy? In [Wahrn.u. vis.Sys] ... reports an observation made by examining human neurons: There are not only axons reaching from cell to cell, but there are connections between the axons, too, and nobody knows what they are used for. A speculative question: Could interaxonal connections mean an indexing of information, one axon indicating a property, the other one an instantiation?

I believe that advances in machine learning will be made by people who are familiar with both the symbol-manipulative and the 'low-level' side of AI. [Hofstadter 83][18] advocates the view that no great progress will be made with the understanding of the 'secondary' processes of mind unless the primary processes are not tackled (pattern recognition etc.). Browns statement in [Bobrow, Hayes 85] can be interpreted in a similar way:

> "I had also hoped that by now we would have created more significant bridges between symbolic and numeric computation where each leverages the other."

In their answer to [Richie,Hanna 84], [Lenat, Brown 83] argue to view their 'concepts' as a "new generation of perceptrons" that "opens exciting research directions in the construction and orchestration of large parallel cognitive systems." (For a review of some limitations of conventional perceptrons see [Minsky 69]).

5.5. Domain Complexity.

I suspect that there may be no way to avoid a long ignition phase for a learning system, in the best case we might be able to abridge it a bit. Probably we actually have to start some self-referential mechanism on a really fast device. Perhaps there is no sensible way to essentially shorten the time needed for collecting analogical knowledge. Strong evidence for the correctness of this belief is provided by our own biological and social development.

A growing number of people believe that the fundamental ingredience of intelligence is complexity. The complexity of the domain(s) is equally important. Only if the environment is diverse there can be a diverse picture of the environment within a learning system. A system like PSALM 3 should be confronted with a large number of domains. Only if there is a lot of potential information from different domains there is a sense in trying to find common features and to analogize.

Since no programmer will be pleased (or not even be able) to represent a large number of domains in a computer one should presumably make use of the richness that already is available outside of the machines: The 'real' world. I would not be astonished at all if the first systems that are considered by humans to be really learning are coupled to our world by broad channels building receptors

[18] If you read [Hofstadter 83], also read [Newell 83].

and effectors. Perhaps this is the only way to gain complexity in an unstructured system: To make use of the great amount of potential syntactic information that already grew during the last 10 or 20 billion years (this growing can be regarded as the essence of evolution, remember).

6. The End.

Why does the word *'self'* have such a strong attraction to human thoughts?

The most beautiful functions usually are considered to be those that partly are defined by themselves: computer scientists love recursion. The most interesting programs often are those that write programs in the same language they are written in. What is the reason for the beauty that can be found in things that are defined by or working on themselves?

The world seems to be structured in a way that often is well–described by certain 'critical' fixpoints that serve to bootstrap the rest. As one of many examples for such defining fixpoints consider the slot 'ToGetValue' out of RLL–1 [Greiner 80].

'ToGetValue' is a slot that is virtually stored on any slot, including itself. It serves to retrieve a function one should use to retrieve the value of some particular slot. In order to find that function it is necessary to find out what it means to ask for a value stored on the slot 'ToGetValue', this means the value of of 'ToGetValue:ToGetValue' must be computed.This value is one of the few initial fixpoints RLL–1 needs to bootstrap itself.

An example of the 'self–nature' of the physical world: The geometrical appearance of our environment seems to be fractally broken [Mandelbrot 83]. 'Fixpoints' to construct fractals are usually simple: The mere recursive application of some generating principle results in an often astounding complexity.

Since the world often gets complicated by the application of simple principles to themselves, it seems to be natural to reflect this complexity also by applying simple principles to themselves. 'Natural' means easy! We like recursion or self–referential languages because they appear to work although the particular principles in each case do not seem to submit a lot of information. Because these principles work *and* still are easy to understand we like to believe that they have to do with the 'essence behind the things'.

The word 'self' also gives some justification for considering the sciences of the mind (call it computer sciences or cybernetics or cognitive science etc.) as the ultimate sciences, as I want to make plausible:

Some scientists try to understand the physics of elementary particles. Some try to understand the nature of micro–biological evolution, and some try to understand why $a^n + b^n \neq c^n,\ a,b,c,n \in \mathbf{N}, n > 2$.

But isn't the most exciting science the one that tries to understand the nature of understanding? Doesn't this science potentially include all the others? Isn't this science the 'fixpoint' that might serve to bootstrap the other sciences? Understanding how to understand requires to be informed about information, to acquire information about how to acquire information, to learn how to learn.

Acknowledgements.

I wish to thank Dr. Werner Konrad for the encouragement he always was willing to give. Many discussions on the nature of learning helped to crystallize the ideas presented above. Werner Konrad also represented an inexhaustible source of references to related literature.

I also like to thank Thomas Laußermair, whose many valuable comments on this work helped to clarify it a lot. Thomas shares my enthusiasm on the subject, and talking with him always resulted in less indefinite notions about the essence of learning.

Hans-Peter Dommel also sacrificed a lot of time to reading and commenting the paper. Hans-Peter has made many suggestions for improvements, which have been incorporated and which changed the face of this work.

Andreas Stolcke (who re-implemented RLL-1) also contributed to the final version of this paper.

Thanks to those who reduced the number of syntactic errors.

Thanks to mama and papa, since without them this work never would have been done. Thanks to grandma and grandpa, and to Adam and Eve, for the same reason.

7. Bibliography.

There are a few papers which are marked with a '?', which means that I could not find out the corresponding year of origin.

[Bobrow,Collins 75](Eds.)
Representation and Understanding. Advances in the Study of Cognition
Academic Press, New York 1975.

[Bobrow, Hayes 85]
Artificial Intelligence: Where Are We
Artificial Intelligence 25 (1985).

[Buhmann, Schulten ?]
A Physiological Neural Network as an Autoassociative Memory.
Physik-Departement, Technische Universität München.

[Charniak,McDermott 85] (Eds.)
Artificial Intelligence,
Addison Wesley, 1985

[Clocksin, Mellish 84]
Programming in Prolog.
Springer Verlag, 1984

[Cramer 85]
A Representation for the Adaptive Generation of Simple Sequential Programs
in [Grefenstette 85]

[Crutchfield, Farmer, Packard, Shaw 87]
Chaos
in Spektrum der Wissenschaft Feb.87.

[Davis 80]
Meta-Rules: Reasoning about Control
Artificial Intelligence 15 (1980).

[Dell 85]
Positive Feedback in Hierarchical Connectionist Models: Applications to Language Production.
in Cognitive Science 9 (1985).

[DeJong 75]
Analysis of the Behaviour of a Class of Genetic Adaptive Systems

Ph.D. thesis, Dept. of Computer and Comm. Sciences, University of Michigan 1975.

[Dershowitz 83]
The Evolution of Programs.
Boston: Birkhäuser 1983.

[Dickmanns, Schmidhuber, Winklhofer 86]
Der genetische Algorithmus: eine Implementierung in Prolog.
Arbeit zum Fortgeschrittenen-Praktikum
Technische Universität München.

[ECAI 86]
European Conference on Artificial Intelligence
Brighton (U.K.) 21.–25. July 1986.

[Eigen 86]
Stufen zum Leben. Die Entstehung des Lebens aus molekularbiologischer Sicht.
in [Maier Leibnitz 86].

[Feigenbaum 81](ed.)
Computers and Thought.
New York: McGraw-Hill 1963.

[Forrest 86]
Implementing Semantic Network Structures Using the Classifier System.
in [Grefenstette 85]

[Geiger 87]
Only the address of the firm can be given in this case:
Gerhard Kratzer GmbH, Automatisierungstechnik München
Maxfeldhof 6, 8044 Unterschleißheim.

[Ginsburg,Opper 75]
Piagets Theorie der geistigen Entwicklung. Eine Einführung
Ernst Klett Verlag, Stuttgart 1975
original: Piaget's Theory of Intellectual Development. An Introduction
Prentice-Hall, Inc., Englewood Cliffs, New Jersey 1969.

[Grefenstette 85](ed.)
Proceedings of an International Conference on Genetic Algorithms and their Applications.
Carnegie Mellon University, Pittsburgh, P.A., July 24–26 1985.

[Goldberg 85]
Genetic Algorithms and Rule Learning in Dynamic System Control.
in [Grefenstette 85]

[Greiner 80]
RLL-1: A Representation Language Language.
Expanded Version of the paper published in the proceedings of the First National Conference of the American Association for Artificial Intelligence.
Stanford University 1980.

[Haase jr. 86]
Discovery Systems
in [ECAI 86]

[Hillis 86]
The Connection Machine.
MIT Press 1986.

[Hofstadter 85]
Gödel, Escher, Bach: Ein endlos geflochtenes Band.
Klett-Cotta, Stuttgart 1985.
Gödel, Escher, Bach: An Eternal Golden Braid.
Basic Books, New York 1979.

[Hofstadter 83]
Subcognition as Computation.
in [Machlup,Mansfield 83]

[Holland 75]
Adaption in Natural and Artificial Systems
University of Michigan Press, Ann Arbor, Michigan, 1975.

[Holland 85]
Properties of the Bucket Brigade
in [Grefenstette 85]

[Holland 86]
Escaping Brittleness
in [Michalski 86]

[INTERLISP 85]
Siemens INTERLISP Version 4, Benutzerhandbuch.
Siemens AG, ZTI SOF 222, 1985.

[Jonckers 86]
Exploring Algorithms Through Mutations.
in [ECAI 86]

[Knuth 74]
Surreal Numbers
Addison-Wesley Publishing Company 1974.

[Kohonen 77]
Associative Memory
Springer 1977.

[Kohonen 80]
Content-Addressable Memories
Springer-Verlag 1980.

[Küchenhoff 86]
Synthesis of Prolog Programs by Knowledge Guided Genetical Learning.
Diplomarbeit, Technische Universität München 1986.

[Küppers 86]
Der Ursprung biologischer Information
Piper, München 86

[Lenat 77]
The Ubiquity of Discovery.
Artificial Intelligence 9, 1977.

[Lenat 82a]
EURISKO: A Program That Learns New Heuristics and Domain Concepts.
Heuristic Programming Project, Stanford University, Stanford, Cal. 94305.

[Lenat 82b]
The Nature of Heuristics
in Artificial Intelligence 19 (1982).

[Lenat 83]
Theory Formation by Heuristic Search
in Artificial Intelligence 21 (1983).

[Lenat, Brown 83]
Why AM and EURISKO Appear to Work

in Artificial Intelligence 23, (1984).

[Machlup,Mansfield 83]
The Study of Information.
New York: Wiley 1983.

[Maes 86]
Introspection in Knowledge Representation.
in [ECAI 86]

[Mandelbrot 83]
The Fractal Geometry of Nature.
New York: Freeman 1983.

[Markl 86]
Evolution und Freiheit. Das schöpferische Leben
in [Maier Leibnitz 86].

[Maier Leibnitz 86]
Zeugen des Wissens
v.Hase & Köhler, 1986.

[Michalski 84](ed.)
Machine Learning: An Artificial Intelligence Approach.
Tioga Publishing Company, 1983.
Springer Verlag 1984.

[Michalsky 86]
Machine Learning 2: An Artificial Intelligence Approach
Morgan Kaufman, Los Altos 1986.

[Minsky 69]
Perceptrons.
The MIT Press, Mass. Inst. of Technology 1969.

[Minsky 81]
Steps Towards Artificial Intelligence
in [Feigenbaum 81]

[Newell 83]
Endnotes to the Papers on Artificial Intelligence.
in [Machlup,Mansfield 83]

[Perlis 85]
Languages with Self-Reference 1: Foundations
(or: We can have everything in First-Order Logic!)
in Artificial Intelligence 25 (1985).

[Piaget 73]
Genetische Epistemologie (Einführung in die genetische Erkenntnistheorie)
s+w Suhrkamp 1973.

[Ritchie, Hanna 84]
AM: A Case Study in AI-Methodology
in Artificial Intelligence 23 (1984).

[Rumelhart, Zipper 85]
Feature Discovery by Competitive Learning
in Cognitive Science 9 (1985).

[Schulten ?]
Ordnung aus Chaos, Vernunft aus Zufall - Physik biologischer und digitaler Informationsverarbeitung.
Technische Universität München.

[Simon 69]
The Sciences of the Artificial
Cambridge, Mass.: MIT Press 1969.

[Steels 86]
KRS: Definition of Knowledge Representation Primitives.
ESPRIT PROJECT 440.

[Stolcke 87]
Implementierung einer selbstreferentiellen Repräsentationssprache in Prolog.
Arbeit zum Fortgeschrittenen-Praktikum
Technische Universität München 1987.

[Sussman 75]
A Computational Model of Skill Acquisition.
American Elsevier, 1975.

[WuvS 86]
Wahrnehmung und visuelles System.
Spektrum Reihe, 1986.

[Wallich ?]
Is AI the Next Logical Step in Data Processing?
in ?

[Weiss 77]
(System) Das lebende System: Ein Beispiel für den Schichtendeterminismus.
in Das neue Menschenbild, edd. Koestler, Smythies, Wien 1978.

[Weizsäcker 85]
Der Aufbau der Physik
Carl Hanser Verlag, Münche, Wien, 1985.

[Weizsäcker, E+C 72]
Wiederaufnahme der begrifflichen Frage: Was ist Information?
Nova Acta Leopoldina 206

[Westerdale 85]
The Bucket Brigade is not Genetic.
in [Grefenstette 85].

[Winograd 75]
Frame Representations and the Declarative–Procedural Controversy.
in [Bobrow, Collins 75].

[Winston 81]
Artificial Intelligence
Second edition
Addison Wesley, 1981.

Table of Contents