Alex Shkotin

Finite Systems Handling Language (YAFOLL message 1)

Studia Humana nr 16, 3-12

2015

DE GRUYTER OPEN

studia humana
QUARTERLY JOURNAL

Studia Humana
Volume 4:4 (2015), pp. 3—12
DOI: 10.1515/sh-2015-0021

Finite Systems Handling Language (YAFOLL message 1)

Alex Shkotin

ACM
60-letiya Oktyabrya prospect 5/1-60
119334, Moscow, Russia

e-mail: ashkotin@acm.org

Abstract:
The concept a finite multi-carrier algebraic system (FMAS) as well as a language for handling systems such as YAFOLL (Yet Another First Order Logic Language) are introduced. The applicability of such systems to building a mathematical model of a part of reality, i.e. a mathematical structure that can be asked questions about the properties of subject domain objects and processes, is demonstrated.
Keywords: finite model, formal ontology, formal language, knowledge representation

1. Introduction

Algebraic system (AS) is a classical mathematical structure with a known procedure for specifying its properties and checking whether it possesses certain properties [1].

A YAFOLL language (abbreviated form: Y!L) fragment is described to create and use a certain type of AS. The processor that executes the YAFOLL language sentences is referred to as Yp. An AS is used in the form of a dialog with *Yp*: one or more sentences are fed to its input, and it returns an empty string or one or more messages. *No value* is one of important messages. Yp consists of two components: a syntax analyzer and an executor. The syntax analyzer passes the entire input text to output, and inserts messages in it in the form of xml elements in the event that errors are found, but its main function is to build a syntax tree. The executor receives the syntax tree of a group of sentences at its input, executes them, and presents a report on its operation as an xml document.

The Y!L sentences include the following commands: The !0! command zeroes the AS served by Yp, the !2! command outputs a text in Y!L where the AS contained in Yp is specified. The AS handling procedure on the whole is as follows: AS is uploaded from a file in Yp, processed, and downloaded from Yp by the !2! command. Thus, AS is contained in the text (*ontology*).

Basic definition

The classical AS definitions ([4], p.46, and [2], Vol I, p. 25) use the following algebraic system components: carriers, relations, operations (functions). Let us start with multiple sorts (see [3], p.71), limitations on the power of sets and partial functions. We will drop the relations immediately. In addition, let's call the carriers sorts for simplicity.

FMAS-0 **definition**

A finite multi-carrier algebraic system of the form 0 (FMAS-0) is a finite set of:

1. finite nonintersecting named sets (**sorts**) and probably some of their named elements (**constants**),

2. (also named) **functions** from a sort or a direct product of sorts into a sort. These functions are called primary functions.

end of definition

Note: An AS without relations is usually called *algebra*.

So, the function is conceived as a partial one (from-function), but prove to be full, and is even sometimes required to be so. There are many examples of unnamed functions in the λ-calculus.

Names

Let's use Flex syntax for regular expression (RE). Let an alphabet contain a set of letters (Lttr), a set of numbers (DIGIT), and letter '_' which we will call US. We will name FMAS and its components using the following RE: **Id** = {Lttr}({Lttr}|{DIGIT}|{US})*

i.e. a letter which can be followed by a chain of letters, digits and "_". For example, the following Y!L sentences:

Declaration sample sort. Declaration place sort.
introduce two sorts: sample and place.

Numbers and strings

The number are specified by the following RE:
Number = {MI}?{DIGIT}+({DOT}{DIGIT}+)?

where MI denotes '-', and DOT denotes '.', i.e. decimal rational numbers, but the '+' sign is implied and is not allowed.

Examples: 0, 10, -3.62.

The strings are specified by the following RE: **String** = {DQ}[^{DQ}]*{DQ} where DQ denotes a double quote mark ("), i.e. this is an arbitrary chain of letters except DQ framed by DQ. For example, "this is a string".

Sort elements

We will denote sort elements using letter strings of the following RE type **Ide** = {US}{Lttr}({Lttr}|{DIGIT}|{US})* i.e. strings beginning with "_" followed by Id. It is clear that the two sets of strings (Id and Ide) do not intersect. Let us introduce the following sort: Declaration TV sort. and specify its elements:

```
! _True TV !
! _False TV !
```

Let's e_m denotes "!" here and below in BNF rules. The Y!L language BNF rule
(fmca) Statement : e_m Ide Id e_m
assigns the Ide element to the Id sort.

The rule
(fmcd) Statement : e_m Ide e_m
removes the Ide element from the sort it belongs.

These two types of sentence (fmcd and fmca) enable introducing the composition and populating the sorts, i.e. adding elements to and deleting elements from a sort. Recall that sorts never intersect according to paragraph 1 of the FMAS-0 definition. The following Y!L sentences fill up the place and sample sorts:

```
! _PLC1809 place !
! _SAM32994 sample !
! _SAM32995 sample !
! _SAM32996 sample !
```

Result 1. Current example

Let's the !0! command (*forget all*) is sent to Yp, and then the Y!L commands previously encountered in the text are executed, then AS will contain three sorts populated as follows:

```
sample={_SAM32994 _SAM32995 _SAM32996}
place={_PLC1809}
TV={_True _False}
```

If now !2! is said to Yp, then we will obtain the AS text in the form of sentences for creation from scratch at the output:

```
!0!
! _True TV !
! _False TV !
! _PLC1809 place !
! _SAM32994 sample !
! _SAM32995 sample !
! _SAM32996 sample !
```

R and S sorts

The language has two built-in sorts R and S. R sort elements are numbers of the Number RE-type. S sort elements are letter strings of the String RE-type. Note: These sorts are denumerable, and do not take us beyond the FMAS is carefully used.

2. Finite System Elements

Function signature

The existence of multiple sorts results in the necessity of specifying not only the number of function arguments, but also the sort of each argument and function value sort. This leads to the notion of function signature (see [3], p.53 where the function signature is called *type*). Let COLON denote the ':' letter.

Signature is a syntactic structure (sig_f) specified by three BNF rules:

```
(sig_eI)          sig_e :   Id
(#sig_arg) sig_arg : sig_e+
(sig_f)   sig_f : sig_arg COLON sig_e
```

The sig_eI rule says that the sig_e non-terminal must be Id. A sort Id is *semantically* implied in this case. The #sig_arg rule says that the sig_arg non-terminal is a chain consisting of one or more sig_e. The sig_f rule says that signature is an argument signature followed by a colon (COLON letter) and a result signature. For examle, the sentence Declaration gathering_place sample : place prime. introduces the primary function gathering_place with a *sample : place* signature, i.e. the function is declared to be unary with an argument from the *sample* sort and a value from the *place* sort.

Truth-values and predicates

The existence of truth-values enables expressing relations of a classic AS through functions of a special type, predicates, with the TV range of values. See also *Relations and Predicates* [5] and [3], p.71. For example, let's declare

Declaration rhyolite sample : TV prime .

predicate that can assume the _True value on a *sample* sort element, which is natural to interpret as that this sample is rhyolite; _False value on a *sample* sort element, which is natural to interpret as that this sample is not rhyolite. In addition, since rhyolite is an originally partial function, there may be no value at all. Whether is it permissible for a predicate to be partial depends on the subject domain. So, if we have a rock sample and information about it in an AS, we may be unaware whether it is rhyolite or not at some time, and the lack of value can be interpreted as *unknown*.

For ex. studies may result that a particular sample represented as _SAM30697 in the AS proving to be really rhyolite. The predicate value should be specified as _True on _SAM30697, which is written as follows in Y!L:

```
!rhyolite ( _SAM30697 ):_True!
```

i.e. _True is assigned to the rhyolite function value on _SAM30697. Now, if we ask Yp ?rhyolite (_SAM30697)? it will answer _True.

Notion of term

The syntactic structure *term* is primarily used to obtain values from function application to arguments. For example, the NOT (_True) term typically has a _False value. We will also use the term to set values for primary functions. For example, the sentences

```
!NOT ( _False ):_True!
!NOT ( _True ):_False!
```

set values of the unary function NOT on TV sort elements.

Let's l_p denotes "(", r_p - ")", EXISTS - ∃, FOR_ANY - ∀, here and below while COLON denotes ':' as before. In a general case, we have syntax:

```
(trmi)   term : Id
(trmie)  term : Ide
(trmn)   term : Number
(trms)   term : String
(trmf)   term :   Id l_p TermList r_p
(trmp)   term :   l_p INFIX term r_p
(trmin)  term : l_p term INFIX term r_p
(trme)   term :   l_p EXISTS   Id COLON Id term r_p
(trma)   term :   l_p FOR_ANY Id COLON Id term r_p
(#trml)  TermList : term+
```

So, trmi says that the term can be Id, which is implied to be a constant and the term value to be the constant value. trmie says that the term can be Ide, and its value is the term itself. trmn says that the term can be Number, i.e. a decimal number, and its value is the term itself. trms says that the term can be String, i.e. a string enclosed in quotation marks, and its value is a string without quotation marks. trmf says that the term can be Id (functions being implied) applied to the list of terms (TermList), and term values are implied to give arguments for the Id function. For example, the NOT (_False) is a trmf rule term. trmp, trmin enable using special letters or key words (-trmp prefixes and -trmin infixes) to be assigned unary or binary functions. The prefixes and infixes are jointly called *fixes*. trme, trma say that quantifier structures are terms as well. Id-1 (the first Id) is implied to be an identifier of a variable bound by the quantifier, and Id-2 the sort of this variable. The term value is _True or _False. #trml says that TermList is a non-empty list of terms.

INFIX can assume the following values: ¬ ∧ ∨ → ≡ ≠ ≤ ≥ = neq leq geq < > + - * / not and or impl eqv

Let's consider an example. The term

(∀ x : sample (∃ y : place (gathering_place (x) = y)))

states that a place of collection exists/is specified for any sample. The value of this term can be true, false or result in a *No value* message on a particular AS. For example if we ask Yp

?(∀ x : sample (∃ y : place (gathering_place (x) = y)))?

No value. The term value is calculated in some language sentences that are easier to consider as commands to the Yp processor. If the term has no value, then command execution is stopped, and the processor outputs a *No value* message.

Definition of a *free identifier* in a term. This is an identifier that is not assigned to any constant, function or quantifier.

Definition of a *closed* term. A closed term satisfies the mandatory requirements and contains no free identifiers. The additional requirement is equivalent to trmi-4.

(trmi-4) If Id is not assigned to a constant or function in the term, then it is assigned to a quantifier.

Specifying primary functions

Many functions can be specified by maps: set of of n's such that the first n-1 values are function argument values, and the last value is function result value. So, the NOT function map consists of two n's: <_True _False> <_False _True>. On the other hand, values can be set for terms. For example, let _SAM30697 be an element of the *sample* sort, then a value can be set for the primary function *rhyolite* as follows:

! rhyolite (_SAM30697):_True!
In a general case, we have syntax:
(fmta) Statement : e_m term COLON term e_m
(fmtd) Statement : e_m term COLON e_m

The fmta rule sentence assigns the term-2 term value to the term-1 term, including a constant. If term-1 term already has a value, then it is replaced. The fmtd rule sentence deprives the term-2 term of the value, including a constant. There are big restrictions on the term structure. Let's start with definitions. term-1 called FSt (Finite System term), and term-2, FSv (Finite System value). The **FSt** must satisfy the following additional requirements: be a trmf term without quantifiers or fixes; its identifiers are only constants and functions, both being primary. It's follows from the definition that sort elements are permissible. **FSv** must be a string, number, sort element or Id of a primary function or constant. Note: The possibility of using the Id of a primary function or constants takes us beyond FMAS because the value is not an element of the sort already.

Requirements

(fmta-1) term-1 must be FSt.
(fmta-2) term-2 must be FSv.
(fmta_proc-1) Consistency of types. The term-1 value type must be equal to the term-2 a value type.
(fmtd-1) term must be FSt.
(fmtd-2) term must be in the FS, i.e. have a value. The desire to remove a non-existing value can be a mistake caused by a lack of understanding of the FS structure.

Specifying an FMAS

Height 1 terms suffice to specify an FMAS, i.e. terms of the type of function application to simple arguments, not other functions.

For example, the following sentences containing height 1 FSt

!OR (_False _False):_False! !OR (_False _True):_True!
!OR (_True _False):_True! !OR (_True _True):_True!
set OR primary function values.
The sentences
!NOT (_False):_True! !NOT (_True):_False!
specify the NOT primary function.
The sentence
Declaration authorial_number sample : S prime.
specifies a function assigning the author's number to the sample.
For example, !authorial_number (_SAM32994):"A"!
The sentences
Declaration latitude place : R prime. Declaration longitude place : R prime.
enable specifying the latitude and longitude for a place:
! latitude (_PLC1555):-7.93!
! longitude (PLC1555):-14.37!

Finite system (FS)

A totality of FSt, FSv pairs where all the FSt are different is called *Finite System* (FS). There are no restrictions on the FSt height, and the following expression can be written:

!NOT(NOT(True)):False!

In doing so, one only has to keep it in mind that Yp goes along the term from the bottom upwards when searching for the term value in a FS: it will try to find a value for NOT(True), and will only search for a value for NOT(NOT(True)) if there is no value for NOT(True). It's unclear whether such opportunity is required in the practice of ontologistics.

Identity relation

It is natural to assign a binary predicate of an identical relation to each sort, which is true then and only then when argument values are one and the same sort element. A function should be introduced, and an '=' infix assigned to it to do that. For example, let *sample* and *place* be two sorts. The following YAFOLL sentences introduce two primary functions (EQU_sample, EQU_place) and declare them computable using the fm_strcmp algorithm, which is part of the Yp processor. Each of these functions is assigned one and the same '=' infix.

Declaration EQU_sample sample sample : TV prime fm_strcmp. Add infix "=" to EQU_sample.
Declaration EQU_place place place : TV prime fm_strcmp.
Add infix "=" to EQU_place.

Term value request

Once values of primary function and constants, if required, i.e. FS, have been specified, we can calculate the values of terms on the FS. It can be sometimes regarded as a clarification of FS properties and sometimes as obtaining the value of interest. Note that first order predicate calculus formulas, including quantifier formulas, are terms. Let q_m denote "?". Having BNF-rule

(st-11) Statement : q_m term q_m

This sentence asks the term value from Yp. Since we are dealing with partial functions, not only a value, but also a message can be the Yp processor response: No value!

Requirements

(st-11-1) The st-11 rule term must be closed, i.e. meet trmi-4 in addition to general requirements.

A query to FS performs two fundamental functions by analogy to AS: It checks whether subject domain axioms are satisfied, It clarifies the subject domain properties. If all the axioms are satisfied, then the AS is considered a model of the system of axioms and subject domain itself.

For ex. ?(∀x:TV ((x∨x)=x))? responses _True.

3. Building an Ontology of the Subject Domain (PROBA DB)

Let's review the formalization of a small part of the petrology knowledge: accumulation of information about rock samples, i.e. place of collection, rock, concentration of chemical substances, article where the information is published, etc. We are talking about all the samples accumulated at all laboratories in the world. The information already accumulated in the Proba database is taken as a sample [6]. This is a 'world' of samples stored at laboratories, places on Earth where they were collected, and publications on sample properties.

Sorts

We will keep to the 'flat' idea about sorts during formalization: if entities of a species are of interest, then this such species can be considered to be a sort. We'll obtain three subject sorts at once: *sample* is a rock sample, *place* is sample collection place, *publication* is publication.

Populating sorts and basic attributes

The rock samples processed and stored at a laboratory can be identified using various methods. The Ide type identifier assigned to the sample can be arbitrary within a FMAS, provided that three requirements are satisfied:

(GUI) global unique identification: the identifiers of different samples must be different.

(I2SW) id to sample works: the sample can be found by the identifier and FMAS in the laboratory.

(S2IW) sample to id works: sample identifier in FMAS can be obtained by the sample at the laboratory.

A unique number of record on sample was taken from the Proba database and updated to the Ide to get GUI. For ex. DB record number 32994 obtains an Ide = _SAM32994.

References to reality functions, i.e. basic attributes of the element of each subject sort enabling finding the sample in reality by their values are introduced to execute I2SW in FMAS. For example, let's introduce the function Declaration authorial_number sample : S prime.

The authorial_number function supports the author's sample number, which is unique for the laboratory that stores the sample. This function must be full, i.e. we have an axiom: Axiom AN_full (∀x:sample(∃y:S(authorial_number(x)=y))).

A material algorithm is assumed to exist: how a sample can be found in the laboratory if its author's number and other basic attributes are known. A material algorithm that is 'reverse' in a certain sense is assumed to execute S2IW, i.e. to determine sample Ide in the FMAS while being one-on-ones with the sample in the laboratory: values of the basic attributes of the sample unambiguously characterizing it in FMAS are located somewhere in the laboratory. This unambiguity is supported by the axiom of uniqueness of the totality of basic attributes. Suppose for simplicity that already authorial_number is globally (for all laboratories) unique. Then the axiom of uniqueness will look as follows: Axiom AN_uni (∀x:sample (∀y:sample (authorial_number(x)=authorial_number(y))→(x=y))).

A set of basic attributes, material algorithms, and axioms of completeness and uniqueness of their basic attributes similarly exists for *publication* sort and *place* sort. In addition, the following formalization method is used for geographical names (for example, Iceland, Atlantic Ocean): they are used to form an Ide (_Iceland, _Atlantic_Ocean) and to assign it to the *place* sort:

!_Iceland place! !_Atlantic_Ocean place!

Additional attributes and relations

The attributes that are not basic attributes can be diverse and are found in big quantities in databases. The presence of numbers and strings in YAFOLL enables adding any DB attribute to FMAS. Also note that the functional relation between DB tables is simply a partial (or full) function in FMAS. It does not mean that each table should be assigned a sort when building an FMAS on the basis of a DB, but some tables will usually be sorts. The gathering_place function assigning the place of collection to the sample was already mentioned above. It has a completeness axiom of its own: Axiom GPfull (∀ x : sample (∃ y : place (gathering_place (x) = y))). And value assignment sentences

! gathering_place (_SAM30681):_PLC1555!
! gathering_place (_SAM30682):_PLC1555!
! gathering_place (_SAM30683):_PLC1555!

Example of a string attribute: Declaration title publication : S prime. And value assignment to it: !title (PUB5633):"A CONTRIBUTION TO THE GEOLOGY OF THE KERLINGARFJELL"!

Predicates of rocks and chemical substances

It is natural to present the term corresponding to a certain rock (for example, rhyolite) in FMAS by the predicate: Declaration rhyolite sample : TV prime. Assuming one of the truth-values on the sample: _True _False or it may have no value although this is prohibited by the axiom: Axiom full_rhyolite (∀ x : sample (∃ y : TV (rhyolite (x) = y))). This predicate is primary in this FMAS, and values may and

should be assigned to it by assigning samples to it, for example as follows, !rhyolite (_SAM30697):_True! !rhyolite (_SAM32994):_True!

Example of chemical substance predicate: Declaration SIO2 sample : TV prime. It will be true on a sample only if the sample is a silica.

4. Conclusion

A part of the YAFOLL language and its Yp processor potential enabling handling FMAS and ask 'first-order' questions about its properties is described.

Finite system. The finite nature of a system simulating a part of reality is a most important element of the approach. A part of reality is simulated exactly as a finite system of elements in most cases. The elements may prove to be very numerous (for direct calculations), of course, or a part of the elements may prove to be conceived as continuous, i.e. 'infinite'.

Comparison with RBD. It should be emphasized that FMAS regarded as a data model (DM) is no weaker than a relational DM. The RDB tables structure can be reproduced one-for-one in the FMAS, too. However, this does not mean that one should act so when building an ontology.

Several languages. In fact, at least 3 languages are combined in YAFOLL: Handling FMAS, Query to FMAS, Language of responses. The latter is quite primitive, but it will develop.

Direct calculations. The idea of direct calculations is fundamental and precedes the idea of logical deduction. The vast majority of engineering calculations are generally algorithmically direct calculations. One of the main objectives of the project as a whole is to find out what types of direct calculations are needed to simulate a subject domain. In doing so, it was natural to start with a first-order predicate calculus language (FOL). The area of direct calculations includes both formalized laws of the subject domain and properties of the subject domain entities. The former ones are accumulated as axioms, and the latter ones as query formulas.

If an entity is conceived as a system in science, technology or legislation, then it can be simulated by a finite system. The properties that such system and the languages used to handle it should possess is exactly the subject of this message and subsequent ones.

References

1. Algebraic structure: 2015. *https://en.wikipedia.org/wiki/Algebraic_structure*. Accessed: 2015-09- 18.
2. Barwise, J. 1982. Introduction to first order logic. In Reference Book on Mathematical Logic I, J. Barwise, Ed., Moscow, Nauka, 13-54.
3. Kolmogorov, A.N., Dragalin, A.G. Mathematical Logic. 3rd stereotypical edition. Moscow: KomKniga, 2006.
4. Maltsev, A.I., Algebraic Systems, Moscow, Nauka, 1970.
5. Finitary relation: 2015. *https://en.wikipedia.org/wiki/Finitary_relation*. Accessed: 2015- 09- 18.
6. Shkotin, A. and Ryakhovski, V. (2015). *Proba DB. Ontology of a relational database - Open science*. [online] Sites.google.com. Available at: https://sites.google.com/site/alex0shkotin/formalnaa-geologia/bd-proba-ontologia [Accessed 18 Sep. 2015].