ATLANTIS PRESS

Atlantis Studies in Uncertainty Modelling, volume 1

11th Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT 2019)

Uncertainty of value and structure

Patrik Eklund[a] **and Magnus Löfstrand**[b]

[a]Department of Computing Science, Umeå University, 901 87 Umeå Sweden, patrik.eklund@umu.se

[b]School of Science and Technology, Örebro University, 701 82 Örebro, Sweden, magnus.lofstrand@oru.se

Abstract

In this paper we argue that 'uncertainty of information' traditionally focuses more on 'uncertainty' leaving 'information' mostly as unravelled. We also aim to explain why that is so, and we then provide suggestions on how to overcome the situation, as related to applications involving information structures.

Keywords: Application, fuzzy, logic, signature.

1 Fuzzy then, now and in the future

Since early days of fuzzy systems, two branches separated and thereafter basically never communicated. One, the larger one, started from fuzzy sets $\alpha : X \to [0,1]$ remaining over the unit interval of uncertainty values, and the other, the smaller one, started from $\alpha : X \to L$ over a lattice structure mostly seen as embracing truth values. In the unit interval, arithmetic and analysis is used, and applications involve engineering. With lattices, lattice operations were seen as logical operations, so that theoretical subtleties were enabled by imposing various structures on that lattice.

These two branches basically never seriously communicated. Those comfortable with analysis on the unit interval were mostly never seeking additional value from the side of algebra. Some theory development on this side emerged quickly within rather trivial generalizations in topology. Those being more fond of algebra and logic, became more and more theoretical, almost as developing antipathy against applications, and at the same time criticizing the simplicity of the unit interval.

Today, these two branches still basically do not collaborate. They almost contra-laborate.

What they both do not see clearly, is how they both hide information. They both involve that X, but they care less about the structure of it.

Those supporting the unit interval, and the calculus in it, usually also knew methods like neural networks quite well. Neural networks is basically about weighted sums, with quite simple mathematics. Fuzzy control based on the 'compositional rule of inference', a fancy name for relational composition, was a big success story within the fuzzy community, and it was all about the unit interval. As far as elements x in X where concerned, there was nothing behind x, except the value itself, and the interesting thing from fuzzy point of view was its uncertainty value $\alpha(x)$ in the unit interval. "$\texttt{Tall}(John) = 0.7$" was fascinatingly simple, and very appealing, but serious and real world application development based on such simple notation soon turned out not to be all that simple.

The fuzzy community created the hype that complicated problems could all be solved by simple solutions.

The algebraic and logical side is mostly introvert. It often starts out saying α is an element in L^X, and then L^X as a lattice structure does not have more structure compared to L. Ignoring X and its content was easy and natural, since 'pointless' was appealing. This way of hiding information among algebraists probably stems from refusing to recognize the practical difference between points and sets, since algebra is more comfortable dealing with sets and powersets. A set A as a subset of X means A is an element of the powerset $\mathsf{P}X$, $A \in \mathsf{P}X$. Now, an algebraist quickly says e.g. that $\mathsf{P}X$ is a boolean algebra, and, of course, it is. There is algebraic structure on $\mathsf{P}X$. Denote it as B, and A pointlessly as x, so that $x \in B$ is the new player in the algebraic structure. This hides information, and disables it forever from becoming unravelled. What an algebraist basically says is that $\alpha : X \to [0,1]$ is not only all too simple mathematics, and the least we should do is to deal with $\alpha : X \to L$, where the algebraic structure is as algebraically appealing as possible. Once this has been done, the jump from L^X to L, ignoring X, is just a natural step to take, and in fact

a "pointless necessity", and thereby universal algebra also falls outside the scope of algebra.

Note how the powerset functor can be quite devastating in hiding information. The "trick" to recover points in powersets is easy. A point x is simply identified with its singleton set $\{x\}$, and thereafter algebra over the powerset proceeds as if points actually exist. Someone may say we can actually do things like $\{2\} + \{3\} = \{2 + 3\}$, but everyone in the fuzzy community knows how things evolve in very different ways if we generalize from that. Also, if x has a property in A, given $f : X \to A$, we cannot say that $\{x\}$ has a property that can be identified with the property of x. Indeed, even if we have the extension mapping $\mathsf{P}f : \mathsf{P}X \to \mathsf{P}A$, and we have $\mathsf{P}f(\{x\}) \in \mathsf{P}A$ maybe as the $\{f(x)\}$, we are no longer in shallow waters if we would start to speak about singleton attributes.

On powersets we also come to relations between points, like a $\rho \subseteq X \times X$. This also, in its initial setting, invites to start by ignoring information that $x \in X$ represents, and having $x_1 \rho x_2$ as the informative part. With ρ assuming to have properties, this then turns to analyzing what we can do given those properties. An equivalence relation identifies points in respective equivalence classes so that it is natural to go up one level and identifies equivalence classes as new points to work with in an application setting. Clearly, attributes residing in points do not easily and naturally carry over to their equivalence classes, unless in very trivial cases. Viewing such a relation ρ in its equivalent form as a mapping $f_\rho : X \to \mathsf{P}X$, invites to different types of generalizations and unravelling as compared to viewing ρ as $\rho \subseteq X \times X$. For instance, in the case of fuzzy relations as $\sigma : X \times X \to [0, 1]$, or something L instead of $[0, 1]$, they are similarly and equivalently describable as $f_\sigma : X \to \mathsf{L}X$.

This paper warmly recommends and invites to more and broad collaboration, in particular on application development, between these the two branches. The generalized recommendation is not necessarily to compare the analysis preference to the algebraic one, and certainly not to see them as competing. Indeed, this paper warmly recommends to look deeper into X, and even deeper into x, rather than continuing to ignore, as both sides do, the content and context of X and x, and, even worse, to blur the distinction between X and $\mathsf{P}X$. Unravelling, i.e., making X into something more elaborate will invite to making P if in some applications we prefer or need to work with structured sets and populations rather than just points and individuals or individual items.

2 Sets and points

We can use a number of examples and practical situations to work with. We could choose from the public or private sectors, including industrial production, marketing, health care, education, and so on. Later in this section we select just one such example, and by no means since it is more typical than any other choice, but simply because it supports our presentation, and may also be quite familiar to many readers. The example is also related to our particular application developments in some presently ongoing projects whose funding we gratefully acknowledge, and builds upon some of some previous work, e.g., in [8, 7].

For an $x \in X$, and in an application, we are often tempted to say that such an x may represent more than it denotes. If we do so without adding any structure to x, even if we change notation to $x_{ThisAndThat}$, it is still just a point, and X is now just the set of all those representing This and That.

It's common to try to overcome such oversimplifications by adding attributes using mappings like $f : X \to A$, with $f(x)$ being what is attributed to x. We then have the pair (f, x), as more like a syntactic expression, which explicitly gives us three different things. There is f, the *operator*. We have x, the *operand*. And we have (f, x), the syntactic expression for the *result of the operation*. We mostly prefer to write $f(x)$ instead of (f, x), like $2 + 3$ or $+(2, 3)$ instead of $(+, 2, 3)$, but then we overemphasize the result of the operation, and we turn intuition quickly to the semantic value of that operation and its result.

This increases the risk of hiding information, and this makes the distinction between syntactic expression and semantic value less apparent. In fact, it invites to viewing them as one and the same thing. Thereby information structure is lost in the translation of syntactic operation to its semantic value.

We now have (f, x), or $f(x)$ if we wish, as an expression that contains f and x and from which we can 'compute' the value $f(x)$ of the operation. If we then decide to denote $f(x)$ with a, i.e., we provide a semantic interpretation where we assign according to $a = f(x)$, and continue to work with a in A, forgetting where a comes from, then we obviously hide both x and f. If we look closer at the example $2 + 3$, or $+(2, 3)$, we are invited to say it's 5, but if we work with 5 only in a subsequent expression $5 + 7$, we decline to recognize that this particular 5 comes from $2 + 3$, which combines 2 and 3 using $+$. Note already at this point that if $2+3$ is only 'approximately 5', it may not be so only because either or both of 2 and 3 are approximations, but $+$ could also be 'approximately $+$', i.e., the operation itself is uncertain. So, the expression $2+3$ is fully

informative. It even includes 5, since we can compute 5 from $2+3$, but if we only see 5 and forget where it comes from, and later wonder where it actually came from. We can then only guess, and say it could have come $1+4$, $5+0$, and so on, but we cannot be sure it came from $2+3$ unless we say something explicit like '5, as the result of $2+3$'.

At this point it is now convenient to speak of the distinction between syntax and semantics. The attribution function $f : X \to A$ is logically a semantic description of some operator $\omega : \mathtt{s}_1 \to \mathtt{s}_2$, if we look at this situation from a universal algebra point of view. We may have a syntactic expression t of type $\mathtt{s}_1$, so that $\omega(t)$ forms another expression, of type $\mathtt{s}_2$. The syntactic ω may then connect with the semantic f, where $\mathtt{s}_1$ assigns to X and $\mathtt{s}_2$ assigns to A. Here we could say that $\mathtt{s}_1$ represents the type of points under consideration, whereas $\mathtt{s}_2$ represents a type of attributes, semantically explained by elements in A. If t assigns to an x so that $\omega(t)$ assigns to $f(x)$, we see how $\omega(t)$ remains in its expression form without necessarily evaluating it, so that $\omega(t)$ is fully informative and not hiding anything. This is now within the realm of *terms over a signature* $\Sigma = (S, \Omega)$, where S is the structure of sorts, or types, and Ω is the structure of operators. The set of all possible expressions is then initially enriched to a set of terms $\mathsf{T}_\Omega X$, where X is a set of variables, and T_Ω is a term functor over some monoidal closed category. As a concrete example of an expression, an element *actuator* as a point in a set without structure is symbolically meaningless, whereas $actuator(\omega(x_1), \ldots, \omega(x_n))$, as a term, builds upon *actuator* and ω as an operator, with $x_1, \ldots, x_n$ as variables. Similarly, a temperature control could be a term $EATC(g(y, z))$. A many-valued interaction between the actuator and the temperature control is given as $\rho(actuator(f(x_1), \ldots, f(x_n)), EATC(g(y, z)))$, where $\rho : \mathsf{T}_\Omega X \times \mathsf{T}_\Omega X \to \{no, weak, strong\}$ is a three-valued relation over $\mathsf{T}_\Omega X$. This unravels hidden information as compared to modelling using only names of elements in X. This notation is based on category theory, where $\mathsf{T}_\Omega : \mathtt{Set} \to \mathtt{Set}$ is a *term functor* [3] that can be extended to a *monad*. The monad properties allow substitutions of expressions within a term to be composable, so that the substitutions $x := g(y, z)$ and $z := 22$ as composed and applied to $EATC(x)$ leads to the term $EATC(g(y, 22))$. This is obvious when we use relations over $\mathsf{T}_\Omega X$. When we need structured sets of terms, we need *monad compositions* [3].

In Section 3 we present detail on the formal term functor construction, and also how uncertainty can be attached to expressions given that also operators are attached with values of uncertainty. There also clarifies the important distinction between 'fuzzy logic' and 'logic of fuzzy'.

3 The term functor

In order to make this paper a bit more self-contained, in the following we briefly and in a general overview fashion describe the construction of terms over and underlying signature. For more detail, and for the purely categorical constructions of the corresponding term monads, the reader is referred to [3].

The many-sorted term monad $\mathbf{T}_\Sigma$ over $\mathtt{Set}_S$, the many-sorted category of sets and functions, where $\Sigma = (S, \Omega)$ is a signature, can briefly be described as follows. For a type $\mathtt{s} \in S$, we have type specific functors $\mathsf{T}_{\Sigma,\mathtt{s}} : \mathtt{Set}_S \to \mathtt{Set}$, so that

$$\mathsf{T}_\Sigma(X_\mathtt{s})_{\mathtt{s}\in S} = (\mathsf{T}_{\Sigma,\mathtt{s}}(X_\mathtt{s})_{\mathtt{s}\in S})_{\mathtt{s}\in S}.$$

The important recursive step in the term construction is

$$\mathsf{T}^\iota_{\Sigma,\mathtt{s}}(X_\mathtt{s})_{\mathtt{s}\in S} =$$

$$\coprod_{\mathtt{s}_1,\ldots,\mathtt{s}_m} (\Omega^{\mathtt{s}_1\times\cdots\times\mathtt{s}_m\to\mathtt{s}})_{\mathtt{Set}_S} \times \mathsf{arg}^{\mathtt{s}_1\times\cdots\times\mathtt{s}_m} \circ \bigcup_{\kappa<\iota} \mathsf{T}^\kappa_\Sigma(X_\mathtt{s})_{\mathtt{s}\in S}$$

and then with

$$\mathsf{T}^\iota_\Sigma(X_\mathtt{s})_{\mathtt{s}\in S} = (\mathsf{T}^\iota_{\Sigma,\mathtt{s}}(X_\mathtt{s})_{\mathtt{s}\in S})_{\mathtt{s}\in S},$$

we finally arrive at the term functor

$$\mathsf{T}_\Sigma = \bigcup_{\iota<\bar{k}} \mathsf{T}^\iota_\Sigma.$$

Note here how our (f, x) in Section 2 is in the form of a pair, where f is in $\Omega^{\mathtt{s}_1\times\cdots\times\mathtt{s}_m\to\mathtt{s}})_{\mathtt{Set}_S}$ and x is in $\mathsf{arg}^{\mathtt{s}_1\times\cdots\times\mathtt{s}_m}$.

The term functor construction can be extended so that $\mathsf{T}_\Sigma : \mathtt{C} \to \mathtt{C}$ operates more generally over monoidal biclosed categories $\mathtt{C}$. If $\mathtt{C}$ is $\mathtt{Set}$, we have the construction above, and with the Goguen category $\mathtt{Set}(Q)$, where Q is a quantale, we have a multivalent and typed situation enabled by the signature acting over the selected underlying category. The algebraic foundations of many-valuedness, including techniques related to the use of quantales and other algebraic structures, is found in [5].

We now have more interesting fuzzy sets $\alpha : \mathsf{T}_\Sigma X \to Q$ as objects of $\mathtt{Set}(Q)$, where $t \in \mathsf{T}_\Sigma X$ potentially carries lots of information, as compared to having only $\alpha : X \to Q$, where $x \in X$ without further attachments to x is basically just a variable that can carry unexplained data. In the case of the term $actuator(\omega(x_1), \ldots, \omega(x_n))$, and with T_Σ over $\mathtt{Set}(Q)$, *actuator* and ω become annotated with uncertainty values in Q. Similarly, in the case of the expression $2+3$, the uncertainty of 2 and 3 are both qualified by values in Q, and so is the operation $+$.

Monad compositions further enable to arrive at generalized sets of terms, where the typical example is composing the term functor T_Σ with the powerset functor P in order to obtain the monad $\mathbf{P} \circ \mathbf{T}_\Sigma$. More elaborate generalized set functors Φ can be applied in order to make use of the composition $\mathbf{\Phi} \circ \mathbf{T}_\Sigma$.

Here we can formally explain the danger of pursuing the 'singleton attributes' possibility, discussed in Section reflookingmoredeeply. The monad multiplication reduces from PTPT to PT, which requires a certain swapper from TP to PT so that we get a jump from PTPT to PPTT. From there we have the respective multiplications from PP to P, as a 'flattening', and from TT to T, as an idempotency, which reduces PPTT to PT. The key to this is indeed that swapper. It is a generalized distributive law. However, we do not have a swapper from PT to TP, and thereby we cannot establish TP as a monad. This we can have $\omega(\{t_1, t_2\})$ reducing to $\{\omega(t_1), \omega(t_2)\}$, but we cannot have $\{\omega_1(t), \omega_2(t)\})$ reducing to something like $\{\omega_1, \omega_2\}(t)$, which makes less sense, since we do not have anything enabling us to go from (S, Ω) to $(S, \mathsf{P}\Omega)$. Similarly, we do not have anything like $(\mathsf{P}S, \Omega)$. However, we do have 'powertypes', as an example of a type constructor, as shown in Section 4.

Here we also see more clearly what it means to hide information by focusing only on the algebra structure L^X, ignoring the information residing in X and its elements. Looking at $\mathsf{T}_\Sigma X$ and focusing only on the algebraic properties of $L^{\mathsf{T}_\Sigma X}$ effectively ignores and hides everything enabled by the information structure enabled by Σ and residing in $\mathsf{T}_\Sigma X$. This is a strong message conveyed in this paper.

4 The three-level signature

In the three-level arrangement of signatures [6], the middle level enables to use various kind of type constructors, where the first and third level clearly distinguishes terms from λ-terms.

Section 3 presents a functorial term construction. The three-level signature enables to provide a functorial λ-term construction. The conventional non-functiorial (in fact, natural language based language based) definition of λ-terms is informal and non-constructive, as it creates undesired terms and demands renaming.

Church's view about λ was that it is just an informal symbol [1]. The three-level signature shows clearly that λ is not to be seen as a *general abstractor*, but rather so that any operator possesses its own capacity to abstract itself. In any expression $\lambda x.f$, λ is unique to f, and should be clearly viewed as "f owns its λ".

In λ-calculus, the main type constructor is the one producing function types. In traditional views of λ-terms, this function type is given from the 'outside', i.e., it is not seen as part of any underlying signature.

In order to see this more precisely, let $\mathtt{s}_1$ and $\mathtt{s}_2$ be two types in S. The function type involving $\mathtt{s}_1$ and $\mathtt{s}_2$ can be denoted $\mathtt{s}_1 \Rrightarrow \mathtt{s}_2$. Even if we want to view $\mathtt{s}_1 \Rrightarrow \mathtt{s}_2$ as a (constructed) type, it is not part of S. This creates an awkward meta-level of constructors, and the formalism for treating these constructors is rather lose, or even non-existing. Thus, the traditional so called 'set' of λ-terms is not well-defined, even if it is 'well understood'.

The three-level arrangement of signatures starts from the basic signature Σ on level one, targeting a resulting Σ' on level three, using type constructors on level two. On level two we have the (Σ-)*superseding type signature* as a one-sorted (one type only) signature $\mathtt{S}_\Sigma = (\{\mathtt{type}\}, \mathtt{Q})$, where $\mathtt{Q}$ is a set of *type constructors* satisfying

(i) $\mathtt{s} :\rightarrow \mathtt{type}$ is in $\mathtt{Q}$ for all $\mathtt{s} \in S$

(ii) there is a $\Rrightarrow\colon \mathtt{type} \times \mathtt{type} \to \mathtt{type}$ in $\mathtt{Q}$

If $\mathtt{Q}$ does not contain any other type constructors, apart from those given by (i) and (ii), we say that $\mathtt{S}_\Sigma$ is a (Σ-)*superseding simple type signature*. Then $\mathsf{T}_{\mathtt{S}_\Sigma} X$, where X is the tuple of objects representing (type) variables, contains all terms which we call *type terms*. We may write $\mathtt{s} \Rrightarrow \mathtt{t}$ for the type term $\Rrightarrow(\mathtt{s}, \mathtt{t})$.

The signature $\Sigma' = (S', \Omega')$ on level three then is based on $S' = \mathsf{T}_{\mathtt{S}_\Sigma}\emptyset$, i.e., the types on level three are those from level one together with the constructed types, on level two appearing as terms (the type terms), added to those basic types coming from level one.

Church's type constructor is in effect our $\Rrightarrow$, so that $(\beta \Rrightarrow \alpha)$ is Church's $(\beta\alpha)$. An interpretation of Church's ι corresponds to our $\mathtt{type}$ and for Church's o there is no corresponding structure.

In summary, the three signature levels underlying the production of λ-terms are then following.

1. the level of primitive underlying operations, with a usual many-sorted signature $\Sigma = (S, \Omega)$
2. the level of type constructors, with a single-sorted signature
   $\mathtt{S}_\Sigma = (\{\mathtt{type}\}, \{\mathtt{s} :\rightarrow \mathtt{type} \mid \mathtt{s} \in S\} \cup \{\Rrightarrow\colon \mathtt{type} \times \mathtt{type} \to \mathtt{type}\})$
3. the level including λ-terms based on the signature $\Sigma' = (S', \Omega')$ where $S' = \mathsf{T}_{\mathtt{S}_\Sigma}\emptyset$, $\Omega' = \{\lambda^{\omega}_{i_1,\ldots,i_n} :\rightarrow (\mathtt{s}_{i_1} \Rrightarrow \cdots \Rrightarrow (\mathtt{s}_{i_{n-1}} \Rrightarrow (\mathtt{s}_{i_n} \Rrightarrow \mathtt{s})) \mid \omega : \mathtt{s}_1 \times \ldots \times \mathtt{s}_n \to \mathtt{s} \in \Omega\} \cup \{\mathtt{app}_{\mathtt{s},\mathtt{t}} : (\mathtt{s} \Rrightarrow \mathtt{t}) \times \mathtt{s} \to \mathtt{t}\}$

Here $(i_1, \ldots, i_n)$ is a permutation of $(1, \ldots, n)$. Note also that level one operators are always transformed to constants on level three. In traditional notation in λ-calculus, substituting x by $\mathtt{succ}(y)$ in $\lambda y.\mathtt{succ}(x)$ requires a renaming of the bound variable y, e.g., $\lambda z.\mathtt{succ}(\mathtt{succ}(y))$.

In our approach we avoid the need for renaming. On level one, and in the case of NAT, we have the substitution (Kleisli morphism) $\sigma_{\mathtt{nat}} : X_{\mathtt{nat}} \to \mathsf{T}_{\mathtt{NAT},\mathtt{nat}}(X_{\mathtt{t}})_{\mathtt{t}\in\{\mathtt{nat}\}}$, where $\sigma_{\mathtt{nat}}(x) = \mathtt{succ}(y)$, x being a variable on level one, and the extension of $\sigma_{\mathtt{nat}}$ is $\mu_{X_{\mathtt{nat}}} \circ \mathsf{T}_{\mathtt{NAT},\mathtt{nat}}\ (\sigma_{\mathtt{t}})_{\mathtt{t}\in\{\mathtt{nat}\}} : \mathsf{T}_{\mathtt{NAT},\mathtt{nat}}(X_{\mathtt{t}})_{\mathtt{t}\in\{\mathtt{nat}\}} \to \mathsf{T}_{\mathtt{NAT},\mathtt{nat}}(X_{\mathtt{t}})_{\mathtt{t}\in\{\mathtt{nat}\}}$. On level three we have $\sigma_{\mathtt{nat}'} : X_{\mathtt{nat}'} \to \mathsf{T}_{\mathtt{NAT}',\mathtt{nat}'}(X_{\mathtt{t}})_{\mathtt{t}\in S'}$, with $\sigma_{\mathtt{nat}'}(x) = \mathtt{app}_{\mathtt{nat}',\mathtt{nat}'}(\lambda_1^{\mathtt{succ}}, x)$, x being a variable on level three, and requiring no renaming in $\mu_{\mathtt{nat}'} \circ \mathsf{T}_{\mathtt{NAT}',\mathtt{nat}'}\ \sigma_{\mathtt{nat}'}(\mathtt{app}_{\mathtt{nat}',\mathtt{nat}'}(\lambda_1^{\mathtt{succ}}, x))$.

On β-reduction we obviously have the following transition from the traditional form to using the three-level signature. Let $[x := t]$ be a substitution, i.e., we have some $\sigma(x) = t$, and choose a $\omega : \mathtt{s}_1 \times \mathtt{s}_2 \to \mathtt{s}$. Then β-reduction

$$\lambda x.\lambda y.\omega(x, y)\ t$$

$$\to_\beta$$

$$\lambda y.(\omega(x, y)[x := t]) = \lambda y.\omega(t, y) :: \mathtt{s}_2 \Rrightarrow \mathtt{s}$$

transforms to

$$(\mu \circ \mathsf{T}\sigma)(\mathtt{app}(\lambda^{\omega}_{\mathtt{s}_1,\mathtt{s}_2}, x))$$

$$\to_\beta$$

$$\mathtt{app}(\lambda^{\omega}_{\mathtt{s}_1,\mathtt{s}_2}, t) :: \mathtt{s}_2 \Rrightarrow \mathtt{s}.$$

These construction over the Goguen category, a monoidal closed category, defines a truly fuzzy λ-calculus. The λ-term monad may also be considered to be over other monoidal biclosed categories [3].

Various 'syntactic set functors' can be further introduced, including the 'powerset' type constructor $\mathsf{P} : \mathtt{type} \to \mathtt{type}$ on level two, intuitively thinking that the 'algebra' of P is the powerset functor, with the underlying monoidal closed category being the category of sets and functions.

The operator $\omega : \mathtt{s} \to \mathsf{P}\mathtt{t}$ can be seen as the underlying syntactic support for enabling typed generalized relations.

In the case of description logic, we can transform it into our categorical framework [2]. This provides a truly fuzzy description logic, which in fact becomes a fuzzy λ-calculus.

Formal concept analysis has also been shown to enable the use of the powertype [4].

5 Conclusions with respect to applications

'Application' is a very broad concept, in particular in real world applications. Our general view on theory for real world applications is that simple solutions for simple problems is less interesting, as it mostly involves shallow scientific methodology.

Simple problems are many, and simple solutions mostly suffice. A simple problem that requires a complicated solution is only apparently simple, but in reality complicated. Complicated problems require complicated solutions. In some cases, a complicated problem may have a simple solution, but in that case the problem was apparently not that complicated. So we have simple problems with simple solutions and complicated problems with complicated solutions. Our focus is on the latter.

The challenge in understanding applications and their underlying problems is understanding the problem domain. Theoreticians usually do not spend sufficient amount of time to understand a problem before trying to solve it. This leads only to simplifying and fitting the problem to an existing theoretical solutions. Such an approach is seldom successful. Successful application development seeks a solution to a problem. We should not first fix the solution, and then seek to fit the problem to the solution. This, in general, dilutes the application to unreal.

Simplifying may be advisable. We may have an information structure where we roll up into lesser detail or at a certain step being content with a helicopter view of the problem. But the underlying information must not be forgotten or ignored, and certainly not hidden so that it can never be recovered. In $actuator(\omega_1(t_1), \ldots, \omega_n(t_n))$, with further specific information residing in expressions $t_1, \ldots, t_n$, we may at some point of application development only need to work with aggregated information related with *actuator*, but later on need to deal specifically with all $\omega_1(t_1), \ldots, \omega_n(t_n)$.

How deep then do we really need to drill down? This is up to what we want to achieve in a particular application. Take hypertension as an example, and suppose we deal with *essential hypertension*, i.e., high blood pressure not diagnosed as a consequence of any underlying disease. In a shallow view of treatment, it's about selecting drug and dose so that pressure goes down, i.e., higher dose means lower pressure. But it's not that simple. There are different types of drugs, targeting the body in different ways. Then we must understand how they target. Some drugs are antagonists or blockers, like the angiotensin receptor and calcium channel blockers (ARB, CCB). What the par-

ticular active ingredients do is to affect cells in the arterial wall, thus relaxing and widening the blood vessel. Then one might wonder how this blocking in cells actually happens in the case of ARBs and CCBs, and the acting molecules find their way through the intestines and then liver into the blood stream, and thereby to the cells in question. Experts in this area will have quite precise answers to this question (even if the authors of this paper don't), and that answer is certainly interesting, but is it relevant in that particular application dealing with hypertension treatment? In this example we see how we must choose the level of detail, i.e., how far we must drill down (expand the underlying signature!) in order to explain the problem in sufficient detail, without unnecessarily over-explaining it. So, blood pressure alone is too shallow, but intracellular molecular interaction may be too much.

Industrial applications related to system-of-systems (SoS) [7, 9] are similar. Such an SoS may be a car with the structure of the car in focus, or the SoS may be a car in traffic, so the traffic system is the main SoS, where the car is just a subsystem of it. In the latter case, *actuator* may be a sufficient level, where in the former case we need $actuator(\omega_1(t_1), \ldots, \omega_n(t_n))$ even drilling down from $\omega_1(t_1), \ldots, \omega_n(t_n)$ into all terms t_i. An SoS as a production plant combines availability and maintenance, e.g., for prediction of production process availability and performance. Improved predictive maintenance decision support through 'what-if' scenario analysis and optimisation is often desired. In a typical shallow approach, focus is only on optimising maintenance for each item of equipment in a production line, thus failing to consider the interaction between maintenance and production in sufficient detail. Components and product taxonomies are more informative within activity and process hierarchies, supporting decision-makers in various ways.

In general for machines, *faults* and *failures* reduce their *function* either partly or completely. The traditional engineering view of functioning does not connect accurately with faults. In order to facilitate these connections, we need a *common language for representation of faults and functioning*, based on appropriate nomenclatures. In addition, machines are operated by a wide range of professionals, and the business of machine production and sales involves other types of professionals.

Numerical approaches to information structure development within the industry as a whole promotes uncertainty and *many-valued considerations* mostly for analyzing variability, and as related to numerical values only. Logical many-valuedness, dealing differently with these underlying structures, are basically missing in most numerical approaches [8].

Acknowledgement

This work is supported by the Production Centred Maintenance (PCM), *NORDIC Icing Center of Expertise* (NOice), and *A digital twin for sustainable and available production as a service* (DT-SAPS) projects, with gratefully acknowledged funding, respectively, from the Knowledge Foundation (Stiftelsen för kunskaps- och kompetensutveckling), the Interreg Botnia-Atlantica 2014-2020 programme, and the Swedish Innovation Agency (VINNOVA) PRODUKTION2030 programme Call nr 11.

References

[1] A. Church, *A formulation of the simple theory of types*, The journal of symbolic logic **5** (1940), 56-68.

[2] P. Eklund, *The syntax of many-valued relations*, J.P. Carvalho et al. (Eds.): IPMU 2016, Part II, CCIS **611**, Springer-Verlag Berlin Heidelberg, 2016, 61-68.

[3] P. Eklund, M.A. Galán, R. Helgesson, J. Kortelainen, *Fuzzy terms*, Fuzzy Sets and Systems **256** (2014), 211-235.

[4] P. Eklund, M.A. Galán, J. Kortelainen, M. Ojeda-Aciego, *Monadic formal concept analysis*, RSCTC 2014, (Eds. C. Cornelis et al.), Lecture Notes in Artificial Intelligence **8536** (2014), 201-210.

[5] P. Eklund, J. Gutiérrez García, U. Höhle, J. Kortelainen, *Semigroups in Complete Lattices: Quantales, Modules and Related Topics*, Developments in Mathematics **54**, Springer, 2018.

[6] P. Eklund, R. Helgesson, *Modern eyes on λ terms*, Presented at International Workshop on 75 Years of the λ-Calculus, St Andrews (Scotland), 15 June 2012. GLIOC Notes, October, 2013.

[7] P. Eklund, M. Johansson, J. Kortelainen, *The logic of information and processes in system-of-systems applications*, Soft Computing Applications for Group Decision-making and Consensus Modeling (Eds. M. Collan and J. Kacprzyk), Studies in Fuzziness **357**, Springer, 2018, 89-102.

[8] P. Eklund, M. Löfstrand, *Many-valued logic in manufacturing*, Position Papers of the 2016 Federated Conference on Computer Science and Information Systems (FedCSIS), ACSIS **9** (2016), 1117.

[9] M. Jamshidi, *System-of-Systems Engineering - A Definition*, IEEE SMC 2005, 10-12 Oct. 2005.