Athens Journal of Sciences 2024, 11: 1-18
https://doi.org/10.30958/ajs.X-Y-Z

A General Model for Representing Knowledge - Intelligent Systems Using Concepts

By Thomas Fehlmann[*] & Eberhard Kranich[±]

Arrow terms of the form $\{a_1, a_2, \ldots, a_n\} \rightarrow b$ *have been proposed by Scott and Engeler, 50 years ago, as a model for combinatory logic. Since combinatory logic is Turing-complete, the model causes interest for domains dealing with knowledge, such as artificial intelligence. It has been used to model neural networks – how does the brain think – and connect the notion of computability with observability in natural science. In software testing, arrow terms serve as representation for test cases that allow combination, and thus automated generation of more test cases for testing complex systems.*
However, knowledge is not a well-defined notion. It is sometimes referred to as awareness of facts or as practical skills and may also refer to familiarity with objects or situations. Knowledge of facts is distinct from opinion or guesswork by virtue of justification. Facts are usually described as a set of conditions, followed by a consequence. This makes the arrow term model an accurate model for knowledge. Knowledge about facts can lead to theories. A theory is knowledge under control. Theories can be approximated by control sequences. The combination of knowledge and theories makes up for intelligent systems.

Keywords: *combinatory logic, arrow term models, artificial intelligence, learning systems, intelligent systems, intuitionism*

Introduction

Paper Intent

In the early 20th century, there were some shocking events taking place in mathematical logic and natural science. Gödel (1931), when trying to solve some of Hilbert's 23 problems, detected that predicate logic, something with a very long history dating back to the ancient Greeks, see Engeler (2020), is undecidable. This insight gave birth to theoretical computer science, including the theory of computation, founded by Turing (1937). For a modern compilation, see e.g., Raatikainen (2020).
Schönfinkel and Curry developed *Combinatory Logic* (Curry & Feys 1958) to avoid the problems introduced when using logical quantifiers, and Church invented *Lambda Calculus* as a rival formalism (Church 1941). Scott and Engeler developed the *Graph Model* (Engeler 1981), based on *Arrow Terms*, and proved that this is a model of combinatory logic. This means that you can combine sets of

[*]Senior Research, Euro Project Office AG, Switzerland.
[±]Senior Research, Euro Project Office AG, Switzerland.

arrow terms to get new arrow terms, and that combinators, accelerators, and constructors can be used to create new elements of algebra.

Graphs in the form of neural networks appeared already at the origins of *Artificial Intelligence* (AI). Its first instantiation in modern times was the *Perceptron*, a network of neurons postulated by Rosenblatt (Rosenblatt 1957). It later became a directed graph (Minsky & Papert 1972). Rosenblatt was also the first postulating concepts, among perception and recognition, as constituent parts of AI (Rosenblatt 1957, p. 1).

Since its origins, AI has experienced difficulties; however, today it seems to have become mainstream as far as there are many AI applications that provide value for the user. In some areas, training an AI model is much simpler and more rewarding than finding and programming an algorithm (Nico Klingler (viso.ai) 2024).

Nevertheless, temporal patterns (Rosenblatt 1957, p. 2) still are not available in AI and provide quite a challenge, as exemplified by the ARC challenge, a sort of intelligence test for AI models, proposed by Chollet (2019).

The Graph Algebra of Arrow Terms

Let $\mathcal{L}$ be a non-empty set. Engeler (1981) defined a *Graph* as the set of ordered pairs:

$$\langle\{b_1, b_2, \dots, b_m\}, c\rangle \tag{1}$$

with $b_1, b_2, \dots, b_m, c \in \mathcal{L}$. We prefer to write $\{a_1, \dots, a_m\} \to b$ instead of the ordered pair to make notation mnemonic and call them *Arrow Terms*. These terms describe the constituent elements of directed graphs with multiple origins and a single node. We extend the definition of arrow terms to include all formal set-theoretic objects recursively defined as follows:

$$\begin{gathered}\text{Every element of } \mathcal{L} \text{ is an arrow term.}\\ \text{Let } a_1, \dots, a_m, b \text{ be arrow terms.}\\ \text{Then } \{a_1, \dots, a_m\} \to b \text{ is also an arrow term.}\end{gathered} \tag{2}$$

The left-hand side of an arrow term is a finite set of arrow terms, and the right-hand side is a single arrow term. This definition is recursive. Elements of $\mathcal{L}$ are also arrow terms. The arrow, where present, should suggest the ordering in a graph, not logical imply.

The Algebra of Observations

Let now $\mathcal{L}$ be more specific, namely the non-empty set of assertions over certain objects of the real world, observable and recognizable by suitable AI models, in some formal language about that domain. Examples include statements about gravity, temperature, matter, molecules, or any object that can be tagged using AI (Nico Klingler (viso.ai) 2024).

Denote by $\mathcal{G}(\mathcal{L})$ the power set containing all arrow terms of the form (2). The formal definition in set-theoretical language is given in equation (3) and (4):

$$\begin{gathered}\mathcal{G}_0(\mathcal{L}) = \mathcal{L} \\ \mathcal{G}_{n+1}(\mathcal{L}) = \mathcal{G}_n(\mathcal{L}) \cup \left\{\{a_1, \dots, a_m\} \to b \middle| a_1, \dots, a_m, b \in G_n(L), m \in \mathbb{N}\right\} \\ \text{for } n = 0, 1, 2, \dots\end{gathered} \tag{3}$$

The definition is recursive. Thus, $\mathcal{G}(\mathcal{L})$ is the set of all (finite and infinite) subsets of the union of all $\mathcal{G}_n(\mathcal{L})$:

$$\mathcal{G}(\mathcal{L}) = \bigcup_{n \in \mathbb{N}} \mathcal{G}_n(\mathcal{L}) \tag{4}$$

The elements of $\mathcal{G}_n(\mathcal{L})$ are arrow terms of level n. Terms of level 0 are named *Observations*, a finite set of arrow terms of level 1 or higher is called *Concept*; finite or infinite sets of arrow terms, including observations and concepts, are called *Knowledge*.

The definition of an application between two finite or infinite sets of arrow terms M, N – observations, concepts, and knowledge equally – makes $\mathcal{G}(\mathcal{L})$ an algebra:

$$M \bullet N = \{c | \exists \{b_1, b_2, \dots, b_m\} \to c \in M;\ b_i \in N\} \tag{5}$$

$\mathcal{G}(\mathcal{L})$ is closed under the application operation.

According to Engeler (Engeler, 1981), the motivation behind this definition is, when starting with observations about some domain, arrow terms represent knowledge about that domain. Examples include the *Neural Algebra* of Engeler (2019). Thus, it might be of interest to engineers who want to handle knowledge, and in fact, AI was the ultimate vision at the time the graph model was conceived (Engeler 1995).

Why the Left Hand of a Concept Must be a Set, Not a Conjunction

The use of a set for the left-hand side of an element of a concept is essential; if one tries with a conjunction, one gets a model for typed lambda calculus, see Bimbó (2012). Such models may still have interesting properties; in some respect, they produce deterministic outcomes and are easy to prove for correctness; however, they are not relevant for AI.

Today's world of intelligent and smart products is far from producing deterministic outcomes, and some products have cyber-physical effects on their environment that might harm people; thus, are safety relevant. Examples include autonomous vehicles and medical instruments but are today growing rapidly.

The difference between conjunctions and sets is that sets might contain contradictory elements, annihilated by conjunctions. For software testing, especially for testing smart systems using AI und *Deep Learning*, it is of essence. In the fa-

mous case of the Tempe crash (March 18, 2018), the crash report shows that the scenario recognition system of the autonomous car on its supervised trial run had seven seconds to react – sufficient for an emergency break – but recognized three different objects: a person, a bike, and plastic bags. These perceptions are mutually excluding, but you would never allow an autonomous car emergency breaking because of a plastic bag, but exactly such different perceptions happen in the real world. Intelligent things as well as humans experience this. A test bench must therefore never require test data to be consistent if it's required to test safety relevant features, such as whether an autonomous car starts the risky action of an emergency stop.

Intelligent Systems

There is currently a lot of talk about AI, including *Large Language Models* (LLM), which have the potential to change the way we work. AI systems have a stunning ability to collect information, learn from them, and deliver responses to related questions, without programming (Wolfram 2023). However, how do observations connect to the real world? Referring to objects that exhibit some usual behavior?

This process is called *Grounding* (Zhong, et al. 2022). Every human knows that gravity prevents heavy objects from flying, something that is easily expressed by a concept, but today's AI can just collect observations about heavy objects and derive a model out of this. Quite a tedious process. It needs a huge bunch of samples until it "knows" about the effect of gravity. By grounding an observation to a known object of the real world, the intelligent system should know.

The kind of knowledge that allows grounding, and much more, are *Concepts*. We will show how to implement concepts in Engeler neural algebra.

Research Questions

In this paper, we discuss several open points:

1. What is a suitable theoretical foundation of AI?
2. Does handling knowledge with concepts enable intelligent systems (AI) to learn faster and become more predictable?
3. How should intelligent systems evolve to incorporate the arrow term algebra into their behavior?
4. Will intelligent systems ever be able to solve unfamiliar problems on their own?

Paper Content

We start with an introduction to Combinatory Logic and explain why this logical construct is of interest both to mathematicians who are looking at the foundations of their science, as well as to engineers who want to build intelligent products.

Next, we introduce Arrow Terms as a model for combinatory logic and explain what it has to do with intuitionism. We explain why arrow terms serve as a generalization of knowledge and how concepts might be used to solve problems.

Then we explore the possibility of programming concepts. The intelligent system might combine such concepts, to the possibility of creating new concepts out of combining them, based on observations by the machine itself.

Finally, we outline a few conclusions and answer the research question.

Combinatory Logic

Combinatory Logic and the Axiom of Choice

Combinatory Logic is a notation to eliminate the need for quantified variables in mathematical logic, and thus the need to explain what means $\exists x \in M$, "there exists some x in some set M", see Curry and Feys (1958) and Curry et al. (1972). Eliminating quantifiers is an elegant way to avoid the *Axiom of Choice* (Fehlmann & Kranich 2020) in its traditional form. Combinatory Logic can be used as a theoretical model for computation and as design for functional languages (Engeler, 1995); however, the original motivation for combinatory logic was to better understand the role of quantifiers in mathematical logic.

It is based on *Combinators* which were introduced by Schönfinkel in 1920. A combinator is a higher-order function that uses only functional application, and earlier defined combinators, to define a result from its arguments.

The combination operation is denoted as $M \bullet N$ for all combinatory terms M, N. To make sure there are at least two combinatory terms, we postulate the existence of two special combinators **S** and **K**. They are characterized by the following two properties (6) and (7):

$$\mathbf{K} \bullet P \bullet Q = P \tag{6}$$

$$\mathbf{S} \bullet P \bullet Q \bullet R = P \bullet Q \bullet (P \bullet R) \tag{7}$$

where P, Q, R are terms in combinatory logic[1]. The combinator **K** acts as projection, and **S** is a substitution operator for combinatory terms. Equations (6) and (7) act like axioms in traditional mathematical logic.

Like an assembly language for computers, or a Turing machine, the **S**-**K** terms become quite lengthy and are barely readable by humans, but they work fine as a foundation for computer science.

The power of these two operators is best understood when we use them to define other, handier, and more understandable combinators: The identity combinator for instance is defined as

$$\mathbf{I} \coloneqq \mathbf{S} \bullet \mathbf{K} \bullet \mathbf{K} \tag{8}$$

---

[1]The use of variables named P, Q, R is borrowed from Engeler (2020).

Indeed, $\mathbf{I} \bullet M = \mathbf{S} \bullet \mathbf{K} \bullet \mathbf{K} \bullet M = \mathbf{K} \bullet M \bullet (\mathbf{K} \bullet M) = M$. Association is to the left.

Moreover, **S** and **K** are sufficient to build a Turing-machine. Thus, combinatory logic is Turing-complete. For a modern proof, consult Barendregt (Barendregt & Barendsen 2000, pp. 17-22).

Functionality by the Lambda Combinator

Curry's *Lambda Calculus* (Barendregt 1977) is a formal language that can be understood as a prototype programming language. The **S**-**K** terms implement the lambda calculus by recursively defining the *Lambda Combinator* $\mathbf{L_x}$ for a variable x as follows:

$$\begin{gathered} \mathbf{L_x} \bullet x = \mathbf{I} \\ \mathbf{L_x} \bullet Y = \mathbf{K} \bullet Y \text{ if } Y \text{ different from } \mathbf{x} \\ \mathbf{L_x} \bullet M \bullet N = \mathbf{S} \bullet \mathbf{L_x} \bullet M \bullet \mathbf{L_x} \bullet N \end{gathered} \tag{9}$$

The definition holds for any term **x** of combinatory logic. Usually, on writes suggestively $\lambda x.M$ instead of $\mathbf{L_x} \bullet M$, for any combinatory term M. Note that $\lambda x.M$ is a combinatory term, as proofed by (9), and that we now introduced some sort of variable into combinatory logic with a precise binding behavior.

The Lambda combinator allows writing programs in combinatory logic using a higher-level language. When a Lambda term gets compiled, the resulting combinatory term is like machine code for traditional programming languages.

The Fixpoint Combinator

Given any combinatory term Z, the *Fixpoint Combinator* **Y** generates a combinatory term $\mathbf{Y} \bullet Z$, called *Fixpoint of* Z**,** that fulfills $\mathbf{Y} \bullet Z = Z \bullet (\mathbf{Y} \bullet Z)$. This means that Z can be applied to its fixpoint as many times as wanted and still yields back the same combinatory term.

In linear algebra, such fixpoint combinators yield an eigenvector solution to some problem Z; for instance, when solving a matrix in linear algebra (Fehlmann 2016). It is therefore tempting to say, that $\mathbf{Y} \bullet Z$ is a solution for the problem Z.

According to Barendregt and Barendsen (2000, p. 12), the fixpoint combinator can be written as

$$\mathbf{Y} := \lambda f.\big(\lambda x.f \bullet (x \bullet x)\big) \bullet \big(\lambda x.f \bullet (x \bullet x)\big) \tag{10}$$

Translating (10) into an **S**-**K** term proves possible but becomes a bit lengthy. It demonstrates how combinatory logic works; consult Fehlmann & Kranich (2022). For more sample combinators, consult Zachos (1978).

However, the fixpoint combinator is not the solution to all our problems. When applying **Y**, or any other equivalent fixpoint combinator to a combinatory term Z, reducing the term by repeatedly using rule (6) and (7) does not always terminate. An infinite loop can occur, and must sometimes occur, otherwise Tu-

ring would be wrong and all finite state machines would reach a finishing state (Turing 1937).

The Graph Model of Combinatory Logic

Why a Model?

A *Model* for a logical structure is a set-theoretic construction that has the properties postulated for the logic and can be proved to be non-empty. Then it means that logic makes sense as far as it describes some structure that really exists.

Einstein-Notation for Arrow Terms

To avoid the many set-theoretical parenthesis, the following notation, called *Arrow Schemes*, is applied, in analogy to the Einstein notation (Fehlmann 2020, p. 6):

- a_i for a finite set of arrow terms, i denoting some *Choice Function* selecting finitely many specific terms out of a set of arrow terms a.
- a_1 for a singleton set of arrow terms; i.e., $a_1 = \{a\}$ where a is an arrow term. (11)
- $\emptyset$ for the empty set, such as in the arrow term $\emptyset \to a$.
- $a_i + b_j$ for the union of two sets a_i and b_j of observations.

The application rule for M and N now reads:

$$M \bullet N = (b_i \to a) \bullet N = \{a | \exists b_i \to a \in M;\ b_i \subset N\} \quad (12)$$

where $(b_i \to a) \subset M$ is the subset of level 1 arrow terms in M. With these conventions, $(x_i \to y)_j$ denotes a *Concept*, i.e., a non-empty finite set of arrow terms with level 1 or higher, together with two choice functions i, j. Each set element has at least one arrow.

The choice function chooses some specific observations a_i out of a (larger) set of observations a. This is what Zhong describes as *grounding* when linking observations to real-world objects (Zhong et al. 2022). If a denotes knowledge, i.e., an infinite set of arrow terms of any level, a_i can become part of a concept consisting of specific arrow terms referring to some specific sample knowledge, specified by the choice function j. Choice functions therefore have the power of focusing knowledge on specific objects in specific areas. That makes choice functions interesting for intelligent systems and AI.

There is a conjunction of choice functions, thus $a_{i,j}$ denotes the union of a finite number of m concepts:

$$a_{i,j} = a_{i,1} \cup a_{i,2} \cup \dots \cup a_{i,j} \cup \dots \cup a_{i,m} = \bigcup_{k=1}^{m} a_{i,k} \quad (13)$$

There is also cascading of choice functions. Let $N = \left(a_j \to x\right)_k$ then:

$$M = \left(\left(\left(a_j \to x\right)_k \to x_i\right)_l \to y\right) \text{ and}$$
$$M \bullet N = \left(x_{i_l} \to y\right) \quad (14)$$

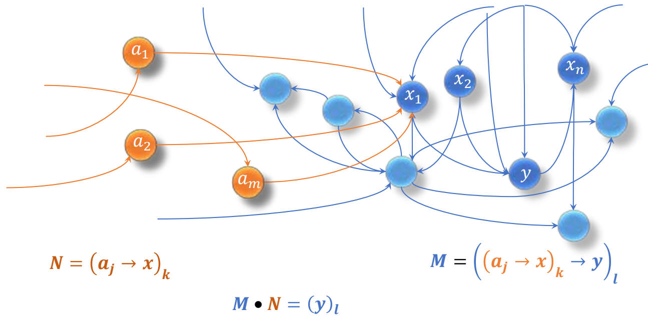
The choice function might be used for grounding a concept to observations.

An arrow scheme without outer indices represents a potentially infinite set of arrow terms. Thus, writing a, we mean knowledge about an observed object. Adding an index, a_j, indicates such a grounded object together with a choice function j that chooses finitely many specific observations or knowledge.

While on the first glimpse, the Einstein notation seems just another way of denoting arrow terms, for representing such data in computers it means that the simple enumeration of finite data sets is replaced by an intelligent choice function providing grounding that must be computed and can be either programmed or guessed by an intelligent system.

Interestingly, as the name “graph model” suggests, arrow terms are just an algebraic way of describing neural networks. Thus, something that nature uses to acquire and work with knowledge.

Figure 1. *A Neural Network as a Combinatorial Algebra*



Combination according to equation (12) yields what we perceive as “response from a neural network.” Thus, combinators play a significant role in AI.

The Graph Model – A Model of Combinatory Logic

The algebra of observations represented as arrow terms is a combinatory algebra and thus a model of combinatory logic. The following definitions demonstrate how observations implement the combinators **S** and **K** fulfilling equations (6) and (7).

- $\mathbf{I} = a_1 \to a$ is the Identification, i.e., $(a_1 \to a) \bullet b = b$
- $\mathbf{K} = a_1 \to \emptyset \to a$ selects the 1st argument: (15)
  $\mathbf{K} \bullet b \bullet \mathrm{c} = (b_1 \to \emptyset \to b) \bullet b \bullet \mathrm{c} = (\emptyset \to b) \bullet \mathrm{c} = \mathrm{b}$

- $\mathbf{KI} = \emptyset \to a_1 \to a$ selects the 2nd argument:
  $\mathbf{KI} \bullet b \bullet \mathrm{c} = (\emptyset \to c_1 \to c) \bullet b \bullet \mathrm{c} = (c_1 \to c) \bullet c = \mathrm{c}$
- $\mathbf{S} = \left(a_i \to \left(b_j \to c\right)\right)_1 \to (d_k \to b)_i \to \left(a_i + b_{j,i} \to c\right)$

Therefore, the algebra of observations is a model of combinatory logic. The interested reader can find complete proofs in Engeler (Engeler 1981, p. 389).

With $\mathbf{S}$ and **K,** an abstraction operator can be constructed that builds new knowledge bases, following equation (9) (9). In the setting of a combinatory algebra, this is called *Lambda Theorem*; see Barendregt (Barendregt 1977).

Instances of the Graph Model

Number Theory

The assertions $\mathcal{L}$ might be numerical formulas. Then $\mathcal{G}(\mathcal{L})$ is the knowledge about number theory that can be combined and processed using the lambda calculus, or solved by fixpoint combinators, or controlled by some controlling combinators. The latter is better known as infinite series, with or without convergence.

Graph Theory

Obviously, the graph model takes its name from directed graphs, where each arrow term represents a node with its predecessors. Neural networks have such a structure. With the graph model, neural networks become properties of an algebra, see equations (6) and (7). For AI, this is interesting because it explains how to implement concepts.

Quality Function Deployment (QFD)

QFD can also be represented by the graph model; the arrow terms become then Ishikawa-diagrams. Sets of arrow terms, or Ishikawa-diagrams, can be represented by matrices. They were indeed at the roots of QFD (Fehlmann 2016, p. 321). QFD is less versatile than AI because the matrices have limited size and learning is by capturing experts' knowledge, not training by big data, but otherwise there are many similarities between QFD and AI.

Neural Network

Engeler used the arrow term model to explain how the brain thinks. The Graph Model of Combinatory Logic explains how complex scripts of behavior and conceptual content can reside in, combine, and interact on large neural networks. The neural hypothesis attributes functions of the brain to sets of firing neurons dynamically: to cascades of such firings, typically visualized by imaging technologies.

Such sets are represented as the elements of what Engeler calls a *Neural Algebra* with their interaction as its basic operation. The neuro-algebraic thesis identifies "thoughts" with elements of a neural algebra and "thinking" with its basic operation (Engeler 2019).

Engeler argues for this thesis by a thought-experiment. It examines examples of human thought processes in proposed emulation by neural algebras. Problems

such as controlling, classifying, and learning are analyzed. In neural algebras these may be posed as algebraic equations, whose solutions may lead to extensions of a neural algebra by new elements. Modelling of such extensions consists of formal analogues to familiar faculties such as reflection, distinction and comprehension which can be made precise as operations on the algebra. Barendregt gives handy examples of such precise algebra operations in the chapter about "The Power of Lambda Calculus" in his seminal book (Barendregt 1984, pp. 17-22).

An advantage of such an approach is that this modelling leads directly to brain functions. From the cascades of such functions, we obtain the neurons involved in them, and their connective structure, and can mathematically describe their behavior using the graph model.

Autonomous Real-time Testing

Fehlmann (2020) proposes the arrow term model for automatically creating test cases for complex systems. Instead of observations, the domain $\mathcal{L}$ consists of assertions about the status of the program under test, and the level 1 arrow terms describe test cases. Sets of test cases can be combined as any arrow term sets and allow developing new test cases automatically.

Controlling operators, see next section, are usually quite simple: in most cases, it is sufficient to combine test cases on the lowest possible level, i.e., on $\mathcal{G}_1(\mathcal{L})$, and connect them with their value for the user. This is especially useful when testing large, complex systems of systems ensuring correct cooperation between different component systems.

Higher-level controlling operators are conceivable and might become of essence for testing intelligent systems. Testing concepts, if reused, have the potential to become the most important asset of companies providing intelligent products.

Controlling Combinators

The Need for a Temporal Extension in AI

AI-powered Visual Recognition Systems excel in recognizing and classifying objects. However, they are weak at recognizing temporal dependencies and unable to combine learnings. AI lacks what humans use in such cases: a concept.

Controlling Combinators for Solving the Control Problem

The concept of *Control* involves a *Controlling Operator* $\mathbf{C}$ which acts on a controlled object X by application $\mathbf{C} \bullet X$. Control means that the knowledge represented by X is completely known and described. It is a similar approach to establishing a fixpoint.

Accomplishing control can be formulated by:

$$\mathbf{C} \bullet X = X \tag{16}$$

The equation (16) is a theoretical statement, usually an infinite loop process.

For solving practical problems, X must be approximated by finite subterms.

The control problem is solved by a *Control Sequence* $X_0 \subseteq X_1 \subseteq X_2 \subseteq \cdots$, a series of finite subterms, determined by (17):

$$X_{i+1} = \mathbf{C} \bullet X_i, i \in \mathbb{N} \tag{17}$$

starting with an initial X_0. This is called *Focusing*. The details can be found in Engeler (2019, p. 299). The controlling operator $\mathbf{C}$ gathers all faculties that may help in the solution. Like equation (10), controlling operators consist of structural content than of single observations. The control problem is a repeated process of substitution, like finding the fixpoint of a combinator.

Within this setting, it is possible to define models for reasoning (Engeler 2019), problem solving, and software and systems testing (Fehlmann & Kranich 2019).

Use the Choice Function for Grounding

A *Concept of Level 1* consists of a set of arrow terms of the form (18)

$$x_j \to y \tag{18}$$

where x denotes observations. The choice function j selects specific observations for some real-world object. With Zhong et al. (2022), we use the term "grounding" for such choice functions. The response y describes the expected properties of the observations x. The choice function j appears again in y, specifying expected behavior or properties that are consequences from the observations x_j. With such a concept, we can describe behavior of real-world objects that are otherwise hard for an intelligent system to guess.

The choice function j is computable and constructive, referring to the properties of the base elements of the set of observations x. According to the Lambda theorem (9), it can be programmed. The concept based on choice functions grounds arrow schemes to basic observations.

The knowledge acquired by new observations growths:

$$X_0 \subseteq X_1 \subseteq X_2 \subseteq \cdots \tag{19}$$

Thus, the index creates a control sequence (19), formally in the same way as Engeler explains how the brain thinks (Engeler 2019, p. 301).

Programming the Choice Function

The more concepts an intelligent system has at its disposal, the faster it will solve problems, and the more reliable and predictable will its decisions be. It is therefore a clever idea to write domain-specific concepts beforehand. Neither for

humans nor for intelligent systems, training the choice function is a simple task. However, because we use combinatory algebra, concepts can be combined. For instance, we might combine a concept looking for pixels with a specific color logo with another exploring specific positioning between the pixels of interest.

Concepts thus seem an approach to add a sort of external programmability to intelligent machines that blast the existing paradigm of "Learning" in AI. We no longer are restricted to training neural networks and creating models – in fact, by multilinear optimization – but add the capabilities of modern algorithmic programming. The arrow terms "assembly language" is the layer where combination occurs. On this layer, concepts can combine with observations and increase knowledge. Higher level concepts might add new power to concepts. Thus, this serves as a strong motivation to continue with the explanation of what a "concept" means in the context of the graph model.

Focusing Using Attractors

The control problem is a repeated process of substitution, like finding the fixpoint of a combinator. Within this setting, it is possible to define models for reasoning, problem solving, and scientific observations. However, not only flat reasoning, but also for solving problems, even if their fixpoint is infinite.

Let X be an expandable, unorganized set of observations. Apply the controlling combinator $\mathbf{C}$ to X with the aim to accomplish control, see equation (16). Then, solutions are obtained by *Focusing*:

$$X_{i+1} = \mathbf{C} \bullet X_i,\ i \in \mathbb{N} \tag{20}$$

Starting with initial evidence X_0, the controlling combinator $\mathbf{C}$ creates the control sequence $X_0 \subseteq X_1 \subseteq X_2 \subseteq \cdots$ towards an optimum solution containing all elements of the control sequence, eventually reaching X. This optimum solution is called an *Attractor*. The details can be found in Engeler (2019, p. 301).

In natural science, it is tempting to call such a controlling combinator a *Theory*, since the control sequence predicts evidence.

The concept $\mathbf{C}$ gathers all faculties that may help in finding the solution of a problem. Using the Lambda theorem as of equation (9), concepts consist of structural content than of single observations. The control problem is a repeated process of substitution, like finding the fixpoint of a combinator. Nevertheless, concepts may relate to the domain $\mathcal{L}$ by choice functions $\boldsymbol{j}$. Thus, it is possible to write concepts that meet safety or security requirements. Or, concepts can be used to implement morally aware behavior, for instance avoiding racial or gender bias, even if the AI training set was not perfect. Concepts thus can address societal reservations about AI by controlling grounding references made in the knowledge used for decisions. Concepts can make AI decisions transparent to humans.

Examples of concepts include combinators that extract only a part of knowledge, like $\boldsymbol{K}$ and $\boldsymbol{KI}$, or conditional branching, or conjunction and disjunction. Some concepts repeat actions until a certain condition holds and may repeat

actions potentially forever. A concept implements mechanisms known from programming but with references to the grounded objects.

Obviously, it is much easier to create such a conceptually defined controlling combinator by programming techniques, programming the choice function j using the Lambda Theorem (9). Programmers can create controlling combinators that select the objects with suitable properties and give these to the machine as concepts. The intelligent system alone would encounter major difficulties to guess the right indexing choice functions, without help by a programmer.

A Classification of Concepts ("Controlling Combinators")

Concepts exist on various levels:

Indexing Controls describe concepts C that apply the same choice function to different but similar knowledge. Such concepts allow grounding objects of the real world to observations and make learning for intelligent systems much faster. Indexing Controls can be programmed easily and used to teach an intelligent system what gravity is, or how children behave when playing with a football. Such knowledge is easily programmable by choice functions.

Most indexing control concepts remain on level 1 and are easy to handle because humans still can understand what concepts do.

Holding Controls are concepts $\mathbf{C}$ that stop adding more knowledge at a certain point after n steps:

$$\begin{gathered} X_0 \subseteq X_1 \subseteq X_2 \subseteq \ldots \subseteq X_{\mathrm{n}} = X_{\mathrm{n}+1} = \ldots \text{ for } X_{i+1} = \mathbf{C} \bullet X_i \\ \text{where } X_0 \subseteq X_1 \subseteq X_2 \subseteq \ldots \text{is a control sequence} \end{gathered} \tag{21}$$

An example is a finite list of facts $\mathbf{C} \bullet X = X$. Because of the hold at a certain point in time, this is still understandable by humans, and they can follow its reasoning.

Attractor Controls are concepts C that continuously add more knowledge, without limit. An example is the famous fixpoint $\boldsymbol{Y} \bullet Z$ that yields $\mathbf{Y} \bullet Z = Z \bullet (\mathbf{Y} \bullet Z)$ for any given combinator Z.

This is a difficult approach because attractor concepts conceive a higher kind of intelligence than humans usually can exhibit. Human controlling sequences are limited anyway because of the finite lifespan of humans, see Engeler's sample "Neural Model-Mathematician" (Engeler 2019, p. 306). Attractor control concepts are relentlessly adding knowledge from a theory. This iterates that as long as needed, until a stage of knowledge is reached that meets certain predefined quality criteria.

The Problem with Concepts

While the theory looks appalling and easy, and programming in Lambda calculus familiar, transforming a Lambda term into a combinator is tedious, and into a set of arrow terms is something only a machine can do flawlessly. That might be exactly the reason combination of concepts hasn't been studied earlier. Combining

arrow term sets simply becomes too complicated. But engineers are frightened by the prospect of computational complexity, as long as it remains controlled.

While programming concepts in Lambda calculus seems not more complicated than writing symbolic programs in Lisp or Scala, transforming Lambda terms into S-K terms, and even more into sets of arrow terms, is far from intuitive and requires help by machines ("Rewrite rules").

Moreover, we can combine concepts not only by functional application ("Currying"), but also by help of higher-level programming concepts such as disjunction, conjunction, or conditional branching, as well as unconditional loops ("For-loops") or even conditional loops that potentially never stop iterations. All these programming concepts can be expressed in combinatory logic and thus might become instrumental for combining concepts for AI systems.

A Roadmap towards Introducing Concepts in Artificial Intelligence

We summarize the arguments given as follows: we have observations made by AI and we use concepts to teach the intelligent system what to do with those observations. It is true that such concepts can be acquired by AI without human help, but this is a tedious and lengthy task. It is easier for a machine to choose concepts from a collection of already existing concepts – which might prove to be successful – than to being trained in them every time from scratch. An initial set of concepts is what human programmers can furnish to an intelligent machine. However, a system can only then be called "intelligent" if it can combine concepts without help from a human.

Joining Forces: Traditional Functional Programming Supports Concepts.

Machines learn to recognize and correctly identify objects, both logically and visually, using traditional AI methods, modeling the real world by multilinear optimization. But only concepts enable them to create new problem solutions.

Concepts can be programmed using the DevOps paradigm. An intelligent system needs a minimum selection of concepts to solve real-world problems. In turn, an intelligent system can combine various concepts and select the most promising ones based on *Cost Functions* that reflect relevance for the user.

This is analogue to the invention of flying machines that initially were thought to imitate bird's light, until aerodynamic uplift was better understood. By training traditional models AI will not be able to develop controlling combinators, or concepts, adopting the way brains recognize patterns. Engeler calls for a "Teaching Combinator" that is needed to help brains to develop such concepts. It looks like Panigrahi et.al. have detected a similar capability of large neural networks that they call "Skills" in exceptionally large LLM (Panigrahi et al., 2023).

Using traditional programming enables machines to learn concepts. Programming concepts is possible thanks to Barendregt's Lambda theorem (9). This does not mean that machines cannot become creative and learn to combine concepts themselves - much like humans do. Concepts are elements of combinatory

algebra, always allowing for combination. The reader should note once more how important it is to use type-free programming.

The problem with combining concepts is the same as with any deep learning approach: a cost function must be found that allows to choose valuable combinations out of the variety of combinations of concepts. Machines are good at simulation, better than humans. It should not be difficult to define such cost functions based on simulation of event outcomes.

From Creating Concepts to Empathetic, Intelligent Systems

Concepts add dynamic, algorithmic, and temporal intelligence to deep learning, that in turn is static, statistical learning. Programmers tell robots what to do with their static insights, using concepts. In turn, robots can select concepts themselves if found useful. This is a new hybrid humanoid: Deep learning plus algorithms combined.

In a competitive environment, good concepts will become decisive for deciding what intelligent system to acquire, and thus deciding about commercial success or failure of intelligent systems.

Conclusions

Intuitionism and Choice Functions

The Graph Model is an extremely rich structure for representing quite different topics such as

- How does the brain think.
- Product improvement with focus on customer needs by QFD.
- Testing of complex, software-intense systems with thousands of Embedded Control Units (ECU).
- Making AI intelligent.

Choice functions offer a constructive way to ground knowledge; existence of a choice always means existence of an algorithm that does the choice, as suggested by Intuitionism (Fehlmann & Kranich 2020). This is counter-intuitive to human perception of the world but reflects the standpoint of mathematical logic (Fehlmann & Kranich 2020). It enables intelligent systems to behave truly reasonable and rationally.

Research Questions

Regarding question 1, we proposed a theoretical foundation for AI which is not new but rather based on ideas from Rosenblatt, Scott, and Engeler.

With concepts, we can build intelligent systems that behave predictably when able to apply the right concept. Testing such systems, or relying on such systems,

is easier than with traditional model-based AI. You not only can do black-box testing but, using concepts, you can even perform white-box tests. This is a big advantage if intelligent systems should become accountable for social and environmental well-being. The answer to research question 2 is affirmative.

Regarding question 3, how should intelligent systems evolve, we answer by referring to DevOps as the development methodology of choice to enable intelligent systems.

The answer to research question 4, will intelligent systems ever be able to solve unfamiliar problems on their own, must remain open. Although Engeler (2019) describes a class of combinators in his neural algebra that accomplish such creative work, the practical proof is yet open.

Also, recent experiences with AI (Panigrahi et al. 2023) suggest a positive answer, at least for LLM that are big enough, but more work in the direction of Turing (1937) and Chollet (2019), who devised some sort of intelligence quotient test for AI, is needed.

What can be concluded, is that adapting the von Neumann principle to AI, representing knowledge by combinators and use a common framework, the arrow terms, to represent both knowledge and programs, yields a wide range of possibilities and new opportunities (Copeland 2006).

Open Questions

How to program concepts exactly?

- Lisp, Scala, others?
- How to link program code to observed objects (grounding)?

What is the cost function for combining concepts?

- Functional size?
- Reliability of the AI machine learning process?

Will AI machines eventually program themselves?

- Is DevOps just a temporary solution?
- Will DevOps become unnecessary for AI?
- Do fixpoints help focusing (Fehlmann & Kranich 2022)?

How to protect the freedom of citizens against AI?

- What about Security & Privacy?
- How to test concepts for compliance?

Acknowledgments

The authors would like to thank Hansruedi Jud from Lab42 in Davos, Switzerland, for his contributions, ingenious ideas, and critical comments regarding the ARC challenge, and the anonymous referees for their comments and suggestions which led to an improvement of the paper.

References

Barendregt HP (1977) The Type-Free Lambda-Calculus. In J Barwise (ed.), *Handbook of Math. Logic,* 1091–1132. Amsterdam: North Holland.

Barendregt HP (1984) *The Lambda Calculus – Its Syntax and Semantics.* Studies in logic and the foundations of mathematics Hrsg. Amsterdam: North-Holland.

Barendregt H, Barendsen E (2000) *Introduction to Lambda Calculus.* Nijmegen: University Nijmegen.

Bimbó K (2012) *Combinatory Logic - Pure, Applied and Typed.* Boca Raton, FL: CRC Press.

Chollet F (2019) *On the Measure of Intelligence.* [Online] Available at: https://doi.org/10.48550/arXiv.1911.01547

Church A (1941) The Calculi Of Lambda Conversion. *Annals of Mathematical Studies 6*.

Copeland J (2006) *The Modern History of Computing,* Stanford, CA: Stanford Encyclopedia of Philosophy.

Curry H, Feys R (1958) *Combinatory Logic, Vol. I.* Amsterdam: North-Holland.

Curry H, Hindley J, Seldin J (1972) *Combinatory Logic, Vol. II.* Amsterdam: North-Holland.

Engeler E (1981) Algebras and Combinators. *Algebra Universalis,* Band 13, pp. 389-392.

Engeler E (1995) *The Combinatory Programme.* Basel, Switzerland: Birkhäuser.

Engeler E (2019) Neural algebra on "how does the brain think?". *Theoretical Computer Science* 777: 296–307.

Engeler E (2020) Aristotle' Relations: An Interpretation in Combinatory Logic. *arXiv: History and Overview.*

Fehlmann TM (2016) *Managing Complexity - Uncover the Mysteries with Six Sigma Transfer Functions.* Berlin, Germany: Logos Press.

Fehlmann TM (2020) *Autonomous Real-time Testing – Testing Artificial Intelligence and Other Complex Systems.* Berlin, Germany: Logos Press.

Fehlmann TM, Kranich E (2019) Testing Artificial Intelligence by Customers' Needs. *Athens Journal of Sciences* 6(4): 265–286.

Fehlmann TM, Kranich E (2020) Intuitionism and Computer Science – Why Computer Scientists do not Like the Axiom of Choice. *Athens Journal of Sciences* 7(3): 143–158.

Fehlmann TM, Kranich E (2022) *Designing and Testing Cyber-Physical Products - 4th Generation Product Management Based on AHP and QFD.* EuroSPI Salzburg, Communications in Computer and Information Science, Springer, Cham.

Fehlmann TM, Kranich E (2022) The Fixpoint Combinator in Combinatory Logic - A Step towards Autonomous Real-time Testing of Software?. *Athens Journal of Sciences* 9(1): 47–64.

Gödel K (1931) Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik* 38(1): 173–198.

Minsky M, Papert S (1972) *Perceptrons: An Introduction to Computational Geometry.* 2nd edition with corrections Hrsg. Cambridge(MA): The MIT Press.

Nico Klingler (viso.ai) (2024) *The Ultimate Guide to Understanding and Using AI Models.* [Online] Available at: https://viso.ai/deep-learning/ml-ai-models/[Zugriff am 9 February 2024].

Panigrahi A, Saunshi N, Zhao H, Arora S (2023) *Task-Specific Skill Localization in Fine-tuned Language Models.* Cornell University, Ithaca, NY: arXiv: 2302.06600v2 [cs.CL].

Raatikainen P (2020) Gödel's Incompleteness Theorems. In EN Zalta (ed.), *The Stanford Encyclopedia of Philosophy.* s.l.:s.n.

Rosenblatt F (1957) *The Perceptron: A Perceiving and Recognizing Automaton (Project PARA),* Buffalo: Cornell Aeronautical Laboratory, Inc.

Turing A (1937) On computable numbers, with an application to the Entscheidungs problem. *Proceedings of the London Mathematical Society* 42(2): 230–265.

Wolfram S (2023) *What is ChatGPT doing ... and Why Does it Work?.* Champaign, IL: Wolfram Media, Inc..

Zachos E (1978) *Kombinatorische Logik und S-Terme,* Zurich: ETH Dissertation 6214.

Zhong V et al. (2022) *Improving Policy Learning via Language Dynamics Distillation,* Cornell University: arXiv:2210.00066v1.