Proceedings of the 7th International Conference on Applied Informatics
Eger, Hungary, January 28–31, 2007. Vol. 1. pp. 325–334.

Container model*

Zsolt Hernáth

Dept. of Information Systems,
Fac. of Informatics, Eötvös Loránd University, Budapest
e-mail: hernath@ullman.inf.elte.hu

Abstract

Considering documentation systems like HTML, Latex or MS Word a document consists of a not necessarily contiguous sequence of characters and structural, probably also typographical mark-ups. Removing mark-ups produces a plain text. Removing punctuation and also whitespaces a sequence of non-whitespace characters remains. Inserting mark-ups, punctuation or whitespace characters into a plain text or pure charcter string a kind of document or structured text arises. We face some analogue also in programming: e.g. the complex value of a particular C-structure typed variable in the executable appears as a contiguous sequence of bytes which holds information neither on the C -structure nor its components' types.

In Container Model two structural operations namely composing finite-length tupples and composing finite-length sequences, further on a set $\mathcal{U}$ whose elements are all finite-length tupples and sequences recursively composed from non-negative interegers are considered. In addition, we consider finite-length sequences of bytes, called raw data. Any non-negative integer is mapped to shortest raw datum that represents its binary scaled value, and any particular $u \in \mathcal{U}$ is mapped to the raw datum that is composed from the raw data of its components by putting them after each other in the order of apperance. The mapping above induces a partition over $\mathcal{U}$ whose elements are called containers. An arbitrary structure-expression of a raw datum using the srtuctural operations above identifies some subset of the container the raw datum identifies. Correlations between relationships among structure-expressions and those among the corresponding container subsets are detected.

Keywords: raw data, complex value, data model

MSC: 68P99 data model

*Supported by the Hungarian Ministry of Education under Grant FKFP0018/2002.

1. Introductuction

This paper brings data, considered as computer-processable codes of notions or objects of our real world, into focus. Data' real content, i.e. data semantics is revealed by the all-time model the components of that, and even relationships between them data actually code. Data semantics, in general, may be – and is actually viewed through three kinds of glasses: structural properties of data occurrences, the operational view, i.e. the data processing made by computer programs, and displaying data as real world notions or objects, usually also done by computer programs.

Coherence between codes' structural properties and their real content can be in any area detected, even though, no substantive general rule for that is not yet recovered. Binary images of executables executed by (today still) von Neumann's Principle based computers look as single contiguous byte sequences. The for us unvisible structural properties of those is given by the continually changed values of registers of CPU.

1.1. Problem

Consider some finite-length byte sequence, and suppose bytes and particular contiguous sequences of bytes (e.g. words) codes real world's things, and the total codes some complex real world's objects. Such objects may be some complex values of some structure typed variables in C programs, more exactly, the corresponding part of their executables. Considering the latter, it is obvious that chopping up some finite-length byte sequence to smaller units of particular lengths, different kind of structuring of data may be obtained, and each of them, according to C scalar types, may occasionally come from more than one different C structure type. In connection of the above, one may put the question if there is any relationship between the set of different C structures yielding the same binary complex value (i.e. a given byte sequence) and the set of all possible different dismemberments of the byte sequence in question. To examine the problem above, a formal algebraic model is set up.

1.2. Base concepts

Definition 1.1. Let $\mathbf{\Gamma}$ be a recursively enumerable set, and $\mathcal{H}$ be a finite set of operations that organize things into particular structures. Elements of $\mathbf{\Gamma}$ and $\mathcal{H}$ are called individuals and contacts, respectively. Each $H \in \mathcal{H}$ groups finite number of things declaring what is called H-contact between those, named H-community. Suppose, elements of $\mathcal{H}$ are strictly non-commutative, and non-associative operations[1] Let $\mathbf{h}$ denote a mapping $\mathbf{h} : \mathcal{H} \to 2^{\mathbf{\Gamma} \cup \mathcal{H}}$, called contact-rule. Introducing the sets $\Gamma_{\mathbf{h}(H)} = \mathbf{h}(H) \cap \mathbf{\Gamma}$, and $\mathcal{H}_{\mathbf{h}(H)} = \mathbf{h}(H) \cap \mathcal{H}$, contact-rule $\mathbf{h}$ is a mapping

[1] Operation like e.g. "composing a set of things" is out of our consideration here.

such that $1 > |\Gamma_{\mathbf{h}(H)}|$ and $H \in \mathcal{H}_{\mathbf{h}(H)}$. Rule $\mathbf{h}$ defines for $\forall H \in \mathcal{H}$ the set of individuals ($\Gamma_{\mathbf{h}(H)}$), and the set of contacts ($\mathcal{H}_{\mathbf{h}(H)}$) the communities established by elements from the latter together with individuals from $\Gamma_{\mathbf{h}(H)}$ may constitute legal (i.e. allowable by $\mathbf{h}$) H-communities. For $\forall H \in \mathcal{H}$ the set of all H-communities are denoted by $\mathcal{L}_\Gamma^H$. For any $c_H \in \mathcal{L}_\Gamma^H$, $|c_H|$ denotes the number of things groupped by H, and for each $1 \leqslant k \leqslant |c_H|$ integer, $c_H[k]$ denotes the k^{th} member of the community, regardless whether $c_H[k] \in \mathbf{\Gamma}$, or $c_H[k]$ is some legal $H^{'}$-community, for some $H^{'} \in \mathcal{H}_{\mathbf{h}(H)}$. The 3-tuple $\mathcal{A} = (\mathbf{\Gamma}, \mathcal{H}, \mathbf{h})$ is named community-autonomy $\mathcal{A}$. The set $\mathcal{U}_\mathcal{A} = \{\emptyset\} \cup \mathbf{\Gamma} \cup \cup\{\mathcal{L}_\Gamma^H : \ H \in \mathcal{H}\}$ is called $\mathcal{A}$-autonomous universe or briefly $\mathcal{A}$-universe.

Definition 1.2. Introducing the set $B = \{d_\emptyset, 00000000, 00000001, \ldots, 11111111\}$, let $<\mathbf{dom};>$ denote the free semi-group over generator set B. Elements of B but $d_\emptyset$ are called bytes, $d_\emptyset$ is called "empty-byte" (the byte sequence of length 0), and elements of **dom** are called raw data. As seen above, no particular notation is introduced for the semi-group operation called multiplying. For some data $d_{i_0}, \ldots, d_{i_k} \in \mathbf{dom}$, the product of them in the given order is denoted by $\prod_{j=0}^{k} d_{i_j}$. For any $d \in \mathbf{dom}$, the length of d is the number of its factors from the set $B - \{d_\emptyset\}$, and denoted by $||\mathrm{d}||$.

Definition 1.3. Let $\mathcal{A} = (\mathbf{\Gamma}, \mathcal{H}, \mathbf{h})$ be an autonomy, and $\mathcal{U}_\mathcal{A}$ the corresponding $\mathcal{A}$-universe. Let ω denote the set of non-negative integers, and let $s : \omega \to \mathbf{\Gamma}$ be a particular enumeration of set $\mathbf{\Gamma}$. Let $t : \omega \to \mathbf{dom}$ denote the bijection that to each non-negative integer n assigns the shortest byte sequence that, as a binary scaled whole number, has a value of n. Let $\tau : \mathcal{U}_\mathcal{A} \to \mathbf{dom}$ be defined for $\forall u \in \mathcal{U}_\mathcal{A}$ as

$$\tau(u) = \begin{cases} d_\emptyset & \text{if } u = \emptyset \\ t(s^{-1}(u)) & \text{if } u \in \mathbf{\Gamma} \\ \prod_{k=1}^{|u|} \tau(u[k]) & \text{if } u \in \bigcup_{H \in \mathcal{H}} \mathcal{L}_\Gamma^H \end{cases}$$

Definition 1.4. Elements of the set $\chi_{\mathbf{dom}} = \{\{u \in \mathcal{U}_\mathcal{A} : \tau(u) = d\} : d \in \mathbf{dom}\}$ are called *containers* of $\mathcal{U}_\mathcal{A}$. Introducing the mapping $\vartheta : \mathbf{dom} \to \chi_{\mathbf{dom}}$ for each $u \in \mathcal{U}_\mathcal{A}$ as $\vartheta(d) = \{u \in \mathcal{U}_\mathcal{A} : \ \tau(u) = d\}$, the container coded by $d \in \mathbf{dom}$ is referred as $\vartheta(d)$.

2. Structure semantics

To introduce structure-semantics, a very simple and minimal autonomy and the corresponding universe, denoted by $\mathcal{U}$, is established here. From now on, as for $\mathbf{\Gamma}$, the set ω, and as for the elements of $\mathcal{H}$, operations that compose finite length sequences resp. finite length tuples are taken. Let contact-rule $\mathbf{h}$ be the "all-is-allowed" regulation, i.e. any tuples resp. any sequences may contain arbitrary

elements from the set $\omega \cup \{\emptyset\}$ and also finite length tuples resp. finite length sequences recursively composed from those. Tuples resp. sequences of $\mathcal{U}$ are denoted $(u_0, \ldots, u_k)$ resp. $\{u_0, \ldots, u_r\}$.

Definition 2.1. For arbitrary $(1 <)r \in \omega$ and for containers $U_1, \ldots, U_r \in \chi_{\mathbf{dom}}$, the product of them is defined as $\prod_1^r U_i = \vartheta(\prod_1^r \vartheta^{-1}(U_i))$.

Proposition 2.2. Set $\chi_{\mathbf{dom}}$ together with the above defined container multiplying constitutes a semi-group $< \chi_{\mathbf{dom}}; \cdot >$ which is isomorphic with semi-group $< \mathbf{dom}; >$.

2.1. Structure semantics expressions

To structuring raw data, we introduce a parenthesizing convention. Parenthesizing has two roles. One is to denote which atomic factors (i.e. elements of **dom** that are not products of factors) are considered constituting a particular raw datum, and the other one is to indicate into which structure (finite-length tuple or finite-length sequence) factors are grouped. We will call any post-parenthesized datum $d \in \mathbf{dom}$ a structure expression of d, if the following is kept.

(1) Each atomic factor is parenthesized in one depth. This is to define atomic factors of d.

(2) Any product is always parenthesized in one resp. two depth that indicates that its factors (either atomic factors or some products recursively built from atomic factors) constitute a finite-length tuple resp. finite-length sequence.

(3) A parenthesis containing nothing represents the atomic factor $(d_\emptyset)$. No further redudant parenthesizing but those occuring in (1)-(3) are allowed and/or obliged.

Since no one-component-tuples as well as no one-member-sequences are formally contained by $\mathcal{U}$, the parenthesizing convention introduced above is unambiguous. A post-parenthesized raw datum may so be considered as the code of a subset of elements of a container product constituting particular structure indicated by parenthesizng, i.e. a particular subset of $\vartheta(d)$. Referring to raw data as hexadecimal scaled integers, let us consider e.g. parenthesized expressions $((0a)(ac5cde))$ and $(((ac)(ac5cde)))$ of datum $0aac5cde \in \mathbf{dom}$, which code the sets

$$\{(u, v) \in \vartheta(0aac5cde) : u \in \vartheta(0a), v \in \vartheta(ac5cde)\} \text{ and}$$
$$\{\{u, v\} \in \vartheta(0aac5cde) : u \in \vartheta(0a), v \in \vartheta(ac5cde)\},$$

respectively.

Definition 2.3. For arbitrary $d \in \mathbf{dom}$, the set of its all structure-semantics expressions are denoted by d^K. For each $\mathcal{K} \in d^K$, the associated subset of container $\vartheta(d)$ is denoted by $\mathcal{K}(d)$, and called the structure-semantics of raw datum d given by $\mathcal{K}$. The set $\sigma_d = \{\mathcal{K}(d) \in 2^{\vartheta(d)} : \mathcal{K} \in d^K\}$ is called the structure-semantics domain of raw datum d.

Definition 2.4. Let $d \in \mathbf{dom}$ arbitrary raw datum and $\mathcal{K}, \mathcal{K}^{'} \in d^K$. $\mathcal{K}^{'}$ is said finer than $\mathcal{K}$, if $\mathcal{K}^{'}$ can be obtained from $\mathcal{K}$ by replacing one atomic factor or more together with their one depth parenthesis by an arbitrary structure semantics expression of the factor(s) considered. If $\mathcal{K}^{'}$ is finer than $\mathcal{K}$ than $\mathcal{K}$ is coarser than $\mathcal{K}^{'}$. The previous resp. latter is denoted by $\mathcal{K}^{'} \preceq \mathcal{K}$ resp. $\mathcal{K} \succeq \mathcal{K}^{'}$.

Lemma 2.5. *For arbitrary $d \in \mathbf{dom}$ relation $\preceq$ is a partial order over set d^K. The poset $(d^K, \preceq)$ contain no least element. The greatest element is expression (d).*

A possible non-trivial refinement of expression $((0a)(ac5cde))$ is the expression $((0a)(((ac)(5c)(de))))$ that is also shown by Figure 1 using graph representation instead of algebraic one. Figure 2 shows two structure-semantics expression, for those the greatest lower bound (GLB) exists. Graph representation of structure-semantics expressions are ordered labeled graphs. Edge and node labels represent legal kind of structuring and elements of **dom**, respectively.

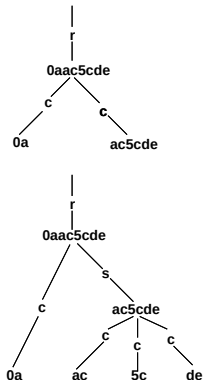

Figure 1: Expression ((0a)(ac5cde)) and its refinement ((0a)(((ac)(5c)(de))))

Lemma 2.6. *For arbitrary $d \in \mathbf{dom}$, and $\mathcal{K}, \mathcal{K}^{'} \in d^K$, the greatest lower bound $\mathcal{K} \bigwedge \mathcal{K}^{'}$ exists iff $\mathcal{K}(d) \cap \mathcal{K}^{'}(d) \neq \emptyset$.*

Definition 2.7. For arbitrary $d \in \mathbf{dom}$, expressions $\mathcal{K}, \mathcal{K}^{'} \in d^K$ are said independent, if $\mathcal{K}(d) \cap \mathcal{K}^{'}(d) = \emptyset$.

Theorem 2.8. *For arbitrary $d \in \mathbf{dom}$, $(d^K, \preceq, \bigvee)$ resp. $< d^K; \bigvee >$ constitutes a join semi-lattice resp. idempotent and commutative semi-group containing unit element expression (d).*

Definition 2.9. Any $d \in \mathbf{dom}$, may be considered a bijection $d : d^K \to \sigma_d$ defined for each $K \in d^K$ as $d(\mathcal{K}) = \mathcal{K}(d)$. Bijection d induces a operation on set σ_d. The induced operation is denoted by $\sqcup$, and for arbitrary $K_1, K_2 \in d^K$, defined as $d(K_1) \sqcup d(K_2) = d(K_1 \bigvee K_2)$.

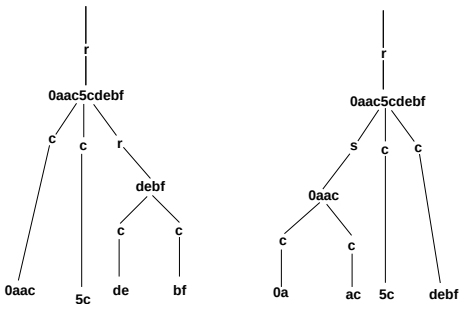

Figure 2: Expressions refining different factors of each other guarantees the existence of their GLB

Theorem 2.10. *For arbitrary $d \in$ **dom**, algebraic structures $< \sigma_d; \sqcup >$ resp. $(\sigma_d, \subseteq, \sqcup)$ constitutes idempotent and commutative semi-group resp. join semi-lattice containing unit element container $\vartheta(d)$. Algebraic structures $< \sigma_d; \sqcup >$ resp. $(\sigma_d, \subseteq, \sqcup)$ are isomorphic with algebraic structures $(d^K, \bigvee)$ resp. $(d^K, \preceq, \bigvee)$, by bijection d.*

Definition 2.11. Let $d \in$ **dom**, and $U \in \mathcal{U}$ be an arbitrary subset of $\mathcal{U}$. The set $\sigma_{d|U} = \{s \cap U : \ s \in \sigma_d\}$ is called the U-constraint of σ_d. For arbitrary $\mathcal{K} \in d^K$ the expression denoted by $\mathcal{K}_{d|U}$ represents sructure-semantics $\mathcal{K}(d) \cap U \in \sigma_{d|U}$, and is called the U-constraint of expression $\mathcal{K}$. U is called a constraint-range. U is a key-range for expression $\mathcal{K}$ resp. σ_d, if $|\mathcal{K}(d) \cap U| = 1$ resp. $\{s \in \sigma_d : \ |s \cap U| = 1\} \neq \emptyset$.

Lemma 2.12. *For arbitrary $d \in$ **dom**, to any non-empty set K of pairwise independent structure-semantics expressions, there exists a U_K key-range.*

2.2. Structure semantics schemata

Data local structure-semantics above can be generalized to particular subsets of **dom** by introducing structur-semantics schemata. Structure-semantics schemata are ordered coloured labeled graphs. Node labels are lengths of factors of data from **dom**, edge labels are the position where the factors within data starts, and edge coloures indicates the structure of factors in question. Using structure-semantics schemata, arbitrary "parts" (i.e. factors of products) of data can be ignored. A schema $\mathcal{G}$ is called exact resp. projection schema depending on whether only a prefixed, if any, or some inside factors are ignored. Figures 3 resp. 4 shows a semi-exact schema of 10 spoor-length and 8 projection-length resp. a projection schema of 10 spoor-length and 5 projection-length. Figures 5 resp. 6 shows the projections of datum $12349abcdef01a2b0fe0$ according to schema on Figures 3 resp. 4. The spoor-length of a schema is the sum of the labels of its initial edge and node, and the projection-length is the sum of the labels of the leaves. Given a schema $\mathcal{G}$ of spoor-length o, the set $\mathcal{O}_{\mathcal{S}}(\mathcal{G}) = \{d \in$ **dom** $: ||d|| = o\}$ is called the the

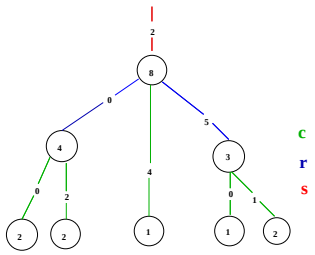

Figure 3: Semi-egzakt structure-semantics schema

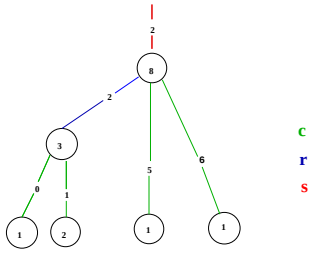

Figure 4: Projection structure-semantics schema

spoor of schema $\mathcal{G}$. A schema is called entirely exact if its spoor-length is equal to its projection-length. Results acheived for data local structure-semantics can be generalized for entirely exact schemata as follows.

Definition 2.13. For arbitrary $d \in \mathbf{dom}$ and entirely exact schema $\mathcal{G}$, let $d[\mathcal{G}]$ resp. $d[\mathcal{G}](d)$ denote the projection resp. the structure-semantics of datum d according to $\mathcal{G}$. Let further on $\mathbf{G}_p$ and $\mathbf{dom}_p$ denote the set of all entirely exact schemata of spoor-length p and their common spoor. For arbitrary $\mathcal{G} \in \mathbf{G}_p$, the set $S_{\mathcal{G}} = \cup\{d[\mathcal{G}](d) \in \sigma_d : d \in \mathbf{dom}_p\}$ is called the structure-semantics of $\mathbf{dom}_p$ according to $\mathcal{G}$, and $\sigma_{\mathbf{dom}_p} = \{S_{\mathcal{G}} : \mathcal{G} \in \mathbf{G}_p\}$ is called the structure-semantics domain of $\mathbf{dom}_p$. For some $\mathcal{G}, \mathcal{G}^{'} \in \mathbf{G}_p$, $\mathcal{G}^{'}$ is said finer than $\mathcal{G}$, and denoted by $\mathcal{G}^{'} \sqsubseteq \mathcal{G}$, if $d[\mathcal{G}^{'}] \preceq d[\mathcal{G}]$.

Theorem 2.14. *For arbitrary $p \in \omega$, relation $\sqsubseteq$ is a partial order over set $\mathbf{G}_p$. Algebraic structure $(\mathbf{G}_p, \sqsubseteq)$ and $(\mathbf{G}_p, \sqsubseteq, \bigvee)$ contain no least element, the greatest element for both is the schema that consists of the only initial edge and node coloured by* **c** *and labeled by* 0 *and by p, respectively. $< \mathbf{G}_p; \sqsubseteq >$ resp. $(\mathbf{G}_p, \sqsubseteq, \bigvee)$ are idempotent and commutative semi-group resp. join semi-lattice containing the above schema as unit element.*

Definition 2.15. For arbitrary $p \in \omega$, an operation denoted by $\sqcup$ may be defined on $\sigma_{\mathbf{dom}_p}$ as follows. For any $\mathcal{G}, \mathcal{G}^{'} \in \mathbf{G}_p$, $d[\mathcal{G}](d) \sqcup d[\mathcal{G}^{'}](d) = d[\mathcal{G} \bigvee \mathcal{G}^{'}](d)$.

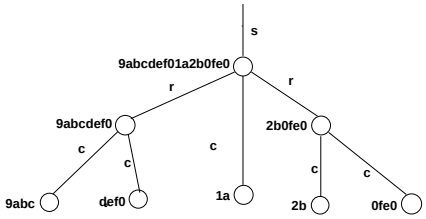

Figure 5: Projection of 12349abcdef01a2b0fe0 according to schema on figure

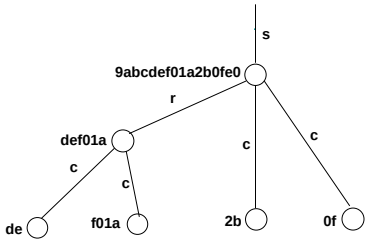

Figure 6: Projection of 12349abcdef01a2b0fe0 according to schema on figure

Theorem 2.16. *For arbitrary* $p \in \omega$, $< \sigma_{\mathbf{dom}_p}; \sqcup >$ *resp.* $(\sigma_{\mathbf{dom}_p}, \subseteq, \sqcup >)$ *are idempotent and commutative semi-group resp. join semi-lattice containing unit element* $\vartheta(\mathbf{dom}_p) = \cup\{vartheta(d) : d \in \mathbf{dom}_p\}$. *Algebraic structures* $< \sigma_{\mathbf{dom}_p}; \sqcup >$ *resp.* $(\sigma_{\mathbf{dom}_p}, \subseteq, \sqcup >)$ *are isomorphic with* $< \mathbf{G}_p; \sqsubseteq >$ *resp.* $(\mathbf{G}_p, \sqsubseteq, \bigvee)$.

Further generalizations of structure-semantics can be achieved by introducing constrainted structure-semantics schemata, and also the notion of *panel schemata*. A panel schema is formally an arbitrary set of structure-semantics schemata. A schema constraint is in turn an arbitrary subset of the universe. Given a panel schema $\mathbf{G}$, the spoor of $\mathbf{G}$ is defined as $\mathcal{O}_{\mathcal{S}}(\mathbf{G}) = \cup\{\mathcal{O}_{\mathcal{S}}(\mathcal{G}) : \mathcal{G} \in \mathbf{G}\}$. Given arbitrary $d \in \mathcal{O}_{\mathcal{S}}(\mathbf{G})$, considering set $\mathcal{G}_d = \{\mathcal{G} \in \mathbf{G} : d \in \mathcal{O}_{\mathcal{S}}(\mathcal{G})\}$, the structure-semantics of d by schema $\mathbf{G}$ is defined as $d[\mathbf{G}](d) = \cup\{d[\mathcal{G}](d) \in \sigma_d : \mathcal{G} \in \mathcal{G}_d\}$. The

structure-semantics of set $\mathcal{O}_{\mathcal{S}}(\mathbf{G})$ is in turn $S_{\mathbf{G}} = \cup\{d[\mathbf{G}](d) : d \in \mathcal{O}_{\mathcal{S}}(\mathbf{G})\}$. From now on we consider only (possible constrainted) entirely exact schemata or panel schemata built from those, referring to them simply as entirely exact schemata.

2.3. Operational- and structure-semantics

Data' operational semantics may be considered as operations given by data types. To make connection between structure-semantics and operational semantics visible, a data type that show structural properties is here defined as a three-tuple $(\mathcal{G}, (A, M), \mu)$, where $\mathcal{G}$ is an entirely exact structure-semantics schema, (A, M) is some abstract algebra with its bearer set A, and a set M of operations on A, and $\mu : S_{\mathcal{G}} \to A$ is a mapping such that the induced partition over $S_{\mathcal{G}}$ is exactly $\{d[\mathcal{G}](d) : d \in \mathcal{O}_{\mathcal{S}}(\mathcal{G})\}$, or some coarser one. Notice according to the above view of data types, the atomicity of a type depends on the operations of abstract algebra (A, M). A type can be considetered as atomic type, if no operation can yield any "particular part" of data of that type. The only "part" of such a type definition that may be visible for Container Model, is the spoor of schema $\mathcal{G}$, and the corresponding structure-semantics given by a partition over that, which may only be used to identify the operational semantics but remains invisible.

Definition 2.17. Let $\mathcal{G}$ be an entirely exact schema, and let π_T denote the partition $\{d[\mathcal{G}](d) \in \sigma_d : d \in \mathcal{O}_{\mathcal{S}}\}$ or some coarser. The pair $(\mathcal{G}, \pi_T)$ is called a datatype T over $\mathcal{O}_{\mathcal{S}}(\mathcal{G})$. Type T is said an object-like type if for each $P \in \pi_T$, P consists of only one element.

3. Conclusion

As for computer processed data, data semantics may be viewed from at least three different aspects, namely: structural properties, operational view, and presenting data as coded real world's objects. The latter two, in general are given by computer programs. Container Model (CM) is a formal algebraic model in order to study structural properties of data semantics. Data model considered in CM is a specialization of that in the logical and physical model in [1]-[3]. Beyond results presented above, one may realize that edge colours of structure-semantics schemata, and even JVM like virtual machines can be given as **dom** wide (data-) types according to Definition 2.17. Considering H-contacts (Definition 1.1) as **dom** wide (contact-)types, any particular schema may be represented by an edge colour within another schema. Extending CM by introducing operations called destructors that resolve H-contacts, and considering both as operations of a graph algebra, arbitrary transmutation on structre-semantics schemata may be achieved. Using Generalized Document Data Model [4], operational- and structure-semantics can be integrated by implementing structure-semantics schemata as XML documents. Using GDDM, data independence of application programs can be ensured in a higher level of abstraction, than that is done by DBMS today in use. That higher

level abstraction data independence concludes the notion of applications' autonomy introduced in [4] later on formalized and studied in [5, 6].

References

[1] Benczúr, A., Adatkezelő rendszerek biztonsági problémái, *Phd Dissertation*, (1978).

[2] Benczúr, A., Adatbázis rendszerek hatékonyságvizsgálati modellje Kolmogorov algoritmikus információmennyisége alapján, *Doctor of Sciences Dissertation*, (1988).

[3] Benczúr, A., Információs rendszerek matematikai modellezéséről, *Manuscript for Students*.

[4] Hernath, Zs., Vincellér, Z., Generalized Document Data Model for Integrating Autonomous Applications, *Proceedings of 6th International Conference on Applied Informatics, January 27-31, 2004, Eger Hungary*, Vol. II, (2004), 83-93.

[5] Benczúr, A., Hernáth, Zs., Porkoláb, Z., Autonomous Application – Towards a Better Data Integration Model, *Advences in Databases and Information Systems, Proceedings of the 9th East-European Conference, ADBIS 2005, Tallinn, September 12-15, 2005*, (2005), 150–163.

[6] Benczúr, A., Hernáth, Zs., Porkoláb, Z., LORD: Lay-Out Relationship and Domain Definition Language, *Advances in Databases and Information Systems, Local Proceedings of the 10th East-European Conference, ADBIS 2006, 3-7 September, 2006, Thessaloniki, Greece*, (2006), 215–230.

Zsolt Hernáth
Pázmány Péter sétány 1/c.
H-1117 Budapest
Hungary