Object-oriented solutions †

V.E.Wolfengagen‡

Vorotnikovsky per., 7, bld. 4
Institute for Contemporary Education "JurInfoR-MSU"
Moscow, 103006, Russia
vew@jmsuice.msk.ru

Abstract

In this paper are briefly outlined the motivations, mathematical ideas in use, pre-formalization and assumptions, object-as-functor construction, 'soft' types and concept constructions, case study for concepts based on variable domains, extracting a computational background, and examples of evaluations.

1 Introduction

An early incite in a *theory of computations* was to incorporate objects for a variety of purposes. They were assumed to represent the existent - *actual, possible or virtual* objects in a problem domain. The nature of existence was also under the concentrated study. The recent years have generated a lot of object assumptions and discussions. Nevertheless, the initial notion of an object became overloaded by the mismeaning and not significant features. Every new research in the area added the excessive troubles to understand the clear sense and meaning of the object paradigm.

An attempt to rearrange the useful ideas will be done here. The main attention is paid to establishing the parallelism between a theory of computations and the object-oriented notions.

1.1 Motivation for object evaluator

Object can be represented by embedding in a host computational environment. An embedded object is accessed by the laws of the host system. A pre-embedded object is observed as the decomposition into substitutional part and access function part which are generated during the object evaluation. They assist to easy extract of the result.

Subsumption is an usual theory-of-computation technique. Counterparts of the entire method – logic, functor category, and applicative computations, – are attached to generate an intermediate computational framework. This intermediate representation is indirectly based on the categorical combinatory logic. The needed optimizations may be obtained equationally.

† *This research is supported by the Russian Foundation for Basic Research (project 93-012-943)*

‡ also: Kashirskoe Avenue, 31, Cybernetics Department, Moscow Engineering Physical Institute, Moscow, 115409, Russia

Proceedings of the 2nd International Workshop on Advances in Databases and Information Systems (ADBIS'95). Vol. 1: Regular papers.
Moscow, June 27–30, 1995.

The resulting model seems to be a kind of *object evaluator*. The object evaluator feature is to incorporate the schematic elements which are subdivided into *individuals* and *individual concepts*. Both of the entities are based on the notion of the *variable domain*. This is a schematic construction and is equipped with both the cloning and transactional means to capture *dynamics*.

All the parts of object evaluator share the same functor model with the parameterized types and assignments. The logical part has been supplied with both the atomic and non-atomic formulae with the variables ranging over the variable domains. The categorical part assists the evaluation to enable the extraction of a substitutional part. The applicative part is capable of separating the computation paths for function and its argument.

In this paper are briefly outlined the motivations, mathematical ideas in use, pre-formalization and assumptions, object-as-functor construction, 'soft' types and concept constructions, case study for concepts based on variable domains, extracting a computational background, and examples of evaluations.

1.2 Evolution of the ideas

A technical intuition for an object is approximately as follows: object is proclaimed to be an entity (by default) with the strictly attached attributes – 'internal state' and 'behavior' [EGS93]. Some of the objects are called the 'dynamic objects', that communicate with each other (note that *communication* is presupposed of great importance). Next step is to classify objects by their type, to collect objects into classes, to superimpose the various inheritances, to compose for generating complex objects. Note that computational intuition tends to establish object as a mathematical process.

1.2.1 Logic to incorporate objects

An approach to apply logic to a phenomena of object seems to be clear and natural (e.g., [HC89], [Jac92] [Gab93]). Nevertheless, adoption of more or less traditional approach of logic is distant of the essence of the initial task to be solved. When the researcher was to pick this kind of science it would combine some significant elements.

1. The conditions of reasoning that transcend not only logic, but both the mathematics and the theory of computation(s).
2. The traditions of observation and insight that led into the foundations of these sciences.

The semantics of traditional logics is often used but this argument is not sound. The most important for the notation is to be usable by the computational tool that applies it to the *environment* to produce the *results*.

To share the concern for the rigorous theory it is not necessary to adopt all the amount of any particular formalism. The more prominent approach seems to be based more on the constraints that can be superimposed by the problem. If the existing formalism turns off to match these conditions then that means a perspective to find out for meaningful thing.

If to confine the search for the theory of objects to areas were formalism has already succeed in the answer may to be missed. A necessary theory is likely to be found where the logic meets the incompleteness, troubles of intensions etc. As a rule, the traditional logical machinery seems to be well applied to pre-formalized reality and is not suitable equipped with the means for more dynamic occasions. When we go back to the generic principal ideas we have more possibilities to expand the predefined tools to deal with the problem as it is arose and used by the computational devices.

1.2.2 Manifesting a category theory

An early trouble was the suitability of a theory for the working researcher. The same is for a category theory.

The theoretician position (see [Law75], [Gog89], [EGS91]) seems to have embraced category theory as the pre-eminent universal science, to adopt its more or less traditional approach with a possible missing the significant initial features.

The term 'arrow thinking' as it is used in a category theory refers to the standardized notion – within this theory, – that prescribes a mapping of the terms and expressions of the initial system into a world of abstract functions. But it is only one element of categorical philosophy. In most significant applications of category theory such a thinking does not map symbolic expressions into real objects with the substantial properties, and such models become only imaginable.

For some systems of logic model is described by the theory, e.g. in the form of cartesian closed category (c.c.c.). The need is to manipulate the elements. A domain T is said to have an element if there is a map $h : I \to T$ (here: I is a domain of assignments). If f is a constant function $f : T \to S$, then $f \circ h : I \to S$. Thus, maps in c.c.c. can behave as functions on elements.

1.2.3 Applicative computational systems

A lot of theories (not necessary logic or category theory) have the ultimate goal to develop a notion or construction which suits for the interpretation of abstract objects. For instance, $\lambda-$*calculus* and *combinatory logic* contain in their foundation a concept of object to suit the computational needs ([Sco80]).

Moreover, an isomorphism between intuitionist logics and typed $\lambda-$calculi was established. An original Curry's theory of functions generated formula-as-type notion under which a proof of a statement $\exists x B$ is a pair $< a, b >$ consisting of an object a and a proof $B[a]$. In practice, a *type* is regarded as an abstract object whereas a *formula* is the name of a type.

All of this is in harmony with a principal feature of the applicative computations, namely: (1) the symbols of function and its argument are treated separately as the distinct objects; (2) the first object is applied to the second under an *application* metaoperator. The advantages of this approach are not yet entirely observed.

1.2.4 Intermediate theoretical framework

All of the theories above seem to have an universality. The method is to add the restrictions to enrich the pure theory by the needed sensitivity.

For instance, the connection between $\lambda-$calculus and c.c.c. ([CCM85]) has generated the variants of categorical combinatory logic. A basic concept for the approach was given by the set of abstract objects, namely, *categorical combinators*. This kind of objects is within both a category and computational system. They share the clear advantages of the distinct subsystems.

1.2.5 Introducing abstract objects

A 'phenomena' of object was discussed many times with a lot of attitudes. Some selected and superimposed questions seem to be as follows:

> how new individuals come into existence and go away as situation changed;
>
> how concepts get their semantics in realistic conditions e.g., with a tremendous set of possible worlds;
>
> traditional (logical) machineries are usable to prove the existence of an individual (under some properties) but give no equipment to name that, possibly generated, individual and refer to it by name in further consideration;
>
> first-order logic provides a tool to support the necessary truths and their consequences. It provides no machinery to deal with the relationships with the creation or destruction of individuals;
>
> what is the machinery to characterize the way individuals change over time;
>
> what is an ability to reify the notion of the state (in different contexts);
>
> how to talk about both the direct and side effects of the actions;
>
> ...

All of this place the state-of-things in the proper perspective. All of this clearly indicate that the long term hoped-for unified logic, categorical framework, or computational system is not yet reached. The variety of logics, theories, models, and approaches tends to more growth.

2 Restricting the topic: pre-formalization

Some efforts to encircle the task will be needed. Both direct and indirect solutions are substantially based on putting the ideas in a certain order. Subsumption is a common technique shared by distinct 'dimensions' – logical, categorical, and computational ([Wol93]).

2.1 The starting assumptions

Most of the approaches start with the notion of a *problem domain*. The problem domain is viewed as a part of physical or imaginable (perceptual) reality, or external world. This is a natural starting point. As a result the observer is to operate with a *representation*. The represented domain is inhabited by the (atomic) entities, or *individuals*. A safety reason is to set up individual as a primary concept that is not assumed to be definable. In fact, the observer operates with the *constructs* that represent the individuals.

> Important: The possibility does exist to gather the individuals into a single domain D, and this D is given from the beginning.

The advanced studies in a theory of computations prescribe D as a domain of *potential* (or: *schematic*) individuals. To the contrast the recent object-oriented studies almost ignore this fact. This ignorance does omit namely the feature of potentiality, or possibility of individual. The individual is possible with respect to some theory (of individuals).

> Advance: The individual may be relativized and gives a family of object-oriented strategies.

E.g., 'this theory of objects is similar to usual'. The individuals (theories) enter the domain and leave it cancelling their own existence. The 'flow of events' in the example may be based on a time flow. Any two theories are to be compared in spite of their existence in different 'moments'. The theories are not necessary fixed, thus all amount of the possible individuals is involved.

Further advance: The individuals are separated, at least, into *possible* and *virtual*.

Only the virtual individuals are completely *ideal objects*. So the regularity of the *observer's language* is increased. In mathematical practice to be a possible individual means to be *described*, but the virtual individual (objects) does need the *axioms*.

Effect: The virtual objects increase the structure regularity of the (initial) domain D.

As a result, clear distinction between *actual*, *possible* and *virtual* individuals induces the inclusion:

$$A \subseteq D \subseteq V,$$

where A is a set of actual individuals, D is a set of possible individuals, and V is a set of virtual individuals.

Advance: The central computational proposal is to generate actual individuals as the different *families* of D,

$$A_i \subseteq D \text{ for } i \in I.$$

Trouble: The object-oriented approaches propose to operate a fuzzy notion of a *thing* and *property* ignoring the distinctions between generic and derived concepts. The language of the observer is likely mixed with the domain D. Thus, the meaning of an individual is violated.

2.2 Other generic notions

Starting with things and properties the observer builds the *composite things* and establishes for his objects the attributes (is there any object without attribute ?). Thus, an observer actually needs a (logical) language, even overcoming his own initial desire. The obvious approach is getting started with a choice of logics.

Trouble: The logics is not homogeneous. Its branches, especially for a theory of computations contain the suitable advances. They do not suit the amorphous idea of a thing and property.

Instead of overcoming this barrier theory of computations enables the regular and working logics of *the descriptions*. The descriptions directly illustrate the difficulties and tend to general operators.

Operating with things and properties gives a specific property - *law*. The law is essentially the constraint superimposing to the properties of a thing.

Recall that in application the observer assigns attributes to things (they are not the intrinsic to things in contrast to properties).

Important: Both the logical formula $\Phi(x)$ and $\lambda-$expression $\lambda x.\Phi(x)$ give the property, but the direct assignment of the property $\Phi(\cdot)$ to the individual x is given by the description:

$$\mathcal{I}x.\Phi(x),$$

with a sense 'the (unique) x that $\Phi(x)$' (compare with $\lambda x.\Phi(x)$, 'those x that $\Phi(x)$').

Filling in the gap: The gap between the observer (and his language) on the one hand and the individuals on the other hand does exist in object-oriented modelling.

An abridgement is given by the *evaluation map*:

$$\| \cdot \| \cdot : \left\{ \begin{array}{l} \text{descriptions} \\ \lambda - \text{expressions} \end{array} \right\} \times \text{assignments} \rightarrow \text{individuals}.$$

(Here: an assignment is temporary viewed as an index ranging the families.) The abridged concepts are an *attribute* a and *property* $\Phi(\cdot)$ (via the description):

$$a = \| \mathcal{I}x.\Phi(x) \|_i \ \text{ for } i \in I \qquad (\mathbf{Attr})$$

An attribute thus defined indicates the set of individuals with a property $\Phi(\cdot)$. In usual terms the *functional representation of attribute* is established (attribute is a mapping *from* a set of things and a set of 'observation points' *into* a set of values). Note that a 'thing' is represented by the 'description'.

Principle adopted: The attribute is defined by (**Attr**). The addition of the uniqueness

$$\{a\} = \{d \in D \mid \ \|\Phi(\bar{d})\|_i = true\} \quad (\mathbf{Singleton})$$

as necessary and sufficient condition

$$\begin{array}{l} \|\mathcal{I}x.\Phi(x)\|_i = a \Leftrightarrow \\ \Leftrightarrow \{a\} = \{d \in D \mid \ \|\Phi(\bar{d})\|_i = true\} \quad (\mathbf{Unique}) \end{array}$$

enforces the observer to conclude: fixing the family $i \in I$ and evaluating $\|\Phi(\bar{d})\|_i$ relatively to every $d \in D$, he verifies the uniqueness of d.

Here the individual is called as a and is adopted as an evaluation of the description relatively to i.

2.3 Functional scheme

A general solution for attributes attracts the set of attribute functions (**Attr**) that is called as a *functional scheme*.

Advance: Equation (**Attr**) is to be revised as follows:

$$\begin{array}{ll} \mathcal{I}x.\Phi(x) = \bar{h} & \text{in a language of observer} \\ \|\bar{h}\| = h & \text{is an individual concept} \\ & \text{in a domain} \\ h(i) = a & \text{is an individual} \\ & \text{in a domain} \end{array}$$

Further advance: Previously given scheme has a universe of discourse as 'concept-individual'. An undevoted observer if needed may prefer the 'individual-state' universe.

Thus, if h is an individual, then a is its *state* under the *forcing condition* i.

Advantage: The generalized individuals (or: concepts) are schematic:

$$h : I \rightarrow C,$$

where h is a mapping from the 'observation points' into the (subset of) attribute C. The latter undoubtedly is *the* set of individuals.

There is a clear reason to call h as a *concept*. Thus a concept really represent the functional scheme.

Effect: The (individual) functional schemes are to be gathered into a greater stock:

$$\{h \mid h : I \to C\} = H_C(I) \quad (\mathbf{VDom}).$$

Certainly, $H_C(I)$ is and idealized object.

Important: The object $H_C(I)$ is a representation, and what is specific the feature of a *variable domain* is captured.

The possibilities and the advantages of a notion of variable domain are applied mostly to the *dynamics*.

2.4 Dynamics of objects

The state in an object-oriented approach is viewed as the value of the functions in the functional scheme at a given point among the 'observation points'. This agrees with the computational framework.

Important: Computationally a set of individuals is generated by:

$$H_C(\{i\}) \subseteq C \text{ for } i \in I.$$

This set is a state of a variable domain $H_C(I)$, where C gives the local universe of possible individuals. The pointer i marks the family of individuals that is 'observed' from i. The states $s_1, s_2, \ldots$ of a functional scheme have a representation by the *stages* of the variable domain:

$$\begin{array}{lcl} H_C(\{i\}) & = & \{h(i)\} \subseteq C \\ H_E(\{i\}) & = & \{h(i)\} \subseteq E \\ \ldots & & \ldots \end{array}$$

Transformations $g : s_1 \mapsto s_2$ are the counterparts of the *events* (they are triples):

$$< s_1, s_2; g > .$$

Generalization: The notion of a variable domain gives the natural observation of the dynamics in an object-oriented approach. Even more, it gives a suitable metatheoretic framework.

To cover the possible effects the *natural transformations* $H_g : H_C \to H_E$ are added. The element-wise consideration gives:

$$\begin{array}{lcccl} H_g(I) & : & h \in H_C(I) & \mapsto & g \circ h \in H_E(I), \\ H_g(\{i\}) & : & \{h(i)\} \subseteq C & \mapsto & (g \circ h)(i) \subseteq E. \end{array}$$

Important: The set of transformations gives the *laws of things* in object-oriented reasoning.

The immediate result gives a clear understanding of interaction of things (via state variable common to interacting things). Thus, the set of natural transformations is a representation of the laws of . . . . And here is a short diagram of what of:

$$\begin{array}{c} \{h(i)\} \subseteq C \\ x_1 \in \{h(i)\}; x_2 \in \{h(i)\} \\ \ldots\ \Phi(x_1) \& \Psi(x_2) \& x_1 = x_2 (= z)\ \ldots\ , \end{array}$$

where z is a common variable (joint state variable).

2.5 Dynamics via evolvent

The more dynamics may be added to an object. The task under solution is a *behavior* of a thing (= state evolution 'in a time'). Note that the state will change due to both the external and internal events.

Important: The evolvent of stages is needed:

$$f : B \to I,$$

where stages are evolved *from I to B* (note the reversed order, so B is later than I).

Computationally are given: $H_g : H_C \to H_E$ for $g : C \to E$ (C, E are the attributes) and $f : B \to I$ for stages I, B. The combined transformation is generated both by f and g:

$$\begin{array}{lcl} h \in H_C(I) & \mapsto & g \circ h \circ f \in H_E(B), \\ \{h(i)\} \subseteq C & \mapsto & ((g \circ h) \circ f)(b) \subseteq E. \end{array}$$

for $b \in B$.

In particular, a stable state is generated by:

$$\begin{array}{lcl} f & = & 1_I : I \to I, \\ g & = & 1_C : C \to C. \end{array}$$

2.6 Object characteristics

The commonly used in object studies are *encapsulation, composition, classification*, and *communication/transaction*.

Encapsulation: An object contains: (1) *state*, (2) capability of *transitions* (state changes; actions; services), and (3) *interface*.

Computationally, an object has: (1) attributes $C, E, \ldots$; (2) transformations $g : C \to E, \ \ldots$; and (3) composable transformations (possibly, they are closed under composition). In particular, objects with exclusively *interface attributes* are viewed as the *static objects*. This can be modelled by $g = 1_C : C \to C, \ f = 1_I : I \to I$ etc.

Composition: As usually, the composite object is assumed to be combined from the other objects.

This means the following: (1) logics (of the properties) is attached, (2) composition (possibly, in a category) is added etc. All of this is in a full harmony with the theory of computations.

Classification: Traditionally, the objects with the same set of properties (attributes, actions) are gathered into a class.

The computational generalization attracts the concept of a variable domain $H_C(I) = \{h \mid h : I \to C\}$ that is defined over the *schematic objects*.

Communication/interaction: Ordinarily communication mainly implies the changes of the object attributes (change is the same as a *request*). A request may cause a state transition (change of the non-interface attributes; change the state of the receiver/sender via interface attributes).

3 Construction of object

A point of importance to determine an object is the notion of *type*. The known results either illustrate the analogy between typed and type-free models, or establish their real connection. In particular, untyped models contain object-as-types via embedding. The computation, e.g., in type-free λ-calculus has a goal to derive an object with the pre-defined properties (*dynamic typing*). To the contrary, the same computation in a typed λ-calculus has to obtain the derived type by the rules (*static typing*).

To conform types with dynamics they are to be fitted the dynamical considerations. The initial set of 'hard' types is usually predefined. To the contrary the 'soft' types are derived from the generic to give rise to a more flexible ground.

Untyped models naturally combine type and its implementation (embedded objects). Sometimes the researcher may prefer to separate them. As a working hypothesis the thesis 'to represent means to classify properly ' meets the opposition from the alternative approach. This second way tends to the 'slight' variations of the initially formed objects.

3.1 Embedding objects into functor category

Give a *construction* to accumulate the intuitive reasons above. Let to consider more than one category. At first, given category $\mathcal{C}$ is a *set* and is assumed as c.c.c. Let $\mathcal{S}$ be the category of all sets and arbitrary functions, a c.c.c. Construction of the functor category (it is a c.c.c.) $\mathcal{S}^{\mathcal{C}^{op}}$ give all the (contravariant) functors from $\mathcal{C}$ into $\mathcal{S}$. The known result is that the functor category is a model for higher order logic.

3.1.1 Object-as-functor

Let a mapping $F : \mathcal{C} \to \mathcal{S}$ be the association to arbitrary domain I of $\mathcal{C}$ a set $F(I)$ of $\mathcal{S}$ and to every map $f : B \to I$ of $\mathcal{C}$ a function $F(f) : F(I) \to F(B)$ so that:

$$F(1_I) = 1_{F(I)}, \text{and} F(f \circ g) = F(g) \circ F(f),$$

provided $f : B \to I$ and $g : C \to B$ in $\mathcal{C}$.

So defined functor F determines the family of objects parameterized by I.

3.1.2 Object-as-domain

To construe an object that models the meaning of the variable domain an example of functor category is used.

For every T of $\mathcal{C}$ let

$$H_T(I) = \{h | h : I \to T\}$$

and if $f : B \to I$ in $\mathcal{C}$, let $H_T(f)$ be the map taking $h \in H_T(I)$ into $h \circ f \in H_T(B)$. It is easy to verify H_T is a contravariant functor.

Transactions. Let $g : T \to S$ in $\mathcal{C}$. There is a natural transformation $H_g : H_T \to H_S$. Every $h \in H_T(I)$ can be mapped to $g \circ h \in H_S(I)$. So defined mapping g determines a rectified idea of transaction.

Clones. The composite map for $f : B \to I$ takes $h \in H_T(I)$ *into* $g \circ h \circ f \in H_S(B)$. Thus, the individuals from $H_T(I)$ are f-cloned into $H_S(B)$.

It is easy to verify $H : \mathcal{C} \to \mathcal{S}^{\mathcal{C}^{op}}$ is a covariant functor, and $\mathcal{C}$ may be assumed to be c.c.c.

3.1.3 Functorial properties

Let functor H_T in $\mathcal{S}^{\mathcal{C}^{op}}$ be treated as a *variable domain*: (1) for every $I \in \mathcal{C}$ an associated domain $H_T(I)$ is the set; (2) the maps $f : B \to I$ in $\mathcal{C}$ give transitions from stage I to stage B.

Every transition clones elements in $H_T(I)$ into elements in $H_T(B)$ along the map f.

The verification of functorial properties of H_T is straightforward. The properties of the *restriction* come down to the following:

$$\begin{array}{rcl} h\rceil 1_I & = & (H_T(1_I))(h) \\ & = & h \circ 1_I \\ & = & h, \\ (h\rceil f)\rceil g & = & h\rceil(f \circ g), \end{array}$$

where $h\rceil f = (H_T(f))(h) = h \circ f$ is an abbreviation.

4 Fragment of a theory of types

Many possible theories of types are known, and the need is of getting down to some details of object-as-functor for types.

The domains A of $\mathcal{C}$ are associated to the *type symbols*, and they are basic types. The derived types are generated by constructions: **1** (empty product), $T \times S$ (cartesian product), $T \to S$ (functional space), $[T]$ (power type).

In the functor category an arbitrary type T is indicated as H_T, and an evaluation mapping $\|\cdot\|$ needs an additional parameter, so that $\|\cdot\|\cdot$. And this is an important stage to treat the functor category as an interpretation for a higher order theory.

4.1 Dynamics: further understanding via logic

The construction of a logical framework reflects the adopted object solutions.

4.1.1 Logical language

A language contains a supply of variables for every type. *Atomic formulae* are the equations:

- $x = y$, where $x,\ y$ are of the same type;
- $y = gx$, where g is a constant $g : T \to S$ of $\mathcal{C}$, x and y have the types T and S respectively;
- $z = [x, y]$, where $x,\ y$ of types $T,\ S$ respectively, z of type $T \times S$;
- $z = x(y)$, where x has type $T \to S$, y type T, z type S;
- $y \in x$, where y is of type T, x of type $[T]$.

Formulae Φ are generated as usually by the *connectives* and *quantifiers*.

4.1.2 Interpretation

Assume the following: A is a domain of $\mathcal{C}$, Φ is a formula, $\|\cdot\|$ is an evaluation of the non-bound variables of Φ.

An evaluation of the *variable* makes $\|\cdot\|$ relative to the domains of $\mathcal{C}$ (e.g., to A) and needs the explanation.

Visible objects are percepted by the observer via his machinery in spite of the doctrine of the predefined objects.

The events evolve from A to B. The inhabitants of the world A evolve, so they inhabit the world B. The world B contains the clones of A-inhabitants, and also some other inhabitants, if any.

$$A \xleftarrow{\quad f \quad} B$$

$$\|\bar{x}\|A \in H_T(A) \xrightarrow{H(f)} (H_T)_f \xrightarrow{\subseteq} H_T(B) \ni \|\bar{y}\|B$$

$$\|\bar{x}\|A = \|\bar{y}\|B = \|\bar{x}\|_f B \tag{1}$$

The evaluation of the atomic formulae is getting down to the case study (are given for atomic case).

Variables.

$\|x = y\|A \iff \|x\|A = \|y\|A$ **(Var)**

Constant function.

$\|y = gx\|A \iff \|y\|A = g \circ \|x\|A$ **(CFun)**

Ordered pair.

$\|z = [x, y]\|A \iff \|z\|A = [\|x\|A, \|y\|A]$ **(DPair)**

Application (variable function).

$\|z = x(y)\|A \iff \|z\|A = \|x\|_{1_A}A(\|y\|A)$ (ε)

Powerset.

$\|y \in x\|A \iff \|y\|A \in \|x\|_{1_A}A$ **(PSet(A))**
$\|y \in x\|B \iff \|y\|B \in \|x\|_{1_B}B$ **(PSet(B))**
$\|y \in x\|_f \iff \|y\|B \in \|x\|_f B$ **(PSet$_f$)**

4.1.3 Construction of concept

The notion of a 'concept' depends on a set of conditions and was studied under the various assumptions. The following matches an intuition for a 'variable domain'.

A notational remark. In the below $\|\cdot\|_{(t/y)}$ means the fixed evaluation where t matches y of the same type. The evaluation $\|\cdot\|_{(t/y)}\rceil f = \|\cdot\|_f$ matches $\|y\|_f$ with every relevant variable y. Any case the restriction $\rceil$ is superimposed to the functor H_T with T is the type of y.

Let concepts $C(A)$, $C(B)$, and C_f be the different restrictions of the H_T:

$C(A) = \{t \in H_T(A) \mid \|\Phi(y)\|_{1_A(t/y)}A\}$ **(Conc(A))**
$C(B) = \{t \in H_T(B) \mid \|\Phi(y)\|_{1_B(t/y)}B\}$ **(Conc(B))**
$C_f = \{t \in H_T(B) \mid \|\Phi(y)\|_{f(t/y)}B\}$ **(Conc$_f$)**

Their relationships correspond to the diagram:

$$\begin{tikzcd} A & B \arrow[l, "f"'] \\ C(A) \arrow[r, "C(f)"] & C_f \arrow[r, "\subseteq"] & C(B) \end{tikzcd}$$

(here: $C_{1_A} = C(A)$; $C_f \subseteq C(B)$; $C = H_T$)

4.2 Case study for variable domains

The 'transaction-clone' notion having been applied to the functor category $H : \mathcal{C} \to \mathcal{S}^{\mathcal{C}^{op}}$ has a benefit of explicate arrow-thinking. In the following family of diagrams the mapping $f : B \to A$ *clones* the individual from A into B. Besides that, the mapping $g : C \to D$ represents the *transition* (an explanatory system is of free choice):

general diagram:

$$\begin{tikzcd} & A & B \arrow[l, "f"'] \\ T & H_T(A) \arrow[r, "H_T(f)"] \arrow[d, "H_g(A)"'] & (H_T)_f \arrow[r, "\subseteq"] & H_T(B) \arrow[d, "H_g(B)"] \\ \mathcal{T} \arrow[from=u, "g"'] & H_{\mathcal{T}}(A) \arrow[r, "H_{\mathcal{T}}(f)"] & (H_{\mathcal{T}})_f \arrow[r, "\subseteq"] & H_{\mathcal{T}}(B) \end{tikzcd}$$

singular:

$$H_C(A) = \{h \mid h : A \to [C]\}$$
$$H_C(A)$$

f-cloned:

$$H_C(f) : H_C(A) \ni h \mapsto h \circ f \in H_C(B)$$

$$\begin{tikzcd} A & B \arrow[l, "f"'] \\ H_C(A) \arrow[r, "H_C(f)"] & H_C(B) \end{tikzcd}$$

non-cloned, g-transacted:

$$H_g : H_C(A) \ni h \mapsto g \circ h \in H_D(A)$$

$$\begin{tikzcd} & A \\ C \arrow[d, "g"'] & H_C(A) \arrow[d, "H_g(A)"'] \\ D & H_D(A) \end{tikzcd}$$

1_A-cloned, g-transacted:

$$H_D(1_A) \circ H_g : H_C(A) \ni h \mapsto g \circ h \circ 1_A \in H_D(A)$$

$$\begin{tikzcd} & A & A \arrow[l, "1_A"'] \\ C \arrow[d, "g"'] & H_C(A) \arrow[r, "H_C(1_A)"] \arrow[d, "H_g(A)"'] & H_C(A) \arrow[d, "H_g(A)"] \\ D & H_D(A) \arrow[r, "H_D(1_A)"] & H_D(A) \end{tikzcd}$$

f-cloned, g-transacted:

$$H_D(f) \circ H_g : H_C(A) \ni h \mapsto g \circ h \circ f \in H_D(B)$$

$$\begin{tikzcd} & A & B \arrow[l, "f"'] \\ C \arrow[d, "g"'] & H_C(A) \arrow[r, "H_C(f)"] \arrow[d, "H_g(A)"'] & H_C(B) \arrow[d, "H_g(B)"] \\ D & H_D(A) \arrow[r, "H_D(f)"] & H_D(B) \end{tikzcd}$$

f-cloned, non-transacted:

$$\begin{aligned} H_C(f) &: H_C(A) \ni h \mapsto h \circ f \in H_C(B) \\ H_D(f) &: H_D(A) \ni h \mapsto h \circ f \in H_D(B) \end{aligned}$$

$$\begin{tikzcd} & A & B \arrow[l, "f"'] \\ C & H_C(A) \arrow[r, "H_C(f)"] & H_C(B) \\ D & H_D(A_D(A) \arrow[r, "H_D(f)"] & H_D(B) \end{tikzcd}$$

The functorial properties of H_T come down to the case study given above.

4.3 Evaluation mapping

The functor category in use may enrich the intuition concerning an evaluation mapping. In particular, the diagram given below reflects

f-cloned evaluation mapping:

$$\begin{tikzcd}
A & B \arrow[l, "f"'] & \\
\|\Phi\|A \arrow[r, "\|\Phi\|f"] & \|\Phi\|_f \arrow[r, "\subseteq"] & \|\Phi\|B \\
\{y\} \arrow[r, "H_T(f)"] \arrow[d, "\in"'] & \{y \circ f\} \arrow[d, "\in"'] & t \arrow[d, "\in"'] \\
H_T(A) \arrow[r, "H_T(f)"] & (H_T)_f \arrow[r, "\subseteq"] & H_T(B)
\end{tikzcd}$$

Similarly, g-transacted, f-cloned evaluation mappings shown in Fig. 1.

(N.B. Possibly, Ψ may be equal to Φ; $y = u$, and $t = v$.) The interpretation of previous diagram depends on the evailable engineering machinery.

An advance in the representation may be achieved with the *concepts* $C_1,\ C_2$ corresponding to $\Phi,\ \Psi$ respectively.

The previous diagram is comprehenced to:

$$\begin{tikzcd}
 & A & B \arrow[l, "f"'] & \\
 & \{y\} \arrow[r, "C_1(f)"] \arrow[d, "\in"'] & \{y \circ f\} \arrow[d, "\in"'] & t \arrow[d, "\in"'] \\
C \arrow[d, "g"'] & C_1(A) \arrow[r, "C_1(f)"] \arrow[d, "H_g(A)"'] & C_{1f} \arrow[r, "\subseteq"] & C_1(B) \arrow[d, "H_g(B)"] \\
D & C_2(A) \arrow[r, "C_2(f)"] & C_{2f} \arrow[r, "\subseteq"] & C_2(B) \\
 & \{u\} \arrow[r, "C_2(f)"] \arrow[u, "\in"] & \{u \circ f\} \arrow[u, "\in"] & v \arrow[u, "\in"]
\end{tikzcd}$$

The only ‘transaction-clone’ dependencies are visible, so an explicit object is extracted.

Note in addition, that the concept-image of g-transacted, f-cloned evaluation mapping:

$$\begin{tikzcd}
 & A & B \arrow[l, "f"'] & \\
 & C_1(A) \arrow[r, "C_1(f)"] \arrow[d, "\subseteq"'] & C_{1f} \arrow[d, "\subseteq"'] & C_1(B) \arrow[d, "\subseteq"'] \\
C \arrow[d, "g"'] & H_C(A) \arrow[r, "H_C(f)"] \arrow[d, "H_g(A)"'] & (H_C)_f \arrow[r, "\subseteq"] & H_C(B) \arrow[d, "H_g(B)"] \\
D & H_D(A) \arrow[r, "H_D(f)"] & (H_D)_f \arrow[r, "\subseteq"] & H_D(B) \\
 & C_2(A) \arrow[r, "C_2(f)"] \arrow[u, "\subseteq"] & C_{2f} \arrow[u, "\subseteq"] & C_2(B) \arrow[u, "\subseteq"]
\end{tikzcd}$$

is in a harmony with the “logical” diagram in Fig. 1.

5 Extracting a computational background

In applications a theory of functions is based on some additional objects.

Applicator

$$\varepsilon_{BC} : (B \to C) \times B \to C$$

which applies function f to its argument x: $\varepsilon_{BC} : [f, x] \mapsto f(x)$.

Currying

$$\Lambda_{ABC} : (A \times B \to C) \to (A \to (B \to C))$$

which shifts variables.

More exactly, if $h : A \times B \to C$, then $\Lambda_{ABC} h : A \to (B \to C)$.

For $k : A \to (B \to C)$ and $h : A \times B \to C$ mapping Λ gives a correspondence. Equationally, it means

$$\begin{array}{rcl} \varepsilon\circ < (\Lambda h) \circ Fst, Snd > & = & h, \text{and} \\ \Lambda(\varepsilon\circ < k \circ Fst, Snd >) & = & k \end{array}$$

for the first projection Fst and second projection Snd:

$$Fst : A \times B \to A, \quad Snd : A \times B \to B.$$

Note, that the equation (ε) may be rewritten:

$$\|z\| = \varepsilon\circ < \|x\|_{1_A}, \|y\| >$$

Next step will be done to determine the meaning of an expression.

5.1 Meaning of expression

The goal is to determine the meaning of an expression $F(x)$, or Fx where F is the description of a function and x is a formal parameter. Thus, x is bound, or substitutional variable.

A treatment may be simplified with the λ-notations. The expression above is to be denoted as $\lambda x.yx$ where the description of a function F is associated to a variable y.

The meaning of a function depends on the meanings of its subparts $y,\ x,\ yx$. Those components, in turn, depend on the value of y.

5.1.1 Building an access

The values of the variables are available via *access functions* from an *environment*. The representation of an environment is given by the domains $D_y,\ D_x,\ \ldots$ which are the ranges of possible values of $y,\ x,\ \ldots$. The domains $D_y,\ D_x$ give the explicit part of an environment Env, and its implicit rest (not be detailed for the current consideration) is denoted by E:

$$Env = (E \times D_y) \times D_x$$

5.1.2 Case study

Atomic parts. An object $\lambda x.yx$ contains atoms $y,\ x$, and non-atomic part yx:

$$\|y\| : Env \to D_y, \qquad \|x\| : Env \to D_x.$$

Non-atomic parts. A non-atomic part yx is evaluated as follows:

the pair $< \|y\|, \|x\| >$ is composed, and

$$< \|y\|, \|x\| >: Env \to D_y \times D_x;$$

the metaoperator ε is applied to the pair:

$$\varepsilon\circ < \|y\|, \|x\| > .$$

To exemplify let $D_y = (D_x \to D'_y)$; thus, $\varepsilon : (D'_y)^{D_x} \times D_x \to D'_y$ is determined by $\varepsilon[u, v] = u(v) = uv$, and D'_y is the range for $\|yx\|, i.e.$

$$\|yx\| = \varepsilon\circ < \|y\|, \|x\| >: Env \to D'_y$$

$$
\begin{tikzcd}
 & A & B \arrow[l, "f"'] & \\
 & \|\Phi\|A \arrow[r, "\|\Phi\|f"] & \|\Phi\|_f \arrow[r, "\subseteq"] & \|\Phi\|B \\
 & \{y\} \arrow[r, "H_C(f)"] \arrow[d, "\in"'] & \{y \circ f\} \arrow[d, "\in"'] & t \arrow[d, "\in"'] \\
C \arrow[d, "g"'] & H_C(A) \arrow[r, "H_C(f)"] \arrow[d, "H_g(A)"'] & (H_C)_f \arrow[r, "\subseteq"] & H_C(B) \arrow[d, "H_g(B)"] \\
D & H_D(A) \arrow[r, "H_D(f)"] & (H_D)_f \arrow[r, "\subseteq"] & H_D(B) \\
 & \{g \circ y\} \arrow[u, "\in"] \arrow[r, "H_D(f)"] \arrow[d, equal] & \{(g \circ y) \circ f\} \arrow[u, "\in"] \arrow[d, equal] & v \arrow[u, "\in"] \arrow[d, equal] \\
 & \{u\} & \{u \circ f\} & v \\
 & \|\Psi\|A \arrow[r, "\|\Psi\|f"] & \|\Psi\|_f \arrow[r, "\subseteq"] & \|\Psi\|B
\end{tikzcd}
$$

Figure 1: g-transacted, f-cloned evaluation mapping

5.1.3 Substitution

The expression $\lambda x.yx$ contains y (free variable) and does not contain x (bound, or substitutional variable; x may be renamed, if needed). To take into account this reason the modified environment $Env \times D_x$ is temporary generated to support the substitution $Subst_x$:

$$Subst_x : Env \times D_x \to Env,$$

where for $i \in Env,\ h' \in D_x$ the result is

$$Subst_x[i, h'] = i_{(h'/x)}.$$

It means that substitution $Subst_x$ for every ordered pair $[i, h']$ gives a correspondent environment $i_{(h'/x)}$ which differs from i exclusively in a point x (x is substituted by h').

An *access function* for $Subst_x$ is generated by the equation:

$$Subst_x =< Fst \circ Fst, Snd >$$

5.1.4 Composition

An observation is as follows: the function $\|yx\|$ and $Subst_x$ are composed:

$$\|yx\| \circ Subst_x : Env \times D_x \to D'_y$$

The meaning of $\lambda x.yx$ depends on Env for y (y has a free occurrence in $\lambda x.yx$, and x is bound). Thus, $\|\lambda x.yx\|$ is a function that associate to y the function associating yx to x. A type consideration gives:

$$\|\lambda x.yx\| : Env \to (D'_y)^{D_x}$$

To the contrast $\|yx\|$ is a function *from* $(E \times D_y)$ and D_x:

$$\|yx\| : (E \times D_y) \times D_x \to D'_y$$

Some difficulties exist to establish the correspondence between meanings $\|\lambda x.yx\|,\ \|yx\|,\ Subst_x$.

$$
\begin{tikzcd}
Env \times D_x \arrow[r, "g"] \arrow[d, "\hat{g} \times id_{D_x}"'] & D'_y \\
(D'_y)^{D_x} \times D_x \arrow[ur, "\varepsilon"'] &
\end{tikzcd}
$$

Figure 2: Commutative diagram for $\|yx\| \circ Subst_x = g$

5.2 Correspondence of the meanings

Let $\|yx\| \circ Subst_x = g$, and

$$g([i, h']) \in D'_y$$

for $g : Env \times D_x \to D'_y$.

For $i \in Env$ and every $h' \in D_x$ the function g is determined by $g_i(h') = g([i, h'])$. Now the function $\hat{g}$ is defined by the equation $\hat{g}(i) = g_i$ for $h' \in D_x$. For arbitrary pair $[i, h'] \in Env \times D_x$ the equation

$$\varepsilon[\hat{g}(i), h'] = g_i(h') = g([i, h'])$$

is valid.

Note, that an operation $\hat{}$ generates the additional metaoperator Λ of *currying*:

$$(\Lambda(g)(i))(h') = g([i, h'])$$

Hence, a curried version of $g = \|yx\| \circ Subst_x$ is exactly $\|\lambda x.yx\|$, and finally the needed equation is obtained:

$$\|\lambda x.yx\| = \Lambda(\|yx\| \circ Subst_x)$$

Let to summarize the above reasons in Fig. 2. In this figure the following notations are used:

$$
\begin{array}{l}
g : Env \times D_x \to D'_y,\ i \in Env,\ h' \in D_x \\
g_i : D_x \to D'_y,\ g_i(h') = g([i, h']),\ \hat{g}(i) = g_i \\
[i, h'] \in Env \times D_x \\
\varepsilon([\hat{g}(i), h']) = g_i(h') = g([i, h'])
\end{array}
$$

At last, an access function for $\|\lambda x.yx\|$ is generated in accordance with the equation:

$$\|\lambda x.yx\| = \Lambda((\varepsilon\circ < Snd \circ Fst, Snd >)\circ < Fst \circ Fst, Snd >)$$

It is easy to verify an optimized version of the access function:

$$\|\lambda x.yx\| = \Lambda(\varepsilon\circ < Snd \circ Fst \circ Fst, Snd >)$$

from the properties of pairs $< \cdot, \cdot >$ and composition.

5.3 Examples

Some examples of computation are briefly given below.

Constant c.

1	$\|c\|i =$	$i \in Env,\ c' \in \{c\}$ for singleton $\{c\}$
2	$= \|0!\|[i, c']$	$\|0!\|$ – a.f. to $\{c\}$ in Env
3	$= Snd[i, c']$	
4	$= c'(= c)$	

Variable x. The evaluation of a variable gives one of the possible atomic cases. The abbreviations F for Fst and S for Snd are used.

1	$\|x\|i =$	$i \in Env$
	Generation of a.f. :	
	$= \|0!\|i_{(h'/x)}$	$h' \in D_x$; $\|0!\|$ – a.f. to D_x in Env
	$= \|0!\|[i, h']$	$Env = E \times D_x$
	$= S[i, h']$	
	$= h'$	
2	$= (\|x\| \circ Subst_x)[i, h']$	$Subst_x : Env \times D_x \to Env$; $Subst_x =< F \circ F, S >$
3	$= (S\circ < F \circ F, S >)[i, h']$	Replace by a.f.
4	$= S(< F \circ F, S > [i, h'])$	Substitution
5	$= S[F(i), h']$	a.f.
6	$= h'$	$h' \in D_x$

Identity transformation. The evaluation of an identity transformation gives a clear separation of access functions (a.f.) and substitution.

1	$\|(\lambda x.x)h\|i=$	$i \in Env$
2	$= \|\lambda x.x\|ih'$	$h' \in D_x$
	Generation of direct access:	
	$= \Lambda\|0!\|ih'$	$\|0!\|$ – a.f. to D_x in Env, x – bound variable, $Env = E \times D_x$
	$= S[i, h']$	$[i, h'] \in Env \times D_x$
	$= h'$	
3	$= \Lambda(\|x\| \circ Subst_x)ih'$	Using a.f.
4	$= \Lambda(S\circ < F \circ F, S >)ih'$	$Subst_x : Env \times D_x \to Env$; $Subst_x =< F \circ F, S >$
5	$= (S\circ < F \circ F, S >)[i, h']$	$[i, h'] \in Env \times D_x$
6	$= S(< F \circ F, S > [i, h'])$	Substitution
7	$= S[F(i), h']$	a.f.
8	h'	

Compound evaluation.

1	$\|(\lambda x.fx)h\|i =$	$i \in Env$
2	$= \|(\lambda x.fx)\|ih'$	$h' \in D_x$
	Generation of access:	
	$= \Lambda\|f0!\|ih'$	$\|0!\|$ – a.f. to D_x in Env; $Env = (E \times D_f) \times D_x$
	$= \|f0!\|[i, h']$	$[i, h'] \in Env \times D_x$
	$= (\varepsilon\circ < \|f\|, S >)[i, h']$	$S(i) \in D_x$
	$= \varepsilon[\|f\|[i, h'], h']$	$\|f\|$ – a.f. to D_f in $Env \times D_x$, i.e. $\|f\| = S \circ F \circ F$
	$= \varepsilon[(S \circ F)(i), h']$	$(S \circ F)(i) \in D_f$
	$= f'h'$	
3	$= \Lambda(\|fx\| \circ Subst_x)ih'$	Replace by a.f.
4	$= \Lambda((\varepsilon\circ < S \circ F, S >)\circ < F \circ F, S >)ih'$ (for $Subst_x =< F \circ F, S >,\ \ Subst_x : Env \times D_x \to Env$)	
5	$= (\varepsilon\circ < S \circ F, S > \circ < F \circ F, S >)[i, h']$ (for $[i, h'] \in Env \times D_x$)	
6	$= (\varepsilon\circ < S \circ F, S >)(< F \circ F, S > [i, h'])$ (Substitution)	
7	$= (\varepsilon\circ < S \circ F, S >)[F(i), h']$ (for $F(i) \in E \times D_f,\ \ h' \in D_x$)	
8	$= \varepsilon[< S \circ F, S > [F(i), h']]$	a.f.
9	$= \varepsilon[(S \circ F)(i), h']$	ε;
	$((S \circ F)(i)$ extracts value of $D_f)$	
10	$= \varepsilon[f', h']$	
11	$= f'h'$	

$$\begin{tikzcd}
I \times T \arrow[r, "\|\Phi\|[\cdot,\cdot]"] \arrow[d, "\Lambda\|\Phi\| \cdot \times id_T"'] & {[\ ]} \\
{[T] \times T} \arrow[ur, "\varepsilon"] &
\end{tikzcd}$$

Figure 3: Evaluation in c.c.c.

5.4 Advanced examples

The additional examples of generalized nature involve more complicated objects.

Evaluation of formula. This kind of object has the following equations:

$\|\Phi\|[i, hi]$	$= \Lambda\|\Phi\|(Fst[i, hi])(Snd[i, hi])$
$\Lambda\|\Phi\|i(hi)$	$= \Lambda\|\Phi\|(Fst[i, hi])(Snd[i, hi])$
	$= \varepsilon[\Lambda\|\Phi\|(Fst[i, hi]), (Snd[i, hi])]$
	$= (\varepsilon\circ < \Lambda\|\Phi\| \circ Fst, id \circ Snd >)[i, hi]$
	$= \|\Phi\|[i, hi]$
$\|\Phi\|$	$= \varepsilon\circ < \Lambda\|\Phi\| \circ Fst, id \circ Snd >$

An abbreviation

$$\|\Phi\| = \|\Phi(x)\| \circ Subst_x$$

is used if there is no ambiguity. Hereafter T is a type of substitutional variable x, and environment Env is renamed by I.

Evaluation in c.c.c. The diagram in Fig. 3 illustrates an idea.

- $\Lambda\|\Phi\| : I \to [T]$;
- $\|\Phi\| : I \times T \to [\ ]$;
- For $i \in I$ and $hi \in T$ an evaluation $\varepsilon[\Lambda\|\Phi\|i, id_T(hi)]$ generates the truth values from [ ].

Individuals in c.c.c. A correspondence of the distinct forms of individuals shows their similarities.

$\boxed{R \rightsquigarrow h_R.}$ Given the relation $R \subseteq I \times T$ a function $h_R : I \to [T]$ is determined by the equality $h_R(i) = \{h' \mid h' \in T \wedge iRh'\}$. In fact, this defines the correspondence $R \rightsquigarrow h_R$. ■

$$\begin{tikzcd}
R \arrow[r, hook] \arrow[d, "g"'] & I \times T \arrow[d, "h_R \times id_T"] \\
\in_T \arrow[r, hook] & [T] \times T
\end{tikzcd}$$

Figure 4: Variants of individuals

$$\begin{tikzcd}
R \arrow[r, hook] \arrow[d, "C \cdot \times id_T"'] & I \times T \arrow[r, "\|\Phi\|[\cdot,\cdot]"] \arrow[d, "\Lambda\|\Phi\| \cdot \times id_T"'] & [\ ] \\
\in_T \arrow[r, hook] & [T] \times T \arrow[ur, "\varepsilon"] &
\end{tikzcd}$$

Figure 5: Computational properties

$$\begin{tikzcd}
I \times T \arrow[r, "Snd"] \arrow[d, "\Lambda(f \circ Snd) \cdot \times id_T"'] & T \arrow[r, "f"] & S \\
S^T \times T \arrow[urr, "\varepsilon"] & &
\end{tikzcd}$$

Figure 6: Built-in function f

$\boxed{h \rightsquigarrow R_h.}$ Given the sets $I,\ T$ the bijection between functions from I into $[T]$ and the relations from I to T is defined as follows. The function $h : I \to [T]$ determines the relation $R_h \subseteq I \times T$ by the biconditional $iR_h h'' \iff h'' \in h(i)$ for $i \in I$ and $h'' \in T$. ■

$\boxed{\in_T.}$ The domain $\in_T = \{< U, h' >| \ U \subseteq T,\ h' \in T \wedge h' \in U\}$ is the relation containing all the necessary information concerning element-subset inclusions. The following biconditionals are valid:

$$[i, h'] \in R \iff h' \in h_R(i) \iff [h_R(i), h'] \in \in_T$$

Hence, R is a domain and $\in_T$ is a range for mapping $h_R \times 1_T$ where $h_R \times 1_T : [i, h'] \mapsto [h_R(i), h']$. ■

The diagram in Fig. 4 reflects the ideas given above.

Here: g is an R-restricted version of $h_R \times id_T$. Note that all of this is quite elementary.

Computational properties of the individuals. The combined diagram in Fig. 5 establishes not so evident correspondences. What is important that the functor $C \cdot \times id_T$ includes as a left counterpart the mapping $C\cdot : I \to [T]$. This mapping is relative to relation R and this relation is induced by the evaluation of the restriction Φ.

In particular, a built-in function for the given (and evaluated) argument in a category results in the diagram in Fig. 6.

A free variable is evaluated according to the diagram in Fig. 7.

A simplified example of computation (note that both the operands are to be embedded into the computational environment) like

$$+[[2/x_1]x_1, [3/x_2]x_2]$$

is in Fig. 8. The entry points for the computations of the distinct operands are, in general, independent. Thus, both the left-part and

$$\begin{tikzcd}
I \times T \arrow[r, "Snd"] \arrow[d, "\Lambda(Snd) \cdot \times id_T"'] & T \\
T^T \times T \arrow[ur, "\varepsilon"] &
\end{tikzcd}$$

Figure 7: Free variable

right-part computations are to be started at the same 'moment'. An additional mappings can_T of canonical embedding of the constants are also used.

The more exact correspondences are as follow:

$$\begin{array}{lcl}
R & = & \{[i, h'] \mid \|\Phi\|[i, h'] = 1\} \\
\Lambda\|\Phi\|i & \in & [T] \\
\Lambda\|\Phi\| & : & I \to [T] \\
C(\{i\}) & = & \{h(i) \mid \Lambda\|\Phi\|i(hi) = 1\} \\
C(I) & = & \{h \mid \|\Phi(x)\|_{[h/x]} : I \to [T]\}
\end{array}$$

(here: $x, h : I \to [T]$, so $h(i) \subseteq T$; x is a free variable.)

6 Conclusions

A common object technique shared by distinct 'dimensions' – logical, categorical, and computational is outlined.

Important: The notion of a variable domain gives a sound ground of the communication analysis (see, e.g.: [WW94], [Jac92]). As may be shown they generate the specific diagrams to consider the variety of transition effects.

Open discussion: The questions arise:
1. Is the language of categories adequate to database dynamics even though the object-oriented approach successively applied?

References

[Bro94] M. L. Brodie. Interoperable information systems: Motivations, challenges, approaches, and status. In *Second International Conference on Cooperative Information Systems, CoopIS-94, Tutorial Notes. May 17–20, 1994*, Toronto, Ontario, Canada, May 1994.

[Bro95] M.L. Brodie. *Interoperable Information Systems: Motivations, Challenges, Approaches, and Status*. Russian Basic Research Foundation, Moscow, Russia, April, 6-7, 1995.

[CCM85] G. Cousineau, P.-L. Curien, and M. Mauny. The categorical abstract machine. In *Functional programming languages computer architecture*, volume 201 of *Lecture Notes in Computer Science*, pages 50–64. Heidelberg, Springer-Verlag, 1985. This is a detaled paper on programming in a category-style.

[EGS91] H.-D. Ehrich, M. Gogola, and A. Sernadas. A categorial theory of objects as observed processes. In J.W. deBakker et. al., editor, *Proceedings of the REX/FOOL School/Workshop*, volume 489 of *Lecture Notes in Computer Science*, pages 203–228. Berlin, Heidelberg, New York, Springer Verlag, 1991.

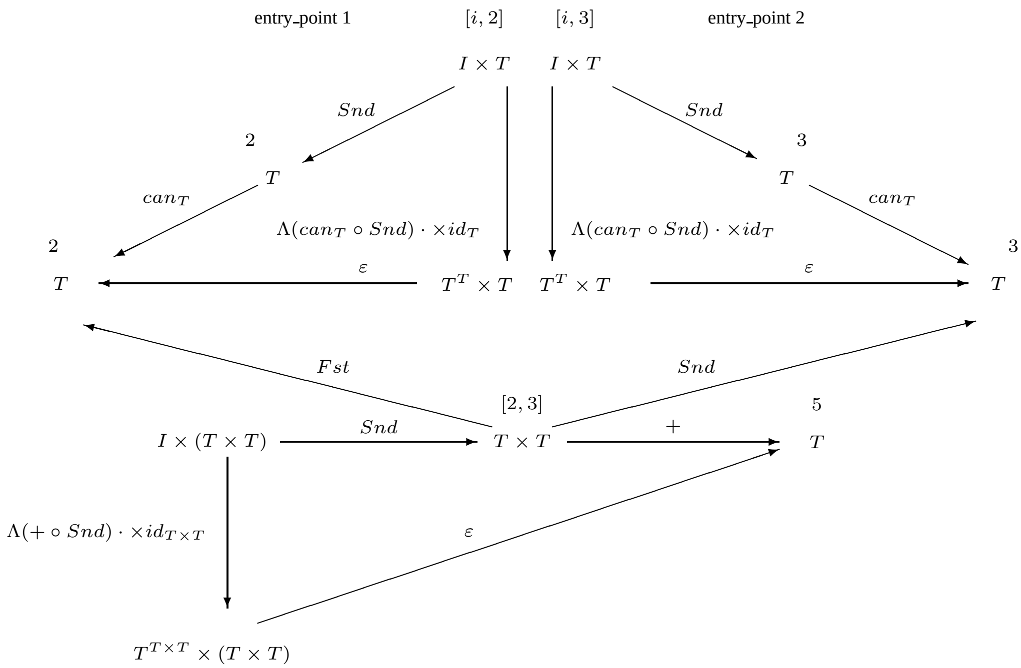

Figure 8: An example of computation $+[2, 3]$

[EGS93] H.-D. Ehrich, M. Gogola, and A. Sernadas. Objects and their specification. In M. Bidoit and C. Choppy, editors, *Recent Trends in Data Type Specification. 8th Workshop on Specification of Abstract Data Types joint with the 3rd COMPASS Workshop, Dourdan, France, August 26–30, 1991, Selected Papers*, volume 655 of *Lecture Notes in Computer Science*, pages 40–65. Berlin, Heidelberg, New York, Springer Verlag, 1993.

[Gab93] P. Gabriel. The object-based specification language Π: Concepts, syntax, and semantics. In M. Bidoit and C. Choppy, editors, *Recent Trends in Data Type Specification. 8th Workshop on Specification of Abstract Data Types joint with the 3rd COMPASS Workshop, Dourdan, France, August 26–30, 1991, Selected Papers*, volume 655 of *Lecture Notes in Computer Science*, pages 254–270. Berlin, Heidelberg, New York, Springer Verlag, 1993.

[Gog89] J. Goguen. A categorical manifesto. Technical report PRG-72, Programming Research Group, Oxrofd University, March 1989.

[HC89] F. Heyes and D. Coleman. Objects and inheritance: an algebraic view. Technical Memo, HP labs, Information Management Lab, Bristol, 1989.

[Jac92] I. Jacobson. *Object-Oriented Software Engineering: A Use-Case Driven Approach*. Addison-Wesley, Reading, Massachusetts, 1992.

[Law75] F.W. Lawvere. Continuously variable sets: algebraic geometry = geometric logic. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73*, pages 135–156. North Holland, Amsterdam, 1975.

[NR95] D.A. Nelson and B.N. Rossiter. Prototyping a categorical database in P/FDM. In L.A. Kalinichenko, editor, *Proceedings of the workshop on Advances in DataBases and Information Systems, ADBIS'95*, page this volume. Russian Academy of Sciences, Institute for Problems of Informatics, 1995.

[Sco80] D.S. Scott. Relating theories of the λ-calculus. In J. Hinhley and J. Seldin, editors, *To H.B. Curry: Essays on combinatory logic, lambda calculus and formalism*, pages 403–450. New York and London, Academic Press, 1980.

[Wol93] V.E. Wolfengagen. Computational aspects of data objects. In *Proceedings of the workshop on Advances in DataBase and Information Systems, ADBIS'93, May 11-14, Moscow, 1993*, Moscow, May 1993.

[WW94] Y. Wand and C. C. Woo. Object oriented analysis of organizational activities: A CoopIS tutorial. In *Second International Conference on Cooperative Information Systems, CoopIS-94, Tutorial Notes. May 17–20, 1994*, Toronto, Ontario, Canada, May 1994.