Hereditarily-finite sets, data bases and polynomial-time computability

Vladimir Yu. Sazonov

Computer Logic Laboratory, Program Systems Institute of Russian Academy of Sciences, 152140 Pereslavl-Zalessky, Russia

Abstract


The first informal result and the aim of this paper (which is the extended version of Sazonov (1988)) is a step towards a somewhat more practically oriented version of set-theoretic “$\Delta$-programming” language [Sazonov (1985, 1987)] which may be considered as a “resource bounded” language for specifying data bases and corresponding queries. In fact, instead of the ordinary universe $\text{HF}(\#)$ of hereditarily-finite sets over any given class $\#$ of urelements, we consider [following the ideas of Red’ko and Basarab (1987)] a more general universe $\text{HFA}(\#)$ of hereditarily-finite sets of “named” elements, i.e. sets of data qualified by some attributes. The second result is more mathematical. It generalizes a theorem of Sazonov (1985, 1987) to the case of $\text{HFA}(\#)$, makes it more precise and implies that $\Delta$-language is sufficiently complete.

Theorem (cf. [42, 43]). If $\Delta$-language involves a linear order on $\#$ then its terms define exactly all operations $\text{HFA}(\#) \rightarrow \text{HFA}(\#)$ which (1) preserve supports of their arguments, (2) transform isomorphic arguments to isomorphic ones, respecting their supports, and (3) are computable in polynomial time under graphical representations of sets in $\text{HFA}(\#)$.

1. Introduction and discussion on resource bounded/unbounded styles of programming

From the complexity theory point of view, polynomial-time computability is usually considered as “tractable” or “feasible” and therefore could be treated as the basis for a real programming style. Nevertheless, one should not completely identify it with practical feasible computability (as one should not identify any other mathematically defined abstract notion with some independent reality). It seems better to say that polynomial computability adequately reflects a very important aspect of real computations, i.e. their relativeness to resource bounds. This conclusion is based on
$\Delta$-characterization of polynomial computability (given in this paper), as well as on its other characterizations in terms of general recursion in a finite domain, or preferably in a finite row of natural numbers (found independently by the author and Immerman, Vardi, Livchak and Gurevich; cf. [20, 24, 28, 38, 39, 51]).

Polynomial-time computability [38, 39] was described in terms of general recursive functions (possibly relative to some other functions) in an abstract finite row of natural numbers $0, 1, 2, \ldots, q - 1$, where the value of the largest natural number, $q$ (the abstract resource bound), is not specified. Essentially the same description was given in [20], where additionally logarithmic space computability was characterized as primitive recursivity in the finite row of natural numbers. Instead of $\Box$-recursive functions, various recursive definitions of predicates and computability notions over linearly ordered finite relational structures were considered [24, 28, 51] in logical, rather than arithmetical terms to obtain descriptions of (N)LOGSPACE, PTIME and PSPACE complexity classes. This work was prompted by the problem of finding good extensions of relational (essentially first order) query languages (cf. [1, 6, 7, 9]). Since then many papers have been written dealing with recursion in finite domains (for example, [21–23, 25, 27, 30]), including finite type versions [18].

Note that previously primitive recursion in a finite row of natural numbers was considered by Mostowski [31]. Also PTIME was defined by Cobham [8] in terms of some kind of limited primitive recursion in the ordinary infinite row of natural numbers. It seems that all the authors who considered recursion in finite domains were strongly influenced by characterization of NP (nondeterministic polynomial-time computability) in terms of $\Sigma_1$-definability in finite relational structures, which was found by Jones and Selman [26] and Fagin [14]. The well-known representation of the notion of computability in terms of finite models given by Trakhtenbrot [50] is also essential.

In [42, 43] and in the present paper relativization to resource bounds consists in using (instead of $\Box$) only explicitly bounded quantifiers over hereditarily-finite sets $T$ ($\forall x \in T, \exists x \in T$) and the corresponding construct of bounded recursive or inductive set definition $[q = \{x \in T | \varphi(x, q)\}]$ of finite subset $q \subseteq T$. Other intuitively bounded operations over hereditarily-finite sets are also allowed and together comprise the so-called $\Delta$-language where, traditionally, $\Delta$ (or $\Delta_0$) means “bounded”.

Computability with bounded resources of both $\Box$ and $\Delta$-approaches can also be characterized as the following reasonable discipline for declarative programming: to use in any program description only those (input or intermediate) data that are given in advance or can be constructed by (finite) resources given in advance. Actually, one can easily recognize that this discipline is implicitly used in such an important domain of computing practice as relational data bases where the answer to any query on a data base state is constructed from this finite state only. It also seems useful to make this discipline more explicit.

Relativization of computations and reasoning to resource bounds also means that the traditional abstraction of potential feasibility, i.e. the abstraction from the very existence of any resource bounds, is not admitted at all. In contrast to such a style of
programming, the ordinary programming languages involve (implicit or explicit)
unbounded positive existential quantifiers. The most direct example is the “Σ-pro-
gramming” of Goncharov et al. [19] (cf. also [44]) based on the ideas of
Kripke–Platek set theory [3]. Unlike \( \Delta \)-style (also based on KP), this \( \Sigma \)-style of
programming presupposes that a program uses not only some input data but also all
the potential infinity of resources which could be involved eventually by unbounded
existential quantification. Another example of an “unbounded” programming con-
struct is WHILE-DO. Analogous searching through the unbounded Herbrand’s
universe is involved in implementation of PROLOG. Such unbounded languages give
rise, in general, to nonhalting programs (both in practice and in theory).

We will be rather liberal here and will admit as “sufficiently bounded” the successor
operation \( x + 1 \) for the ordinary, unbounded row of natural numbers or set-theoretic
operations \( \{x, y\}, x \cup y \) in the unbounded universe of hereditarily-finite sets. But free
iteration of such operations via, say, primitive recursion, is not allowed in general,
being the key point of the abstraction of potential feasibility. For example, \( x + y \) is the
iteration of \( x + 1 \), \( x^y \) the iteration of \( x + x \), \( x^y \) the iteration of \( x \cdot x \), superexponen-
tiation the iteration of \( x^x \), etc., the last two operations being surely nonfeasible. In other
words, the abstraction involved in the successor or, say, addition or union operations
seems rather harmless in comparison with the very strong idealization connected with
iteration of any operations and giving rise to nonfeasible computations.

It seems that our strongly bounded reality is described somewhat inconsistently by
an “unbounded” traditional approach to programming (and also to mathematics)
based on the abstraction of potential feasibility or even on the abstraction of actual
infinity. That is why we prefer here \( \Delta \)-programming (and even \( \Delta \)-mathematics) which
do not use these very strong abstractions, and investigate in [41–44] conditions under
which \( \Sigma \)-programming is “conservative” over \( \Delta \).

However, we should not forget that the real source of unbounded programming
actually exists and is connected, for example, with the tasks of (unbounded in various
senses) searching for a proof of a theorem or a winning strategy in a game, or
a solution for an NP-complete problem, etc. Therefore, our aim should consist not in
completely rejecting the unbounded style of thinking, but simply in making the
bounded one legal and more elaborated. (Cf. also investigations on bounded arith-
metic, for example [4, 10, 32 34, 39, 40, 46, 48], on bounded set theory closely related
with \( \Delta \)-language [41–43] and on linear logic as logic of resources, especially [17].)
From the point of view of computer science, applied mathematics and corresponding
mathematical foundations, a more proper and possibly somewhat restricted role of
the unbounded style of programming and mathematical thinking is to be found.

In contrast with the ordinary “flat” relational data bases corresponding to the
\( \xi \)-approach, we are concerned here with nested data bases (cf. e.g. [35, Chapters 1 and
7] and [5, 11–13, 36, 37]). This is done via hereditarily-finite sets and means that
complex data may be constructed from some more simple data which, in turn, are
constructed from some other data, etc., up to some atoms (urelements). Such a nested-
ness may be connected with the nature of the reality which the data base should
Almost all interesting and sufficiently complex objects usually have a "nested" form. On the other hand, nested representation of data in a computer allows more efficient processing.

So, we again return to the above discussion on "feasibility". Note that both flat and nested cases, as they considered here, correspond to the same complexity notion of polynomial-time computability (if general recursion in finite domains or sets is involved). However, this correspondence does not completely characterize the efficiency of these approaches. The main point is concerned with the form of quantification and recursion over the data.

For example, in the flat case quantifiers are formally unbounded. The state of a data base consists of several finite domains. That is why quantification is actually finite and therefore implicitly bounded. Nevertheless, if the quantified domain is sufficiently large then quantification, especially if repeated, may be rather difficult to implement in practice. For example, if you need to know something about some small group of persons (which does not constitute the whole domain given in advance but which is well described), why should you quantify over all the numerous persons in the domain?

Explicitly bounded quantifiers of $\lambda$-language $\forall x \in T, \exists x \in T$, together with the big freedom of constructing any (sets $T$ of) nested data which we need, allow us to overcome this problem to some extent. The user has all the necessary machinery to (re)organize the data in such a way that all expected quantifications and recursions in the possible queries on these data will be most likely relativized (i.e. bounded) to sufficiently small sets $T$ of data.

Of course, this discussion cannot serve as a rigorous substantiation of the efficiency of the $\lambda$-language. After all, we simply argue that $\lambda$-language is sufficiently flexible and that it corresponds to the requirement of relativeness of computations to resource bounds in a way which seems somewhat more adequate than, although not so straightforward as, in the flat case (in which even the intended current "world" is required to be finite). In $\lambda$-language, relativeness to resource bounds is represented by explicitly bounded quantifiers and recursion, i.e. by syntactical means, unlike the flat case where this is done only via "finite" semantics.

In contrast to the "tuple-relation" approach of [35, 36] and analogous to [11–13, 16, 37], we develop the nested case in set-theoretic language. The kernel part of our $\lambda$-language (i.e. that excluding TC, C, and inductive $\lambda$-separation; cf. Section 3) is equivalent to the language of Basic Set Theory of Gandy [15] (in the absence of attributes and urelements) and gives [41–43] exactly provably total $\Sigma$-operations of Kripke–Platek Set Theory [3] without foundation axiom. The main distinctions from the Dahlhaus and Makowsky SETL-like language [11–13] and the Gilula and Stolboushkin STARSET language [16] are as follows: $\lambda$-language is declarative, it does not contain unbounded tools like WHILE-DO and allows (in the present version; cf. [45]) use of attributes. Also, STARSET uses only (finite) classes of sets of urelements, i.e. very low degree of nestedness, essentially as in the flat case.

Therefore, we can briefly characterize our $\lambda$-language as a flexible and complete declarative set-theoretic programming language allowing free construction of (sets of)
nested data and using only bounded quantification and bounded recursion. Here completeness means, intuitively, that no essential bounded declarative construct is forgotten. Formally, $\Lambda$-language describes exactly polynomial time computability over the universe of hereditarily-finite sets (under a graphical representation of sets). This kind of characterizing completeness (or expressibility) of $\Lambda$-language is the main difference of our approach to nested data bases from that of [35–37].

The rest of this paper is organized as follows. Section 2 introduces the universe $\text{HFA}(\mathcal{U})$ of hereditarily-finite sets with urelements and attributes and interprets it as the universe of possible states of nested data bases together with some typing discipline for data. Corresponding declarative set theoretic $\Lambda$-programming language is described in Sections 3 and 4. Then preserving the isomorphism of data by $\Lambda$-programs is described in Section 5. This allows an appropriate definition in Section 6 the notion of polynomial computability over $\text{HFA}(\mathcal{U})$ with respect to graphical representations of HF-sets. The Main Theorem that this notion coincides with $\Lambda$-expressibility is also proved. Section 7 contains concluding remarks on various codings of HF-sets and some perspectives.

2. Universe $\text{HFA}(\mathcal{U})$ and quasirelational nested data bases

In this section we give our somewhat different version of the definitions in [37]. Note that symbol := is used below for equality or equivalence by definition.

The following two clauses define inductively the universe $\text{HFA}(\mathcal{U})$ of hereditarily-finite sets with attributes, where $\mathcal{U}$ is an arbitrary collection of urelements (= elementary data; in applications $\mathcal{U}$ can be taken as a set of not very long words in some alphabet):

1. $\emptyset \in \text{HFA}(\mathcal{U})$, empty set $\emptyset \in \text{HFA}(\mathcal{U})$;
2. if $x_1, \ldots, x_k \in \text{HFA}(\mathcal{U})$, $a_1, \ldots, a_k \in \mathcal{U}$, then $\{a_1 : x_1, \ldots, a_k : x_k\}$ is the set of ordered pairs $a_i : x_i$, which are results of qualifying elements $x_i$ by attributes $a_i$, then $x \in \text{HFA}(\mathcal{U})$.

In the second case we write $a_i : x_i \in x$ and say that $x_i$ is an element of the set $x$ named by $a_i$ or that $x_i$ is the $a_i$th projection of quasituple or quasirecord $x$. If all attributes $a_i$ in $x$ are different then $x$ is called a tuple or record. If all attributes $a_i$ coincide then $x$ is called a uniform set. If all $x_i$ are thought of as (quasi)tuples then $x$ is considered as (quasi)relation. A one-element set $\{a : x\}$ is called a singleton set. It represents the named element $a : x$. Note that any named element $a : x$ by itself is not formally a member of the universe.

As usual, for any sets $x, y \in \text{HFA}(\mathcal{U}) \setminus \mathcal{U}$ which are not urelements we have

$$x \subseteq y \& y \subseteq x \Rightarrow x = y,$$

where

$$x \subseteq y := \forall a : z \in x. (a : z \in y).$$

(Here, both $a$ and $z$, but not $x$ are quantified. Thus, the order and repetitions of
elements in a set are irrelevant, as they should be for sets and as the reader could expect.)

Note that urelements are treated both as attributes and as elementary data, sets from $\mathbf{HFA}(\mathcal{U})$ being considered as complex, nested data. Of course, another way would be to take attributes to be nonoverlapping with, or to be only some part of, urelements. In our definition empty set $\emptyset$ also plays the role of an additional, "empty" attribute. This is done because we need at least one attribute and reserve this possibility when there exist no urelements.

To abbreviate, we can group together elements in $x$ with the same attribute and will not mention the distinguished empty attribute $\emptyset$. For example, $\{u, v; a: y, z; b: w\} := \{\emptyset: u, \emptyset: v, a: y, a: z, b: w\}$. Then the ordinary universe $\mathbf{HF}(\mathcal{U})$ of hereditarily-finite sets over $\mathcal{U}$ can be considered as the part of universe $\mathbf{HFA}(\mathcal{U})$ consisting of those sets which involve (at any depth) only empty attributes. We may consider, for example, the following (named) datum

**Student-Ivanov:**

\[
\{\text{STUDNAME: Ivanov-I.; BIRTH\_YEAR: 1968; GROUP: 323; }
\text{FACULTY: Phis; FRIENDS: Petrov-A., Sidorov-C.}\}
\]

Attributes are aimed at marking the user's intentions concerning ways of using corresponding data. We may additionally introduce a typing discipline for using data, for example, as follows.

Suppose $u::v$ (four dots) is some given binary relation in $\mathcal{U}$ which means intuitively that "$u$ is of type $v$". For example, $323::\text{INT}; 1968::\text{INT}; 1968\_oct\_15::\text{DATE}; \text{Ivanov-I.}::\text{NAME}; \text{Ivanov-I.}::\text{WORD}; \text{Petrov-A., Sidorov-C}::\text{MAN\_NAME}; \text{Phis}::\text{WORD}, \text{etc.}

Let us extend inductively this binary typing relation to the whole universe $\mathbf{HFA}(\mathcal{U})$: $x::\alpha := x::\gamma x \lor (x, \sigma \notin \mathcal{U} \land \forall a: y \in x. \exists h: \beta \in x. (a = h \land y::\beta))$.

For example, the above data on the student Ivanov have the type

\[
\{\text{STUDNAME: NAME; BIRTH\_YEAR: INT; GROUP: INT; }
\text{FACULTY: WORD; FRIENDS: MAN\_NAME, WOMAN\_NAME; }
\text{HOMEADDRESS: ADDRESS}\}
\]

and are probably incomplete with respect to this type (home address of Ivanov is missing and probably not all friends are listed).

If every urelement has a type (in $\mathcal{U}$) then every $x \in \mathbf{HFA}(\mathcal{U})$ also has a type. To obtain a type of $x$ we can just replace any urelement involved in $x$ (possibly at some depth) immediately after colon $::$ by its type. The resulting type of $x$ will hopefully
be much simpler than the object \( x \) itself because many different elements involved in \( x \) (at some depth) may have identical types. Also an element may have many different types. For example, empty set has arbitrary set type, \( 0::\alpha, \alpha \in \text{HFA}(\mathcal{W}) \setminus \mathcal{W} \), and if \( x::\alpha \) and \( x \subseteq \beta \) as HF-sets then \( x::\beta \). Generally denote \( x \subseteq \beta := \forall x::\alpha.(x::\beta) \). Evidently, if \( \alpha = \{a_1:x_1, \ldots, a_k:x_k\} \) and \( x_i \subseteq \beta_i \) then \( \alpha \subseteq \{a_1:\beta_1, \ldots, a_k:\beta_k\} \).

Let \( x, y, \ldots ::\alpha \) be some objects (quasituples) of type \( \alpha \). Of course, type \( \alpha \) contains important information about these objects. So, any such data \( x \) can be pictured (e.g. in the screen display of a computer) in the form of a table, in fact, a column (cf. Fig. 1), where \( \alpha(x), \alpha(y), \ldots \) are the table representations of information about \( x, y, \ldots \) which, together with \( \alpha \), completely determine \( x, y, \ldots \). It could be said that \( \alpha(x) \) is "\( x \) from the point of view of \( \alpha \)" or "\( x \) organized via type \( \alpha \)" or "\( x \) minus \( \alpha \)". To specify the detailed shape for \( x \) let us consider two cases: (1) \( \alpha \in \mathcal{W} \) and (2) \( \alpha \notin \mathcal{W} \). In the first case it should be \( x \in \mathcal{W} \). Then let \( \alpha(x) \) be just \( x \). In the second case \( \alpha = \{a_1:x_1, \ldots, a_k:x_k\} \) and therefore \( x = \{a_1:x_{11}, \ldots, x_{1n_1}; a_2:x_{21}, \ldots, x_{2n_2}; \ldots; a_k:x_{k1}, \ldots, x_{kn_k}\} \) where \( n_i \geq 0 \) and \( x_{ij}::\alpha_i, 1 \leq i \leq k, 1 \leq j \leq n_i \) and, moreover, for each \( i \) the named objects \( a_i:x_{ij} \) are all objects in \( x \) with attribute \( \alpha_i \) such that \( x_{ij}::\alpha_i \). Note that accidentally some \( a_i \) may coincide for different \( i \) and corresponding lists \( x_{i1}, \ldots, x_{in_i} \) may intersect. Therefore, such representation of \( x \) may be somewhat overloaded, but nevertheless correct reflect the typing nature of \( x \). The more detailed picture for \( x \) is shown in Fig. 2. Here \( a_i \) and \( x_{ij} \) are all objects in \( x \) with attribute \( \alpha_i \) such that \( x_{ij}::\alpha_i \). Note that accidentally some \( a_i \) may coincide for different \( i \) and corresponding lists \( x_{i1}, \ldots, x_{in_i} \) may intersect. Therefore, such representation of \( x \) may be somewhat overloaded, but nevertheless correct reflect the typing nature of \( x \). The more detailed picture for \( x \) is shown in Fig. 2. Here \( a_i \) and \( a_i(x_{ij}) \) indicate that \( a_i \) is the attribute of \( x_{ij} \).
\( x_i(x_{ij}) \) are pictured analogously (and can be enlarged or reduced in screen display if necessary). Consider a simple example of such a picture (Fig. 3). This data (birth date, learner group, advancement in various subjects and friends of) characterizes student Petrov. Note that in this example some (underlined) atomic types (NAME, INT, WORD) may seem somewhat superfluous by overloading the information. If we have a type UNIV (universal) in \( \mathcal{U} \) such that \( u : \mathcal{U} \) UNIV for each \( u \in \mathcal{U} \), then we may replace in any type \( \alpha \) each atomic subtype by the type UNIV. Evidently all statements \( x : \alpha \) will be preserved after this replacement. In the table form we may also omit all these atomic subtypes. Then the heading of the above table will look even simpler (Fig. 4).

However, unlike urelements, we evidently cannot have a universal type for all objects because \( x : \alpha \) implies \( \text{depth}(x) \leq \text{depth}(\alpha) \) (and conversely in the case of empty \( \mathcal{U} \)). To have such information about all students, we consider data pictured roughly as shown in Fig. 5 (so that it can be displayed on-screen). Any data \( x \in \mathcal{HFA}(\mathcal{U}) \) can be characterized not only by its type but also by some integrity conditions. For example, by following notations in Fig. 2 for our example concerning Petrov, we can require

\[
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
\text{STUD_NAME} & \text{BIRTH_DATE} & \text{GROUP} & \text{ADVANCEMENT} & \text{FRIENDS} & \text{FRIENDS} \\
\hline
\text{NAME} & \text{YEAR} & \text{MONTH} & \text{SUBJ} & \text{YEAR} & \text{MARK} & \text{FRIENDS} \\
\hline
\text{Petrov_A.} & 1968 & April & 345 & \text{Log} & 1988 & 4 \\
\hline
\end{array}
\]

or even as

\[
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
\text{STUD_NAME} & \text{BIRTH_DATE} & \text{GROUP} & \text{ADVANCEMENT} & \text{FRIENDS} & \text{FRIENDS} \\
\hline
\text{YEAR} & \text{MONTH} & \text{SUBJ} & \text{YEAR} & \text{MARK} & \text{FRIENDS} \\
\hline
\end{array}
\]

Fig. 3.

or even as

\[
\begin{array}{|c|c|c|c|c|c|c|}
\hline
\text{STUD_NAME} & \text{BIRTH_DATE} & \text{GROUP} & \text{ADVANCEMENT} & \text{FRIENDS} \\
\hline
\text{YEAR} & \text{MONTH} & \text{SUBJ} & \text{YEAR} & \text{MARK} \\
\hline
\end{array}
\]

Fig. 4.
that \( n_1 = n_2 = n_3 = 1 \) for STUD_NAME \( (n_1) \), BIRTH_DATE \( (n_2) \) and GROUP \( (n_3) \), and that there should be no restriction on the numbers \( n_4, n_5, n_6 \) of data corresponding to subtypes for ADVANCEMENT, FRIENDS:MANNNAME and FRIENDS:WOMANNNAME, respectively. Additionally, in the subtable for ADVANCEMENT we should require that all objects of type ADVANCEMENT be records. As to table for STUDENTS, it is natural to require that the functional dependency on student names holds.

A quasirelational nested data base of type \( \alpha \) can generally be defined as a class of possible data \( x \) also called states of this data base, which have the type \( \alpha \) and satisfy some additional integrity conditions \( \varphi(x) \) on these states. More formally, this class is \( \{x::\varphi(x)\} \). A function (or a program or specification defining it) \( T(x) \) on these states taking values in the universe \( \text{HFA}(\mathcal{U}) \) is called a data base query. The value \( y = T(x) \) is considered as an answer to the query \( T \) in the data base state \( x \). We may require that all answers have the same type, e.g. \( \forall x::x. (T(x)::\beta) \). In this case we say that \( T \) has the type \( \alpha \to \beta \).

Now our aim is to describe a general specification language for data base typing, integrity conditions and queries.

### 3. Set-theoretic \( \Delta \)-programming language

Let \( \mathcal{M} = (\mathcal{U}, \leq_\mathcal{U}, ::_\mathcal{U}, \mathcal{R}_1^{\mathcal{U}}, \mathcal{R}_2^{\mathcal{U}}, \ldots) \) be a first-order relational structure, where the set (or class) \( \mathcal{U} \) consists of urelements, \( \leq_\mathcal{U} \) is a linear order on \( \mathcal{U} \), ::\( \mathcal{U} \) is some binary relation on \( \mathcal{U} \) read as "... is of type ..." and \( \mathcal{R}_1, \mathcal{R}_2, \ldots \) are some other relations on \( \mathcal{U} \).

We consider the following \( \Delta \)-programming language over \( \text{HFA}(\mathcal{U}) \) which will consist of \( \Delta \)-formulas and \( \Delta \)-terms defined by simultaneous induction as follows. Its variables and \( \Delta \)-terms will have the values in \( \text{HFA}(\mathcal{U}) \) (and possibly in \( \mathcal{U} \subseteq \text{HFA}(\mathcal{U}) \)). Relations of the structure \( \mathcal{M} \) are naturally extended to the universe \( \text{HFA}(\mathcal{U}) \) as false for arguments not in \( \mathcal{U} \subseteq \text{HFA}(\mathcal{U}) \). For example, \( x \leq_\mathcal{U} y \) means that \( x, y \in \mathcal{U} \) and \( \mathcal{M} = x \leq_\mathcal{U} y \).
A-formulas are constructed from atomic ones $T \subseteq S$, $T \supseteq S$, $\bigwedge_{i=1}^{k}(T_1, \ldots, T_k)$, $T = S$, $A: T \subseteq S$ (or $T \subseteq S$) and $T \subseteq \mathcal{U}$ by using connectives $\&$, $\lor$, $\neg$, $\Rightarrow$ and bounded quantifiers $\forall a: x \in P, \exists a: x \in P$ binding variables $a, x$, where $S, T, T_j, P, A$ are $A$-terms and $P$ does not depend on $a, x$. Note that $\mathcal{U}$ is not a term. This is one place predicate or a class $\mathcal{U} \subseteq \text{HFA}(\mathcal{U})$. The semantics of $A$-formulas is evident and depends on the semantics of $A$-terms. Note that $A: T \subseteq S$ is false if the value of $A$ is not in $\mathcal{U}_0$ or if the value of $S$ is in $\mathcal{U}_0$.

$A$-terms are constructed from atomic ones $a, b, x, y, \ldots$ (variables over $\text{HFA}(\mathcal{U}))$, $\emptyset$ (the empty set constant) and from any previously constructed $A$-formulas $\varphi$ and $A$-terms $A_i, T$, $T, P, E, V, F$, where $P$ does not depend on $a, x$ and $A_i$ are terms deliberately taking values in $\mathcal{U}_0 := \mathcal{U} \cup \{\emptyset\}$ (e.g. if $A_i$ is the constant $\emptyset$ or a term of the kind $[T \in \mathcal{U}]$; see examples below), by using the following operators.

Explicit enumeration $\{A_1; T_1, \ldots, A_k; T_k\}$. This denotes a set of $\leq k$ (different) named elements because some pairs $A_i; T_i$ may incidentally have equal values in the universe.

Union $\bigcup \{T(a, x) | a: x \in P & \varphi(a, x)\}$. This is the ordinary union of all sets $T(a, x)$ where $a: x$ ranges over those named objects of the set $P$ for which $\varphi(a, x)$ holds. Therefore, variables $a, x$ are closed. Other free variables in $T, P$ and $\varphi$ remain free and are considered as parameters. Of course, they may become bounded in some external union construct or bounded quantification construct where this union may be inserted as a subterm. We additionally postulate that this union is equal to $S \subseteq \text{HFA}(\mathcal{U})$, if $\exists a: x \in P. \varphi(a, x)$ and $\forall a: x \in P. (\varphi(a, x) \Rightarrow T(a, x) = S)$ are true in $\text{HFA}(\mathcal{U})$. This addition is natural and especially important if $S$ is an urelement and does not contradict the above definition of the union if $S$ is a set. Without this addition the language would be incomplete. For example, $A$-definition of the if-then-else construct presented below is otherwise insufficient for urelement arguments. Of course, we could split this complex union into simpler constructs: simple union $\bigcup T$, taking the image $\{T(a, x) | a: x \in P\}$ and $A$-separation $\{a: x \in P | \varphi(a, x)\}$ (cf. examples below).

Inductive (nonmonotone) $A$-separation

$$\text{IND}[q = q \cup \{a: x \in P | \varphi(a, x, q)\}] \text{ or } \text{IND}[q = \{a: x \in P | a: x \in q \lor \varphi(a, x, q)\}].$$

This expression binds variables $q, a$ and $x$ and is considered as a term (not as a formula) of $A$-language. It denotes a distinguished solution $q_\alpha \subseteq P$ of the equation (written between square brackets) computed iteratively as the result of stabilizing the following monotonic sequence of finite subsets of $P$:

$$q_0 := \emptyset \subseteq q_1 \subseteq \cdots \subseteq q_\beta \subseteq \cdots \subseteq P, \exists n. (q_n = q_{n+1} = q_\omega),$$

$$q_{n+1} := q_n \cup \{a: x \in P | \varphi(a, x, q_n)\};$$

Note that if $\varphi$ is inflationary in $q$ ([21, 22], i.e. $a: x \in q \Rightarrow \varphi(a, x, q)$ always holds) or monotonic (i.e. $p \subseteq q \& \varphi(a, x, p) \Rightarrow \varphi(a, x, q)$ holds) then the same $q_\omega$ is also the (least,
in the monotonic case) solution of the more simple equation \( q = \{ a : x \in P \mid \varphi(a, x, q) \} \) with the same sequence \( q_o \).

Transitive closure \( \text{TC}(T) \) denotes the set of named elements \( a : y \) of set \( T \), named elements of these elements \( y \), etc., up to urelements. Thus, \( \text{TC}(T) \) is a kind of history of building set \( T \).

Collapse operation \( C(E, F, V) \) (see Section 4). This finishes the description of syntax and semantics (escape collapsing) of \( A \)-language.

The following are useful examples of \( A \)-formulas, \( A \)-terms and their abbreviations, some of which could be taken for convenience as primitive \( A \)-constructs:

\[
T \in S := \exists a : x \in S. (T = x);
\]

\[
\forall x \in T. \varphi(x) := \forall a : x \in T. \varphi(x);
\]

\[
\bigcup \{ T(x) \mid x \in P \land \varphi(x) \} := \bigcup \{ T(x) \mid a : x \in P \land \varphi(x) \};
\]

\[
T \subseteq S := \forall a : x \in T. (a : x \in S);
\]

\[
T \subseteq S := T \subseteq S \land S \subseteq T;
\]

\[
\{ A : T \} := \text{singleton set, or labelled (named) by } A \text{ object } T;
\]

\[
\text{Subset}(S) := \exists a : x \in S. \forall b : y \in S. (a = b \land x = y);
\]

\[
\bigcup T := \bigcup \{ x \mid x \in T \}, \text{for example, } \bigcup \{ A : T \} = T;
\]

\[
T \setminus S := \bigcup \{ T, S \};
\]

\[
\emptyset(T) := \{ a : a \in T \};
\]

\[
\{ A(a, x) : T(a, x) \mid a : x \in P \land \varphi(a, x) \} := \bigcup \{ \{ A : T \} \mid a : x \in P \land \varphi \};
\]

\[
\{ T(a, x) \mid a : x \in P \land \varphi(a, x) \} := \emptyset(T(a, x)) \mid a : x \in P \land \varphi(a, x);\]

\[
\{ a : x \in P \mid \varphi(a, x) \} := \{ a : x \mid a : x \in P \land \varphi(a, x) \}, \text{for example}
\]

\[
\emptyset = \{ x \mid y \in \emptyset \} \land \varphi \land \neg \varphi;
\]

\[
T \setminus \emptyset := \{ a : x \in T \mid a : x \notin \emptyset \};
\]

if \( \varphi \) then \( T \) else \( S \) := \( \bigcup \{ x \mid \in T, S \mid (\varphi \Rightarrow x = T) \land (\neg \varphi \Rightarrow x = S) \};\)

\[
\text{if } \varphi \text{ then } T \text{ else } \emptyset := \text{if } T \in \emptyset \text{ then } T \text{ else } \emptyset;
\]

\[
\emptyset \cup T := \{ x \mid a : x \in T \} \land \emptyset \mid a : x \in T \} \land \emptyset \text{ - uniformization of } T;
\]

\[
\langle S, T \rangle := \{ S, S \cup T \} \land \text{ ordered pair coding for the case when } S \text{ and } T \text{ are singletons};
\]

\[
\langle A : S, B : T \rangle := \langle \{ A : S \}, \{ B : T \} \rangle \land \langle S, T \rangle := \langle \emptyset : S, \emptyset : T \rangle;
\]

\[
\pi_1(P) := \text{if } P = \{ \bigcup P \} \text{ then } \bigcup P \text{ else } \bigcup (P \setminus \{ \bigcup P \});
\]

\[
\pi_2(P) := \text{if } P = \{ \bigcup P \} \text{ then } \bigcup P \text{ else } \bigcup P \setminus \pi_1(P).
\]
It follows that $\pi_1$ and $\pi_2$ give the first and the second projections of any pair $\langle S, T \rangle$ if $S$ and $T$ are singletons, i.e. $\pi_1(\langle a:x, b:y \rangle) = \{a:x\}$ and $\pi_2(\langle a:x, b:y \rangle) = \{b:y\}$. Define
\[
\text{pair}(z) := z = \langle \pi_1(z), \pi_2(z) \rangle \land \text{singleton}(\pi_1(z)) \land \text{singleton}(\pi_2(z)).
\]

Other natural coding of ordered pairs could be given as
\[
\langle S, T \rangle := \{\text{FIRST} : S, \text{SECOND} : T\}
\]
where FIRST and SECOND are some distinguished different urelements (attributes), if any. Define also
\[
\text{Dom}(R) := \bigcup \{\pi_1(z) \mid z \in R \land \text{pair}(z)\};
\]
\[
\text{Range}(R) := \bigcup \{\pi_2(z) \mid z \in R \land \text{pair}(z)\};
\]
\[
\text{Field}(R) := \text{Dom}(R) \cup \text{Range}(R);
\]
\[
\{a:x\} \times T := \{ \langle a:x, b:y \rangle \mid b:y \in T \} \quad \text{direct product with a singleton};
\]
\[
S \times T := \bigcup \{ \{a:x\} \times T \mid a:x \in S \} \quad (= \{ \langle a:x, b:y \rangle \mid a:x \in S \land b:y \in T \});
\]
\[
T^2 := T \times T.
\]
The support of $x$ is defined by
\[
\mathcal{V}[x] := \{u \in \mathcal{U} \mid a:u \in TC(\{x\})\} \cup \{u \in \mathcal{U} \mid a:y \in TC(\{x\})\}
\]
and consists of those urelements from which $x$ is constructed. For example, $\mathcal{V}[x]$ is the same urelement. Note that we are using $\mathcal{V}[x]$ not as denotation for “$x \in \mathcal{U}$”. Evidently, there holds

**Proposition 3.1.** All $\Delta$-definable operations $T(x)$ (including collapsing operation defined below in Section 4) preserve support, i.e. $\mathcal{V}[T(x)] \subseteq \mathcal{V}[x]$ is true for any $x \in \text{HFA}(\mathcal{U})$.

The disjoint union or direct finite sum is defined by $\sum_{i \in I} T(i) := \sum \{T(i) \mid i \in I\} := \bigcup \{T(i) \times \{i\} \mid i \in I\}$ or $T_1 + T_2 + \cdots + T_n := T_1 \times \{1\} \cup T_2 \times \{2\} \cup \cdots \cup T_n \times \{n\}$, where $0, 1, \ldots, n$ are ordinals of some other different objects of the universe. Then
\[
[S]_i := \bigcup \{\pi_1(z) \mid z \in S \land z = \langle \pi_1(z), \{0:i\} \rangle\} \quad \text{with}
\]
\[
A:T \in [S]_i \iff \langle A:T, i \rangle = \langle \{A:T\}, \{0:i\} \rangle \in S,
\]
\[
\left[ \sum_{i \in I} T(i) \right]_j = T_j, \quad \text{for } i \in I,
\]
\[
[T_1 + T_2 + \cdots + T_n]_j = T_j, \quad \text{for } j = 1, 2, \ldots, n.
\]
The following is the legal inductive definition in $\Delta$-language of the initial well-founded (or acyclic) part $\mathcal{W}.\mathcal{T}(R)$ of any (finite) binary relation $R \in \text{HFA}(\mathcal{U})$.
Hereditarily-finite sets

considered as a set of pairs:

\[ \mathcal{W} \mathcal{F}(R) := \text{IND} \left[ w = \{ x \in \text{Field}(R) \mid \forall y \in \text{Field}(R). \right. \]

\[ \langle \langle y, x \rangle \in R \to y \in w \} \].

This well-founded part \( \mathcal{W} \mathcal{F}(R) \subseteq \text{Field}(R) \) is obtained by adjoining to \( \mathcal{W} \mathcal{F}(R) \) step-by-step, up to stabilizing, those vertices in \( \text{Field}(R) \) whose all immediate predecessors have been adjoined at previous steps. In particular, the first adjoined vertices are initial ones, which have no predecessors.

The following construction, whose expressive power may be shown to be equivalent to the inductive \( \Delta \)-separation operator, is useful. This is the \textit{operator} of (nonmonotonic) inductive definition of \( \Delta \)-predicates or classes \( \mathcal{P} \subseteq \text{HFA}(\mathcal{\mathcal{M}}) \) (which may be infinite and therefore not elements of \( \text{HFA}(\mathcal{\mathcal{M}}) \))

\[ \text{IND}\left[ \mathcal{P} = \{ x \mid \varphi(x, \mathcal{P} \cap T(x)) \} \right]. \]

Here \( \varphi(x, y) \) is any \( \Delta \)-formula with set variables \( x \) and \( y \), and \( \Delta \)-term \( T \) (considered as the generalized transitive closure operator) is required to satisfy conditions (i) \( x \in T(x) \) and (ii) \( y \in T(x) \Rightarrow T(y) \subseteq T(x) \). The semantics of this operator is the corresponding infinite (actually, “locally finite”) limit \( \mathcal{P}_\omega = \bigcup_{n \geq 0} \mathcal{P}_n \) which can be seen to satisfy the equation for \( \mathcal{P} \).

Note, that if \( \varphi(x, y) \) is inflationary or monotonic in \( y \) then this construction may be equivalently rewritten as

\[ \text{IND}\left[ \mathcal{P} = \{ x \mid \varphi(x, \mathcal{P} \cap T(x)) \} \right]. \quad (#) \]

The original general version is easily reduced to an inflationary one by replacing \( \varphi(x, y) \) by \( x \in y \lor \varphi(x, y) \).

**Proposition 3.2.** (a) **Operator \((#)\)** for inflationary or monotonic \( \varphi \) is equivalent to the definition

\[ \mathcal{P} := \{ y \mid y \in v(y) \} \text{ (where } v(y) = \mathcal{P} \cap T(y) \text{) and inductively} \]

\[ v(y) = \{ x \in T(y) \mid \varphi(x, v(y) \cap T(x)) \} \text{ (inductive } \Delta\text{-separation).} \]

(b) Conversely, inductive \( \Delta \)-separation \( v = \{ x \in y \mid \varphi(x, v) \} \) for \( \varphi \) inflationary or monotonic in \( v \) is reduced to an inductive definition of the kind \((#)\) as follows:

\[ v = r \cap v^- = T^- \subseteq r, \text{ where inductively} \]

\[ v^- = \{ x \mid x \in r \land \varphi(x, v^- \cap T(r \cup \{ x \})) \}. \]

**Proof.** (a) Let \( \mathcal{P}_0 := \emptyset \) (as a class), \( \mathcal{P}_{n+1} := \{ x \mid \varphi(x, \mathcal{P}_n \cap T(x)) \} \), and \( v_0 := \emptyset \), \( v_{n+1}(y) := \{ x \in T(y) \mid \varphi(x, v_n(y) \cap T(x)) \} \). Then we have \( v_n(y) = \mathcal{P}_n \cap T(y) \). For \( n = 0 \) this is trivial. Then \( v_{n+1}(y) = \{ x \in T(y) \mid \varphi(x, \mathcal{P}_n \cap T(x)) \} \) by induction hypothesis and by condition (ii) on \( T \) and, therefore, \( = T(y) \cap \mathcal{P}_{n+1} \), as required.
Then, by (i) \( y \in T(y) \), we have \( \mathcal{P}_\alpha = \{ y \mid y \in v_n(y) \} \). It follows also, as required, that \( \mathcal{P}_\alpha = \{ y \mid y \in v_n(y) \} \) and \( v_n(y) = \mathcal{P}_\alpha \cap T(y) \) where \( \mathcal{P}_\alpha = \bigcup_n \mathcal{P}_n \), and \( v_n(y) = \bigcup_n v_n(y) \) are monotonic infinite and finite limits, respectively, and the solutions for \( \mathcal{P} \) and \( v(y) \).

(b) Here \( v_n = \mathcal{V}_n \subseteq r \) for all \( n \) and, therefore, both sequences stabilize and \( v_n = \mathcal{V}_n = r \). It is because \( v_{n+1} = \{ x \in r \mid \phi(x, v_n) \} \subseteq r \subseteq \mathcal{T}(r \cup \{ x \}) \) and therefore \( v_n \) and \( \mathcal{V}_n \) satisfy the same recurrent equation \( v_{n+1} = \{ x \in r \mid \phi(x, v_n \cap \mathcal{T}(r \cup \{ x \})) \} \) (equal to \( \{ x \in r \mid \phi(x, v_n) \}) \).

A simple example of using this construct is the (monotonic) inductive definition of A-relation \( \phi \) given in Section 2. Another is the following (monotonic) inductive A-definition of lexicographical linear ordering \( x < y \) which canonically extends a given linear ordering \( <_y \) on urelements to the whole universe:

\[
x < y \iff x <_y y \vee (x \neq y \& y \neq \mathcal{V} \& \exists a: a \in y \wedge \forall b: b \in x \wedge y)
\]

(b) \( a \neq a \& v < u \) \vee (\( a = a \& v = u \) \vee b \in x \cap y \)).

Note that the formula under quantifiers could be rewritten in a shorter form as \( a \neq b \vee \exists (b \in x \cap y \lor b \in x \cap y \neq a \) or, in a more useful (although nonmonotonic, in fact nonpositive) form, as \( a : b < v \Rightarrow b : b \in x \cap y \) with the same effect.

The recursive A-separation operator can also simulate simultaneous recursion with \( \phi_i \) inflationary or monotonic in \( q_1, \ldots, q_m \):

\[
q_1 = \{ a: x \in P_1 \mid \phi_1(\{ a: x \}, q_1, \ldots, q_m) \},
\]

\[
q_2 = \{ a: x \in P_2 \mid \phi_2(\{ a: x \}, q_1, \ldots, q_m) \},
\]

\[
q_m = \{ a: x \in P_m \mid \phi_m(\{ a: x \}, q_1, \ldots, q_m) \},
\]

where for convenience we consider that \( \phi_i \) depend on \( \{ a: x \} \) instead of simply \( a, x \). First, consider one set \( P_1 + \cdots + P_m \) instead of sets \( P_1, \ldots, P_m \). Then the sum \( q = q_1 + \cdots + q_m \) gives monotonic and bijective correspondence between subsets \( q \subseteq P_1 + \cdots + P_m \) and families of subsets \( \{ q_i \} = q_i \subseteq P_i \), \( i = 1, \ldots, m \), and the above system of equations is reduced to the unique one

\[
q = \Sigma_i \{ a: x \in P_i \mid \phi_i(\{ a: x \}, [q], \ldots, [q]_m) \},
\]

or, equivalently, to the equation having the form of inductive A-separation:

\[
q = \left\{ z \in P_1 + \cdots + P_m \left| \bigwedge_{i=1}^m (\pi_2(z) = i) \& \phi_i(\pi_1(z), [q], \ldots, [q]_m) \right. \right\}.
\]

(Note, that \( \pi_1(z) \) is a singleton for \( z \in P_1 + \cdots + P_m \)).

Note, that A-language may be interpreted almost word-for-word not only in \( \mathbf{HF} \) but in much more general universes also involving infinite sets, for example, in any \( \mathbf{ZF} \)-universe which contains \( \mathbf{HF} \) as an element. The minor exception is the stabilization process in the definition of inductive A-separation which will require transfinite steps in this case. Also, the binary relation \( < \) defined in this section is no longer
a linear order. The corresponding more general problem of characterizing the expressibility of \( \Lambda \)-language will be considered in a separate paper (cf. also the abstract [47]).

4. Collapsing operation

To complete the description of the semantics of \( \Lambda \)-language we should define the collapsing operation \( C(e, v) \) in \( HFA(\mathcal{H}) \). This operation allows transformation of vertices \( v, v', \ldots \) and edges \( a:v \rightarrow v' \) of any (finite) graph \( e \) to sets and a membership relation between these sets. Therefore this is a very powerful tool to directly construct hereditarily-finite sets (and data of a data base) according to any given "plan" in the form of a graph.

We will relate (and identify) with any set \( e \in HFA(\mathcal{H}) \) a graph with labelled edges \( a:v' \rightarrow v \) which are represented by all labelled pairs \( a: \langle v', v \rangle \in e, a \in \mathcal{H}, \) or by corresponding triples \( (a, v', v) \). Elements of \( e \) which are not such pairs are ignored. Then, collapsing associates with each vertex \( v \) of \( e \) and generally with each \( v \in HFA(\mathcal{H}) \) an object \( C(e, v) \) in the universe essentially according to the following self-explanatory picture where \( a_i : v_i \rightarrow v, 1 \leq i \leq k, k \geq 0, \) are all edges in \( e \) having \( v \) as the end vertex:

\[
C(e,v) = \{ a_1 : C(e,v_1), \ldots, a_k : C(e,v_k) \}
\]

\[
C(e,v) = v \text{ if } v \in \mathcal{H} \text{ and } v \text{ is an initial vertex.}
\]

The general definition is inductive:

\[
C(e,v) := \begin{cases} v & \text{if } v \in \mathcal{H} \text{ and } v \text{ is an initial vertex of } e \text{ (i.e. } \forall a : p \in e \text{ (} p \neq \langle - , v \rangle )) \text{) } \\
\text{then } v \text{ else } & \\
\text{if } v \text{ is in the well-founded (i.e. initial acyclic) part of graph } e \text{ (cf. examples in Section 3) } \\
\text{then } \{ a : C(e,v') \mid a : \langle v', v \rangle \in e \} \text{ else } \emptyset. & \\
\end{cases}
\]

Thus, the set \( C(e, v) \) is built up with the help of \( \{ , \} \) and in accordance with the edges of \( e \) from \( \emptyset \) and from those urelements \( v \) which are initial vertices of the graph \( e \) by using labels of edges in \( e \) as corresponding attributes.

If we restrict \( C \) only to well-founded graphs \( e \) then this definition becomes simpler. Another natural version \( C_1 \) of \( C \) with \( C_1(e,v) = v \) for all initial vertices \( v \) proves to be equivalent to the original one from the point of view of \( \Lambda \)-definability as follows. First,
$C_1(e,v) = C(e',v)$ where, roughly, $e' = e \cup \{a:v_1 \rightarrow v_2 \mid a:v_1 \in v_2 \epsilon TC(\{v\})\}$. More precisely, vertices of $e$ and of $TC(\{v\})$ should be replaced by their suitable copies so that possible collisions disappear. Second, $C(e,v) = C_1(e',v')$ where $e'$ is the result of replacing each initial vertex not in $\mathcal{U}$ by $\emptyset$ and $v'$ is the vertex in $e'$ corresponding to given vertex $v$ in $e$.

Also, without loss of generality, we may restrict $C$ to graphs $e \in \text{HFA}(\mathcal{U})$ in which urelements of the universe $\text{HFA}(\mathcal{U})$ can serve only as initial vertices of the graph $e$ or as labels of its edges and, therefore, $C(e,v) = v$ holds for all $v \in \mathcal{U}$. Otherwise, replace any edge $a:v_1 \rightarrow v_2$ in $e$ by $a:v'_1 \rightarrow v'_2$ where $v'$ can be defined, for example, as $v' := \begin{cases} v & \text{if } v \in \mathcal{U} \\ \{v\} & \text{if } v \notin \mathcal{U} \end{cases}$. The resulting graph $e'$ is as required and isomorphic to $e$.

Any two tuples $(e,v)$ and $(e',v')$ are called $\mathcal{U}$-isomorphic, $(e,v) \cong_{\mathcal{U}} (e',v')$, if $e$ is isomorphic to $e'$ as graphs with the isomorphism identical on those initial vertices of $e$ and $e'$ which are urelements (the same for $e$ and $e'$), and if, moreover, $v'$ corresponds to $v$ via this isomorphism. Evidently

$(e,v) \cong_{\mathcal{U}} (e',v')$ implies $C(e,v) = C(e',v')$.

Unfortunately, the above inductive definition of collapsing, as well as that of transitive closure, do not fit in the form of inductive $\Delta$-separation. Both $C$ and $TC$ should be considered as initial constructs because they are not definable in the rest of $\Delta$-language. Actually, for each of them there exists a corresponding coding of $\text{HF}$-sets such that, relative to this coding, the operation under consideration is not polynomial computable, although all operations expressible in the rest of the language are polynomial computable. These considerations, which are mainly concerned with the pure universe $\text{HF}$, are postponed to another paper.

As a simple application of collapsing and of linear order $<$ on the universe (cf. Section 3) we may define the cardinality of any set $a$ (which is a finite ordinal) as $\mathcal{C}(\alpha\epsilon\beta)(a) = C(\langle \alpha\epsilon\beta, \max(a) \rangle)$, where $e = (a := \langle \{x,y\} \mid x < y \rangle$ is the linear order on $a$ induced by $<$ and $\max(a) := \{x \in a \mid \forall y \in a. (y < x)\}$ is the maximal element in $a$.

5. Preserving the isomorphism of data by constructs of $\Delta$-language

Operation $C$ defined above allows one to represent each element $x$ of the universe $\text{HFA}(\mathcal{U})$ as $x = C(E(x), x)$, where

$E(x) := \{a:u,v \mid a:u,b:v \in TC(\{x\}) \& a:u \epsilon v\}$

with attributes $b$ ignored) is the graph with labelled edges corresponding to the membership relation on the transitive closure of $\{x\}$ and with the vertex $x$ distinguished. The second argument $x$ of $C$ is considered here only as the "atomic" graph...
vertex. Thus, the graph $E(x)$ with the vertex $x$ distinguished contains all necessary information about the actual set $x$.

Remember, that $\mathbb{U}$ is the underlying set of the first-order structure $\mathcal{M}$ to which $\Delta$-language is relativized. Therefore it is also reasonable to consider for any $x \in \text{HFA}(\mathcal{M})$ the following structure (which is somewhat richer than $(E(x), x)$ and derived additionally from that part of $\mathcal{M}$ which is contained in the support of $x$):

$$G(x) := G_{\mathcal{M}}(x) := \langle W(x), E(x), \mathbb{U}[x], \leq_{\mathcal{M}} \upharpoonright \mathbb{U}[x], \mathbb{A}_{\mathcal{M}} \upharpoonright \mathbb{U}[x], \ldots \rangle.$$  

Here for each $x$ the set $W(x) := \bigcup \{ \{a, v', v\} \mid a : (u', v) \in E(x) \}$ is the underlying set of vertices and labels. Other components $E(x) \subseteq \mathbb{U}[x] \times W(x)^2$ (where, strictly speaking, $E(x)$ is not the subset of $\mathbb{U}[x] \times W(x)^2$ and therefore it should be slightly modified),

$$x \in W(x),$$

$$\mathbb{U}[x] \subseteq W(x),$$

$$\leq_{\mathcal{M}} \upharpoonright \mathbb{U}[x] \subseteq \mathbb{U}[x]^2 \subseteq W(x)^2,$$  

are considered, respectively, as ternary two-sorted relation (in $a, u, v$) over the sets $\mathbb{U}[x]$ and $W(x)$, the vertex distinguished, the one-place “support predicate” on $W(x)$ or a subset of the sort $W(x)$, and the restrictions of basic relations $\leq_{\mathcal{M}}$, etc., of $\mathcal{M}$ to these “support elements”.

Let $\sigma$ be the signature (= similarity type) of such structures $G(x)$. We will also use $\sigma$ as the name of $\Delta$-definable class $\sigma \subseteq \text{HFA}(\mathcal{M})$ of all $\sigma$-structures $g = \langle w, e, v, u; \leq_{\sigma}, \ldots \rangle \in \text{HFA}(\mathcal{M})$ possibly not of the form $G(x)$. Also, let

$$g_1 \equiv_\sigma g_2$$

denote the ordinary isomorphism relation between $\sigma$-structures.

We do not assert that $\equiv_\sigma$ is $\Delta$-definable because it seems inevitably to involve quantification over bijections from $w_1$ to $w_2$ which is not bounded in our sense. It would be proved bounded if we were to introduce into $\Delta$-language the powerset operation $2^x := \{ y \mid y \subseteq x \}$ which gives the set of all subsets of arbitrary set $x$. However, the powerset goes beyond intuition on computability with bounded resources. Note, that extending $\Delta$-language by the powerset operation or by “stronger” bounded quantifiers $\forall x \in T$ and $\exists x \in T$ with inclusion $\subseteq$ instead of membership $\in$ gives rise to Kalmar elementary computability or the polynomial-time hierarchy [48], respectively, instead of the polynomial-time computability.

Elements of $u \subseteq w$ are called support elements of $g$ and the corresponding structure $g_u = \langle u; \leq_u, \ldots \rangle$ of the same signature as $\mathcal{M}$ is called the support structure of $g$. In general we may not require $u \subseteq \mathbb{U}$. However, this special case and even the case of $u = w \cap \mathbb{U}$ is particularly important.

Two objects $x$ and $y$ in the universe $\text{HFA}(\mathcal{M})$ are called isomorphic,
if there are corresponding $\sigma$-structures $G(x)$ and $G(y)$, i.e. $G(x) \cong_\sigma G(y)$. Note, that such an isomorphism $x \leq y$ is uniquely determined by its restriction to corresponding (possibly different) support sets $\mathcal{U}[x]$ and $\mathcal{U}[y] \subseteq \mathcal{U}$ of structures $G(x)$ and $G(y)$. (However, the analogous assertion cannot hold for arbitrary $\sigma$-structures.) More generally, $\bar{x} \leq \bar{y}$ for $\bar{x} = x_1, \ldots, x_n$, $\bar{y} = y_1, \ldots, y_n$ means that \{ $x_1, \ldots, x_n$ $\} \cong $ $\{ $ y_1, \ldots, y_n $ \} with $x_i$ corresponding to $y_i$ as vertices of graphs $G(\bar{x}) = G(\{x\})$ and $G(\bar{y}) = G(\{y\})$, respectively. Also let

$$x \equiv_\mathcal{U} y$$

mean that $x \leq y$ with $\equiv$ identical on $\mathcal{U}[x] = \mathcal{U}[y]$. This relation $\equiv_\mathcal{U}$ over the objects in the universe is analogous to the previously defined isomorphism relation $\cong_\mathcal{U}$ over the pairs $(e, e)$. Evidently,

$$x \equiv_\mathcal{U} y \text{ implies } x = y.$$

It can be proved by induction over the $\Delta$-syntax that

**Proposition 5.1.** In $\text{HFA}(\mathcal{U})$ $\Delta$-predicates are invariant under isomorphism of data and $\Delta$-operations transform isomorphic arguments to isomorphic ones, respecting their supports.

This means that the following two conditions hold.

(i) For any $\Delta$-definable predicate $\varphi$ always

$$\bar{x} \equiv \bar{y} \text{ implies } \varphi(\bar{x}) \Leftrightarrow \varphi(\bar{y}).$$

(ii) More generally, $\Delta$-definable operations transform isomorphic objects to isomorphic ones so that, moreover, their supports (which are also preserved, by Proposition 3.1) are respected.

Here respecting of supports by an operation $F$ over $\text{HFA}(\mathcal{U})$ means that if a bijection $H$ between $\mathcal{U}[\bar{x}]$ and $\mathcal{U}[\bar{y}]$ determines an isomorphism $\bar{x} \equiv \bar{y}$ (in fact, an isomorphism $\cong_\sigma$ of $\sigma$-structures $G(\bar{x})$ and $G(\bar{y})$) then the same $H$ induces the bijection between corresponding support subsets $\mathcal{U}[F(\bar{x})] \subseteq \mathcal{U}[\bar{x}]$ and $\mathcal{U}[F(\bar{y})] \subseteq \mathcal{U}[\bar{y}]$ (supports are preserved!) which, in turn, determines an isomorphism $F(\bar{x}) \cong F(\bar{y})$ (in fact, an isomorphism of $\sigma$-structures $G(F(\bar{x}))$ and $G(F(\bar{y}))$).

Clauses (i) and (ii) may be rewritten equivalently as

(i') $\varphi(H(\bar{x})) \Leftrightarrow \varphi(\bar{x})$ and

(ii') $H(T(\bar{x})) = H(T(\bar{x}))$,

where bijection $H : \{ z \mid \mathcal{U}[z] \subseteq u_1 \} \to \{ z \mid \mathcal{U}[z] \subseteq u_2 \}$ is the unique extension of any isomorphism $h = H \mid u_1 : u_1 \to u_2$, $u_1, u_2 \subseteq \mathcal{U}$, between corresponding substructures $\mathcal{M} \upharpoonright u_1$ and $\mathcal{M} \upharpoonright u_2$ of the underlying structure $\mathcal{M}$ which satisfies the equivalence $x \equiv y \Leftrightarrow H(x) \equiv H(y)$ for every $x, y \in \{ z \mid \mathcal{U}[z] \subseteq u_1 \}$.

In the case of pure universe $\text{HF}$ instead of $\text{HFA}(\mathcal{U})$ both (i) and (ii) trivially hold for all predicates and operations independently on their $\Delta$-definability.
If we consider, in place of $G(x)$, weaker structures based only on $E(x)$ without taking into account the $\mathcal{M}$-part of $\text{HFA}(\mathcal{M})$ then the above properties of invariance and preserving isomorphism of objects would fail because $\mathcal{M}$-part is actually presented in $\mathcal{A}$-language by corresponding predicate symbols $\preceq, \vdash, \mathcal{R}_1$, etc.

6. Polynomial-time computability over $\text{HFA}(\mathcal{M})$

To define polynomial-time computability over $\text{HFA}(\mathcal{M})$ we need the following simple, although, somewhat tedious, technical considerations on coding the objects of the universe by finite linearly ordered graphs via collapsing. Linear order arises from representation of graphs as inputs and outputs of a Turing machine. Roughly speaking, any operation $F: \text{HFA}(\mathcal{M}) \to \text{HFA}(\mathcal{M})$ is said to be polynomial computable if it is polynomial computable by using such codes for the arguments and values of $F$. This kind of definition is strongly dependent on the choice of the coding (cf. Section 7), and a graphical representation of sets seems natural. Actually, all the details of such a definition of computability over $\text{HFA}(\mathcal{M})$ are considerably more transparent for the “pure” universe $\mathcal{H}$ than for $\text{HFA}(\mathcal{M})$ (cf. [42, 43]), the last general case deserving the special attention of this paper.

Let $\text{CODES}=\{\langle g, \preceq_g \rangle\} \subseteq \text{HFA}(\mathcal{M})$ be the $\mathcal{A}$-definable class of all $\sigma$-structures $g=\langle w; e, v, u; \preceq_u, \ldots \rangle$ augmented with a linear order $\preceq_g$ on $w$ and, moreover, such that

(a) $u = w \cap \mathcal{M}$,
(b) elements of $u$ can serve only as initial vertices or as labels of the edges of the graph $e$ (cf. the corresponding note after the definition of collapsing) and
(c) $g_u \subseteq \mathcal{M}$, i.e. the corresponding support structure $g_u = \langle u; \preceq_u, \ldots \rangle$ is a substructure (= the restriction on $u$) of the underlying structure $\mathcal{M} = \langle \mathcal{M}; \preceq_u, \ldots \rangle$.

It follows that $g_u$ is linearly ordered by the relation $\preceq_u = \preceq_g \upharpoonright u$ (possibly different from $\preceq_g \upharpoonright u$; however, we could additionally require in the definition of CODES that $\preceq_u = \preceq_g \upharpoonright u$).

Due to the $g_u$-part of codes $g$ all $\mathcal{A}$-definable operations (which may involve $\mathcal{M}$-predicates) will be polynomial-time computable relative to the coding map defined below $\mathcal{C}: \text{CODES} \to \text{HFA}(\mathcal{M})$ (cf. Theorem 6.1). On the other hand, the converse statement is also true and is based on using linear orders $\preceq_g$ on $w$, $\preceq_g$ on $\mathcal{M}$ and $\preceq$ on the whole universe. (Compare also the discussion in [22] on the role of linear order and on the abstract polynomial-time computable global predicates in the case of finite first-order structures.)

Each pair $\langle g, \preceq_u \rangle$ in CODES is considered as a code of a set $\mathcal{C}(\langle g, \preceq_u \rangle) := C(e, v)$ in $\text{HFA}(\mathcal{M})$ (with the ordering $\preceq_g$ and $g_u$-part actually ignored by $C$. Thus, $\mathcal{C}: \text{CODES} \to \text{HFA}(\mathcal{M})$, or even $\mathcal{C}: \text{HFA}(\mathcal{M}) \to \text{HFA}(\mathcal{M})$, is a $\mathcal{A}$-definable operation.

Evidently, $\mathcal{H}[\mathcal{C}(\langle g, \preceq_u \rangle)] = u$ always and $\mathcal{C}$ takes isomorphic $\sigma$-structures from CODES into isomorphic sets:

$g_1 \equiv_\sigma g_2$ implies $\mathcal{C}(\langle g_1, \preceq_1 \rangle) \equiv \mathcal{C}(\langle g_2, \preceq_2 \rangle)$.
Moreover,

\[ g_1 \cong_{s,g} g_2 \text{ implies } \mathcal{C}(\langle g_1, \leq_1 \rangle) = \mathcal{C}(\langle g_2, \leq_2 \rangle), \]

where \( g_1 \cong_{s,g} g_2 \) means that simultaneously \( g_1 \cong_s g_2, \ u_1 = u_2 \) and the isomorphism \( \cong_s \) is identical on \( u_1 \) (which implies \( (e_1, v_1) \cong g(e_2, v_2) \) and \( C(e_1, v_1) = C(e_2, v_1) \)).

We say that any mapping \( H : \text{CODES} \rightarrow \text{CODES} \) defines an operation \( F : \text{HFA}(\mathcal{A}) \rightarrow \text{HFA}(\mathcal{A}) \) if

\[ F \circ \mathcal{C} = \mathcal{C} \circ H \]

holds (i.e. if the corresponding square diagram commutes).

Let us \( \lambda \)-define a right inverse operation to \( \mathcal{C} \):

\[ \mathcal{G} : \text{HFA}(\mathcal{A}) \rightarrow \text{CODES} \subseteq \text{HFA}(\mathcal{A}), \]

\[ \mathcal{G}(x) := \langle G(x), \leq | W(x) \rangle, \]

where \( \leq \) is the lexicographical linear order on \( \text{HFA}(\mathcal{A}) \) induced by the linear order \( \leq_w \) on urelements and represented in \( \lambda \)-language (cf. Section 3). Then, evidently, \( \mathcal{G}(x) \) is the (canonical) code of \( x \), i.e. \( \mathcal{C}(\mathcal{G}(x)) = x \) and, therefore, any operation \( F : \text{HFA}(\mathcal{A}) \rightarrow \text{HFA}(\mathcal{A}) \) which is defined by some \( H \) (i.e. satisfies \( F \circ \mathcal{C} = \mathcal{C} \circ H \)) may be represented as

\[ F = \mathcal{C} \circ H \circ \mathcal{G}. \]

It follows that \( F \) is \( \lambda \)-definable by \( H \) and is also invariant under isomorphic variations, in the sense of \( \cong_{s,g} \), of (values of) \( H \) due to the corresponding property of \( \mathcal{C} \) and \( \mathcal{G} \) mentioned above. If \( H \) is defined only up to the weaker isomorphism relation \( \cong_s \) then \( F = \mathcal{C} \circ H \circ \mathcal{G} \) is not determined uniquely, but only up to isomorphism relation \( \cong \) (of values of \( F \)).

**Definition.** Any operation \( F : \text{HFA}(\mathcal{A}) \rightarrow \text{HFA}(\mathcal{A}) \) is called polynomial-time computable if there exists a mapping \( H : \text{CODES} \rightarrow \text{CODES} \) which defines \( F \) and which is polynomial-time computable in the following special sense (cf. the auxiliary definition below).

First note that elements \( \langle g, \leq_g \rangle \), of \( \text{CODES} \) with \( g = \langle w; e, v, u; \leq_w, \ldots \rangle \) cannot be treated directly as inputs or outputs of a Turing machine because they are still abstract HF-objects and also involve urelements \( u \in \mathcal{U} \) which are considered here as objects of an arbitrary nature. However, for each such code \( \langle g, \leq_g \rangle \) there exists its unique isomorphic “hard” copy \( \langle g_0, \leq_0 \rangle_0 = \langle g_0, \leq_0 \rangle, \ g_0 = \langle w_0; e_0, v_0, u_0; \leq_{w_0}, \ldots \rangle, \ g \cong g_0, \ \leq \cong \leq, \) with \( w_0 \) and \( \leq_0 \) being an initial segment of natural numbers and its natural ordering, respectively. This copy may be written on a Turing machine tape in any natural way (for example, \( k \)-place predicates may be considered as \( k \)-dimensional \( 0 \)-\( 1 \)-matrices and written linearly row by row). The isomorphism type (or, equivalently, the “hard” copy) of any output \( \langle \tilde{g}, \leq_{\tilde{g}} \rangle, \ \tilde{g} = \langle w; \tilde{e}, \tilde{v}, \tilde{u}; \ldots \rangle, \) of some mapping \( H : \text{CODES} \rightarrow \text{CODES}, H : \langle g, \leq_g \rangle \mapsto \langle \tilde{g}, \leq_{\tilde{g}} \rangle, \) may incidentally depend only on the
isomorphism type of corresponding input \( (g, \leq_g) \). Then such \( H \) induces uniquely corresponding transformation \( h_0 : (g, \leq_g)_0 \mapsto (\tilde{g}, \leq_{\tilde{g}})_0 \) of hard copies of codes.

Conversely, given arbitrary (or only polynomial-time computable) transformation \( h_0 \) of hard copies of codes from CODES, the corresponding \( H : \text{CODES} \rightarrow \text{CODES} \) evidently exists. However, \( H \) is determined only up to isomorphic variations of its values in the sense of \( \cong_\sigma \) (not of \( \cong_\alpha, \gamma \)). Unfortunately, as was noted above, this does not suffice to determine the unique operation \( F : \text{HFA}(\mathcal{M}) \rightarrow \text{HFA}(\mathcal{M}) \) (if any) such that \( F = \mathcal{G} \circ H \circ \mathcal{G} \). Therefore, we need to compute additionally some embedding of support set \( (\tilde{u})_0 \) of the resulting (output) hard code \( (\tilde{g}, \leq_{\tilde{g}})_0 \) into actual urelements \( \mathcal{U} \subseteq \text{HFA}(\mathcal{M}) \) or, by another approach, embedding of \( (\tilde{u})_0 \) into support set \( (u)_0 \) of the initial (input) hard code \( (g, \leq_g)_0 \) corresponding (via unique isomorphisms) to actual inclusion \( \tilde{u} \subseteq u \) (which is required in this case).

Denote by \( h \) any such transformation of hard codes together with some embeddings of output support sets to urelements \( \mathcal{U} \) or to input support sets. Now, the values of the corresponding \( H \) are determined up to \( \cong_{\alpha, \gamma} \). Conversely, any \( H \) (or any \( H \) satisfying the condition \( \tilde{u} \subseteq u \) and) inducing some \( h_0 \) also induces a corresponding \( h \) in the sense of the first (or second) approach.

The first approach demands some way of interpreting urelements from \( \mathcal{U} \) in hard inputs and outputs and therefore presupposes that they are not objects of an arbitrary general nature. However, the second approach, which we adopt here, is quite general in this sense. Evidently, we should pay for such generality by the restriction to those \( F \), definable in this approach, which preserve supports of their arguments. This restriction seems quite natural, at least from the point of view of data bases: the response on any query about a finite data base state involves only data of this state. Therefore, we complete the above definition by the

\textbf{Auxiliary definition.} Any mapping \( H : \text{CODES} \rightarrow \text{CODES} \), \( H : (g, \leq_g) \mapsto (\tilde{g}, \leq_{\tilde{g}}), \ g = \langle \hat{w} ; e, v, u, \ldots \rangle, \ \tilde{g} = \langle \hat{w} ; \tilde{e}, \tilde{v}, \tilde{u}, \ldots \rangle \), satisfying the condition \( \tilde{u} \subseteq u \), is called \textit{polynomial-time computable in the special sense} if it induces polynomial-time computable transformation \( h \) of hard codes with the embeddings of support sets from outputs to inputs.

Evidently, we may consider that the polynomial bounds on the time of computation are taken with respect to a cardinality of input structures and even to a cardinality of a transitive closure of arguments for \( F \) (due to representation \( F = \mathcal{G} \circ H \circ \mathcal{G} \) and a definition of \( \mathcal{G} \) based eventually on TC).

It proves that any operation \( F \) based on an arbitrary transformation \( h \) of the above kind should not only preserve supports of the arguments, but also take isomorphic arguments to isomorphic values, respecting their supports (cf. clause (ii) in Section 5).

This property of \( F \) follows from the representation \( F = \mathcal{G} \circ H \circ \mathcal{G} \) and from the possibility of choosing for \( F \) a version of \( H : \text{CODES} \rightarrow \text{CODES} \) (in fact, of \( H : \text{HFA}(\mathcal{M}) \rightarrow \text{HFA}(\mathcal{M}) \), by \( \Delta \)-definability of CODES \( \subseteq \text{HFA}(\mathcal{M}) \)) which satisfies the
same property and corresponds to a given transformation \( h \). Actually, an appropriate \( H \) may be easily \( \Delta \)-defined by \( h \) (using a suitable analog of the \( \circ \) operation defined at the end of Section 4). Here we consider \( h \) as an operation \( h : HF \to HF \) where (some) pure sets naturally represent hard codes and corresponding embeddings. Therefore, satisfying clause (ii) for \( F \) is reduced to that for \( H \) and, finally, to that for \( h \). But for pure \( h \) this property evidently always holds (even if \( h \) is not \( \Delta \)-definable or not computable).

Moreover, we will show below that a version of \( H \) and, therefore, the corresponding \( F \) are \( \Delta \)-definable if the transformation \( h \) of hard codes is polynomial-time computable.

The definition of computability over \( HFA(\mathcal{M}) \) also implies that in the formulation of the theorem in the Abstract clauses (1) and (2) could be omitted, being implicit:

**Theorem 6.1.** \( \Delta \)-terms and \( \Delta \) formulas define exactly all operations and predicates in the universe \( HFA(\mathcal{M}) \) computable in polynomial time.

For this theorem we should also generalize the above definition to the case of computability of many-place set-theoretic operations \( F(x_1, \ldots, x_k) \). The case of set-theoretic predicates is trivially reduced to the case of operations.

Let us simply consider slightly richer code structures \( g = g^* = \langle w; e, v_1, \ldots, v_k, u; \ldots \rangle \) than in CODES (satisfying the same clauses (a), (b), (c) and) with several distinguished vertices \( v_i \) corresponding, by collapsing, to the arguments \( x_1, \ldots, x_k \). Note that given any structures \( g_i = \langle w_i; e_i, v_i, u; \ldots \rangle \), \( 1 \leq i \leq k \), which represent corresponding objects \( x_i \in HFA(\mathcal{M}) \) (via collapsing), we can \( \Delta \)-define the structure \( g \) which represents simultaneously all \( x_i \). We may simply let \( w = w_1 \cup \cdots \cup w_k \), \( u = u_1 \cup \cdots \cup u_k \), etc. if we first take care of \( w_i \cap w_j = u_i \cap u_j \) for all \( 1 \leq i, j \leq k \). No additional collisions arise because urelements of the universe \( HFA(\mathcal{M}) \) can serve only as initial vertices or as labels of all edges in graphs \( e_i \) (cf. clause (b) of CODES definition).

**Proof (sketch).** First, we will show how \( \Delta \)-language can be implemented in polynomial time (and even implemented by a real computer) by considering only three \( \Delta \)-language constructs: unordered pair operation \( \{a:x, b:y\} \), equality \( x = y \) and membership \( a:x \in y \) predicates. The cases of other constructs are based on analogous ideas and are left to the reader. We will operate immediately on CODES instead of their hard copies.

Computing \( z = \{a:x, b:y\} \) consists simply of constructing a graph \( g^* \) (representing \( z \)) from any given graph \( g^{xy} \) (representing \( x \) and \( y \)) as follows:

(a) add any new distinguished vertex \(*\) and edges \( a:v_1 \to *, b:v_2 \to * \) from the old distinguished vertices \( v_1 \) and \( v_2 \) corresponding to \( x \) and \( y \) while preserving all the old vertices and edges and

(b) let \( u^* = u^{xy} \) and \( g^*_u = g^{xy}_u \) (cf. Fig. 6).

Evidently, this procedure is correct and polynomial-time computable.
To decide the equality $x = y$ between any two objects represented by a graph $g^{xy} = \langle w; e; u, v_1, v_2; \ldots \rangle$, we essentially need to do the following:

(a) Transform this graph inductively to a “canonical extensional well-founded form”. Begin with the initial vertices and identify iteratively those pairs of vertices $v', v''$ from the well-founded part of (the current version of) $g^{xy}$ for which $\langle v', v'' \rangle \notin u^2$.

(b) Establish whether both distinguished vertices $v_1$ and $v_2$ were in the well-founded part of the original graph $g^{xy}$ and became identified.

(c) Let $v_1$ or $v_2$ not be in the well-founded part and therefore the corresponding object $x$ or $y$ be an empty set. Then we decide on the simpler equality $\emptyset = y$ or $x = \emptyset$, and check that the other vertex $v_2$ or $v_1$ is also not in the well-founded part or has no predecessors and is not an urelement.

In the case of success we have $x = y$, and conversely. This is proved by induction on the initial well-founded part of the graph. The procedure described is polynomial computable since the number of pairs of identified vertices is at most quadratic.

The membership predicate $a : x \in y$ is reduced to that of equality (and bounded existential quantification) as follows: Check if there exists a vertex $v$ in $g^{xy}$ with the edge $a : v \rightarrow v_2$ and such that $v_1$ is identified with $v$ in the above procedure of identifying vertices of $g^{xy}$.

Conversely, we should show how any polynomial-time computable operation $F$ over $\text{HFA}(\mathcal{A})$ can be represented in $\mathcal{A}$-language. By the above definitions, $F = \mathcal{G} \circ H \circ \mathcal{G}$ with $H$ inducing polynomial computable transformation $h$ of hard copies of codes with the supports inclusions. Therefore, given $h$, we need only find corresponding $\mathcal{A}$-definable version of $H : \langle g, \leq_g \rangle \mapsto \langle \hat{g}, \leq_{\hat{g}} \rangle$, $g = \langle w; e, v, u; \ldots \rangle$, $\hat{g} = \langle \hat{w}; \hat{e}, \hat{v}, \hat{u}; \ldots \rangle$, up to isomorphism relation $\cong_{\alpha, \beta}$ for output structures $\hat{g}$.

Note, that if the input structure $g = \langle w; e, v, u; \ldots \rangle$ contains $n$ elements then $\hat{g} = \langle \hat{w}; \hat{e}, \hat{v}, \hat{u}; \ldots \rangle$, being computable in polynomial time by $H$ (or $h$), contains $\leq n^k$ elements for some $k > 0$. Therefore, instead of $\hat{w}$ we may consider the set $w^k$ (cartesian power).

Then (the corresponding version of) the mapping $H$ becomes a (global) operator transforming predicates over any linear ordered $w$ to predicates over $w^k$, i.e. eventually...
again over $w$. Any such polynomial-time computable operator can be represented by a suitable $\omega$-recursive scheme over $w$ considered as a finite row of natural numbers (cf. Introduction and [20, 38]) or by suitable inductive definitions over first-order relational structure $\langle g, \leq_g \rangle$ [7, 21–25, 27, 28, 30, 51].

In any case, recursive/inductive definitions of predicates over $w$ can easily be simulated in our $\Delta$-language with the help of the inductive $\Delta$-separation construct. As the result, given any structure $\langle g, \leq_g \rangle$ in CODES, we will obtain a $\Delta$-definition of a new $\sigma$-structure $g^{[k]} = \langle w^k; u^k; \ldots \rangle$ corresponding to $g$ and of a linear order $\leq_{g^{[k]}}$ corresponding to $\leq_{\tilde{g}}$ with the underlying set $w^k$ and the support set $u^{[k]} \subseteq w^k$. The required embedding between support sets $\tilde{u} \subseteq u$ is realized by a corresponding finite (global) function $u^{[k]} \rightarrow u$, $u \subseteq w$, $u^{[k]} \subseteq w^k$, which, being polynomial computable, can also be $\omega$-recursively (and $\Delta$-) defined by $\langle g, \leq_g \rangle$. Finally, slightly redefine $\langle g^{[k]}, \leq_{g^{[k]}} \rangle$ to the isomorphic (version of) $\tilde{g} = \langle \tilde{w}; \ldots; \tilde{u}; \ldots \rangle$ and $\leq_{\tilde{g}}$ so that actually $\tilde{u} \subseteq u$.

By this construction, the resulting version of $H : \langle g, \leq_g \rangle \mapsto \langle \tilde{g}, \leq_{\tilde{g}} \rangle$ induces a given $h$, as required. This finishes the sketch of the proof of the theorem. $\square$

7. Concluding remarks and perspectives

The acyclic finite graph representation of HF-sets (also independently considered by Dahlhaus and Makowsky [12, 13]), being very natural, is not unique and probably not the best. The following two simple representations of HF-sets immediately suggest themselves.

The first is the ordinary bijective Akkerman’s encoding $e : \omega \rightarrow HF$ of sets by natural numbers defined as $e(2^{n_1} + 2^{n_2} + \cdots + 2^{n_k}) = \{e(n_1), \ldots, e(n_k)\}, \quad n_1 > n_2 > \cdots > n_k$. This corresponds exactly to the lexicographical ordering of the universe HF of pure sets $\Delta$-defined in Section 3 for the case HFA(\$). Unfortunately, even such a simple operation as singleton $\{x\}$ is very difficult to compute because it is represented by the exponentiation: $\{e(n)\} = e(2^n)$. This fundamental drawback arises because in defining $e$ there was no care for real computability, only for pure mathematical elegance.

Another encoding of sets is based on well-formed bracket expressions: $\{\}, \{\{\}\}, \{\{\}\{\}\}, \ldots$ Here, too, such simple sets as ordinals $0 := \emptyset \ := \{\}, \ 1 := 0 \cup \{0\}, \ 2 := 1 \cup \{1\}, \ldots, n + 1 := n \cup \{n\}, \ldots$ require an exponentially increasing number of brackets. Bracket expressions are also equivalent to trees, the latter being a partial case of graphical representation of sets.

It follows that graphical representation is more economical than the bracket type: the second reduces to the first in polynomial time, but not conversely. This also means that graphs allow more sets of HF (for example, more ordinals) to be denoted in a shorter way than the bracket expressions. At the same time it is interesting to note that the two classes of polynomial-time computable operations $HF \rightarrow HF$, based on graph and bracket codings, respectively, are not each included in the other. (The first of these classes was considered in this paper in the case of HFA(\$).) This situation
Hereditarily-finite sets proves to be typical for any pair of (so called regular) codings one of which is not polynomial reducible to the other.

Moreover, an even more economical version of the graphical representation of HF-sets (although not so natural) may be defined by considering graphs with edges augmented by natural numbers in decimal notation. This means that corresponding edges should be considered as chains of many ordinary edges. Then the graph consisting of, say, only one edge $v_1 \rightarrow 1000 \rightarrow v_2$ denotes (via collapsing) a hereditary singleton with a nesting rank equal to one thousand. This allows sets of enormously large ranks to be denoted very briefly. Actually, it proves that the operation $\text{Rank}(x)$ which gives the (ordinal) rank of any set $x$ is definable in our $\Delta$-language only by using both collapsing and transitive closure operations.

All these encodings, except that of Akkerman, may be characterized as polynomial-time computable and regular in an appropriate sense. They allow consideration of corresponding versions of $\Delta$-language for which the analog of the main theorem of this paper holds. Therefore we have a spectrum of $\Delta$-languages, each corresponding in its own way to polynomial-time computability, and they should be further investigated. (Some of results mentioned were obtained in cooperation with Leontjev.) It is possible that for different applications different versions of such languages will be appropriate. There might be a kernel $\Delta$-programming language (with $C$ and TC omitted and) with the possibility of “switching” to the appropriate $\Delta$-extensions and corresponding implementation of $\Delta$-programs on a computer device.

Our general aim is reaching a harmony between the abstract mathematical notions of finite objects (such as hereditarily-finite sets) and computer reality with its inherent resource boundedness. Yet we cannot speak about proper mathematical understanding of the nature of such finite objects from the bounded resource point of view because of the variety of formal representations (and corresponding intuitive images) of HF-sets. (A somewhat analogous situation was in the ordinary set theory of “all” finite or infinite sets, especially the well-known independence results of K. Gödel and P. Cohen.) In particular, we should consider suitable axiomatic(s) of Bounded set theory(ies) corresponding to $\Delta$-language(s) (cf. [41–43] and also [39, 46] where the constructive/nonconstructive nature of finite binary strings was axiomatically investigated in the framework of a bounded arithmetic).

A different computational interpretation of $\Delta$-language may be based on a reduction notion for $\Delta$-formulas (as for lambda terms). For example, $S \in \{x \in T \mid \varphi(x)\}$ is reducible to $S \in T \& \varphi(S)$ and $\forall x \in \{T_1, T_2\} \cdot \varphi(x)$ is reducible to $\varphi(T_1) \& \varphi(T_2)$, etc. (cf. [43] where corresponding normalizing and Church–Rosser properties are stated and used for proving a conservation result).

Another interesting version to be investigated is the case of an HF-like universe of sets for which some kind of antifoundation axiom may hold (cf. Aczel [2]). For example, we may consider a singleton set $\Omega$ which is its own unique element, $\Omega = \{\Omega\}$. This allows consideration of arbitrary finite graphs (even with cycles) as machine (i.e. internal) representations of some data bases and simultaneously use of the advantages of abstract (or high-level) set-theoretic programming (with bounded resources). We
also suppose that an $\Delta$-programming point of view may be useful in considering the connection of an antifoundation axiom with the concurrent communication systems (CCS) of Milner [29] stated by Aczel [2]. (Note that our attributes formally correspond to actions in Milner’s processes.) Thus, data bases may not be the only possible application of $\Delta$-programming. We are going to devote some future more technical papers to these subjects.

Acknowledgment

The author thanks A.B. Livchak who attracted his attention to Data Bases, and M.K. Valiev and F. Afrati for useful comments on preliminary versions of this paper.

References


