Formalization of the Algebra of Nominative Data in Mizar

Artur Korniłowicz
Institute of Informatics, University of Białystok,
Ciołkowskiego 1M, 15-245 Białystok, Poland
Email: arturk@math.uwb.edu.pl

Andrii Kryvolap, Mykola Nikitchenko, Ievgen Ivanov
Taras Shevchenko National University of Kyiv,
64/13, Volodymyrska Street, 01601 Kyiv, Ukraine,
Email: krivolapa@gmail.com, nikitchenko@unicyb.kiev.ua,
ivanov.eugen@gmail.com

***Abstract*—In the paper we describe a formalization of the notion of a nominative data with simple names and complex values in the Mizar proof assistant. Such data can be considered as a partial variable assignment which allows arbitrarily deep nesting and can be useful for formalizing semantics of programs that operate in real time environment and/or process complex data structures and for reasoning about the behavior of such programs.**

I. Introduction

THE importance of the problem of elaborating the theory of programming and connecting it with software development practice was recognized by many researchers. In particular, it was mentioned as one of the grand challenges in computing by T. Hoare in his influential talk "The Verifying Compiler: a Grand Challenge for computing research of the 21st century" [1] with the implication that an important step towards solution of this challenge is development of a verifying compiler that should have a high impact on software quality and reliability. More generally, one may argue that development of practical tools and methods of automatic static analysis of a program (e.g. model checking, verification against a formal specification using logical methods and theorem provers, etc.) that can make sure that it has the desired runtime properties before the program is run is an important research topic.

However, nowadays software is used in many application domains and the traditional idea of proving properties of the input-output relation associated with a program is not always sufficient.

For example, practically relevant safety properties of software in a real-time embedded system [2], [3] (that is a part of a larger hardware-software system which interacts with the physical environment using sensors and actuators) or a cyber-physical system [2], [3], [4] (e.g. that consist of networks of computing devices that interact with physical environment) normally can be expressed not in the form of a property of a program or its input-output relation itself, but in terms of admissible behaviors of a larger software-hardware system and an environment to which it belongs.

The way of proving such properties depends on the chosen mathematical model of the hardware and the environment and it is important to note that such a proof is practically relevant only under the assumption that the model relative to which safety is proven adequately represents the behavior of the real system and its environment, and the environment usually includes unknown and/or random elements, and the scope of the model usually is limited.

As a toy example, suppose that a program P can control the behavior of a physical system S by assigning and modifying the value of a certain parameter p. The behavior of S is described by a differential equation $\frac{d}{dt}x(t) = f(p,t,x(t))$, where f is some fixed real-valued function. Assume that:

– p is updated by P at discrete time moments $t = t_1, t_2, \ldots$ determined by P, between which p remains constant (so on each bounded closed time interval p can be considered a piecewise constant function);

– P controls S in the open-loop fashion, i.e. P has no or does not use feedback from S.

Then one may formalize the above mentioned equation as a switched system [5] where p is a switching signal (if P uses some feedback from S it may be considered as a kind of a hybrid dynamical system [6]) and assume that its solutions $t \mapsto x(t)$ defined on intervals of the form $[0,T)$, $T \in (0,+\infty) \cup \{+\infty\}$ describe all possible evolutions of the state of the system in continuous time which start at the initial time moment $t = 0$.

Suppose that we want to check that the composite ("cyber-physical") system consisting of S together with the program P and a computing device which executes it, which we will denote as "$S + P$", has the following property which we will call "$NONNEG$": if $x(0) \geq 0$, then $x(t) \geq 0$ for all real $t \geq 0$ for any solution x which is defined at $t = 0$ and cannot be continuously extended forward in time (i.e. the value x that describes some characteristic of the system S does not fall below 0, if it starts at or above 0).

In order to check, if it holds for a particular f and P, it is necessary to formulate precisely the semantics of P.

For example, suppose that f has a simple polynomial form $f(u,v,w) = uw^2$, so that the equation has the form $\frac{d}{dt}x(t) = p(t)x^2(t)$, and the program P is given in the source code form in some imperative programming language with C-like syntax as in Algorithm I (where $L1$–$L6$ are labels).

A common way of giving an operational semantics [7] to programs in languages of this type is to define the notion of a program state that includes the information about its current point of execution and the current content of its variables, and

Algorithm 1 Example

```
L1: i = 0;
L2: for (; i<10; i++) { // i=0,1,...,9
L3:   p = i; // assign p the value of i
L4:   sleep(1); // wait for 1 second
L5: }
L6:
```

define a state transition system that describes the possible paths of evolution ("runs") of the program state during execution. Then reasoning about the relation between the program states at different program execution points can be done using e.g. the Floyd-Hoare logic [8], [9], [10].

The content of variables in a program state is usually formalized as a function mapping names of program variables to their values (*variable assignment*). The commonly used notation for such an assignment has the form $[var_1 \mapsto value_1, var_2 \mapsto value_2, \dots]$, where var_i are variables and $value_i$ are the values that the variables have. In P we can consider i a normal read/write variable the value of which is stored in the memory or a CPU register. The variable p and the assignment to it can have different interpretations (e.g. normal memory location, a write-only register/hardware port, etc.).

Using the labels $L1$–$L6$ to identify program execution points, and variable assignments to represent the content of variables, a program state can be formalized as a pair "(*label*, *variable assignment*)", and a possible run of P can have the form of a sequence such as:
$(L1, [i \mapsto 0, p \mapsto 0])$, $(L2, [i \mapsto 0, p \mapsto 0])$,
$(L3, [i \mapsto 0, p \mapsto 0])$, $(L4, [i \mapsto 0, p \mapsto 0])$,
$(L5, [i \mapsto 0, p \mapsto 0])$, $(L2, [i \mapsto 0, p \mapsto 0])$,
$(L3, [i \mapsto 1, p \mapsto 0])$, $(L4, [i \mapsto 1, p \mapsto 1])$, ...

Obviously, in our case $NONNEG$ cannot be checked by looking at such runs of P alone, instead the behavior of S and the timing of interaction between P and S should be taken into account. In particular, runs of P should be augmented with timing information to obtain a switching signal p that determines the behavior of S and ultimately allows one to check if $NONNEG$ holds for $S + P$.

A convenient way to add timing information and the behavior of S to runs of P is to extend the program state, or, more specifically, extend the variable assignment with variables that represent time (t) and the state of S (x), although, or course, they differ in nature from i and p. Ignoring the execution time of instructions other than "sleep(1)", this extended idealized run of $S + P$ can have the form:
$(L1, [i \mapsto 0, p \mapsto 0, t \mapsto 0, x \mapsto x_0])$,
$(L2, [i \mapsto 0, p \mapsto 0, t \mapsto 0, x \mapsto x_0])$,
$(L3, [i \mapsto 0, p \mapsto 0, t \mapsto 0, x \mapsto x_0])$,
$(L4, [i \mapsto 0, p \mapsto 0, t \mapsto 0, x \mapsto x_0])$,
$(L5, [i \mapsto 0, p \mapsto 0, t \mapsto 1, x \mapsto x_1])$,
$(L2, [i \mapsto 0, p \mapsto 0, t \mapsto 1, x \mapsto x_1])$,
$(L3, [i \mapsto 1, p \mapsto 0, t \mapsto 1, x \mapsto x_1])$,
$(L4, [i \mapsto 1, p \mapsto 1, t \mapsto 1, x \mapsto x_1])$, ...

where $x_i = x(i)$ for a solution x of the switched system $\frac{d}{dt}x(t) = p(t)x^2(t)$, $x(0) = x_0$, where

$$p(t) = \begin{cases} i, & t \in [i, i+1), i \in \{0, 1, \dots, 9\}, \\ 9, & t \geq 10. \end{cases}$$

Note that if $x_0 \neq 0$, this solution x dominates the solution y of the initial value problem $\frac{d}{dt}y(t) = y^2(t)$, $y(1) = x_0$ for $t \geq 1$, which is $y(t) = 1/(1 + x_0^{-1} - t)$ and which has a finite time blow-up at $t = 1 + x_0^{-1}$ (i.e. $\lim_{t \to 1 + x_0^{-1}-} x(t) = \infty$), so x is undefined (and cannot be continuously extended) past the time $1 + x_0^{-1}$. The physical meaning of this situation is model- and application-specific, e.g. it may represent a physical event after which the current model cannot represent adequately S, or may be a modeling artifact. Some results and discussion of the physical meaning of finite time blow-up situations can be found e.g. in [11], [12], [13], [14].

In any case, such situations cannot be ignored during model analysis and property checking and has to be represented in a run of $S + P$. E.g., although if $x_0 \geq 0$, then $x(t) \geq 0$ on the maximal interval of its existence, which is bounded from above, $NONNEG$ does not hold since it requires $x(t) \geq 0$ to hold for all real $t \geq 0$ for such an x.

Although the model of S may lose its meaning after $t = 1 + x_0^{-1}$, depending on the model and application, the run of a program P may still have sense past this time moment (e.g. if S represents a physical system remotely controlled by a computer running P, a finite time blow-up indicates a critical failure in S, then P may continue running after this event).

Naturally, the situation can be represented by *partial variable assignments* in the extended run of $S + P$, e.g.:
$(L1, [i \mapsto 0, p \mapsto 1, t \mapsto 2])$,
means that x is genuinely undefined, but other variables are defined, meaningful and have assigned values at time $t = 2$.

Basically, a partial variable assignment d is a set of pairs of *names* and *values*, where names are chosen from some fixed set V, but not all elements of V may appear in d as names (e.g. $V = \{i, p, t, x\}$, $d = [i \mapsto 0, p \mapsto 1, t \mapsto 2]$). Such assignments are also called *nominative sets* [15].

In contrast, a variable assignment can be called *total*, if all elements of V appear in it as names (e.g. $V = \{i, p, t, x\}$, $d = [i \mapsto 0, p \mapsto 1, t \mapsto 2, x \mapsto 3]$).

Obviously, total variable assignments can be formalized mathematically as total functions on V and partial assignments (nominative sets) can be formalized as partial functions on V.

Reasoning about total variable assignments is well supported in tools that aid formal specification and verification of software, e.g. proof assistants [16] such as Isabelle, Coq, PVS, etc. In particular, Isabelle/HOL "record" data types are convenient for introducing total assignments with a predefined set of names. Otherwise, they may be formalized as functions on a type of names.

On the other hand, partial variable assignments are usually implemented manually on top of total assignments using option types (data types that add a special "none"/undefined value to an existing type), and a built-in library of basic operations on them that reflect program operations is generally not available. Another issue is that built-in or library data types that represent partial variable assignments with (arbitrarily deep) nesting such as

$$\begin{aligned}&[\ local\_vars \mapsto [i \mapsto 1, j \mapsto 0],\\ &\ \ IO\_interface \mapsto [p \mapsto 1],\\ &\ \ global\_time \mapsto 1,\\ &\ \ systemS\_state \mapsto [x \mapsto 1],\\ &\ \ anotherSystemState \mapsto [\\ &\qquad\qquad subsystem1 \mapsto [y \mapsto 1, z \mapsto 1],\\ &\qquad\qquad subsystem2 \mapsto [y \mapsto 1, z \mapsto 1]]\ ]\end{aligned}$$

are also not available (i.e. partial variable assignments where the values of some variables themselves can be partial variable assignments). However, such assignments can be very useful to reflect the typical way in which data is grouped in programs and formalize programs that operate on complex data structures, e.g. such partial assignments naturally formalize data encoded in the popular JSON (JavaScript Object Notation) data format widely used in web applications.

In this paper we propose a library (written in the Mizar proof assistant [17], [18]) that provides a formalization of nominative sets and operations on them. The Mizar system has its own proof verifier[1] used to verify the logical correctness of proofs written in the Mizar language[2]. The system contains a very rich library (based on an axiomatic set theory [29]) of formalized mathematical theories called Mizar Mathematical Library (MML).[3,4] It has also a library support for the notion of a partial function without using option types. It is then well-suited for formalizing the mentioned partial variable assignments with nesting, and, more generally, has a potential for developing formalizations of models of real-time and cyber-physical systems and logics for reasoning about them.

We hope that it will be useful for formalizing models of hardware-software systems and applying logical methods to verification such systems. In this direction we developed an extension of the Floyd-Hoare logic [10] that takes advantage of partial variable assignments and also supports partially defined pre- and post-condition predicates. More information about it can be found in [10].

In our library we use the theory of *nominative data* [15], [37], [38] from the *composition-nominative approach* [37] to program semantics: partial variable assignments are formalized as so-called nominative data with simple (unstructured) names and complex (structured) values. They are also called *Type SC* (simple names, complex values) nominative data, or nominative data of the type TND_{SC} [15]. They have hierarchical structure and can be considered as labeled oriented trees with arcs labeled with names.

The set of nominative data over a given set of (simple/basic) names V and set of atomic (basic) values A is denoted as $ND(V,A)$ and is defined as follows [15]:

$$ND(V,A) = \bigcup_{k\geq 0} ND_k(V,A),$$

where

$$ND_0(V,A) = A \cup \{\emptyset\},$$
$$ND_{k+1}(V,A) = A \cup \left(V \xrightarrow{n} ND_k(V,A)\right), \quad k \geq 0.$$

where for any set X, $V \xrightarrow{n} X$ denotes the set of all partial functions from V to X, the domain of which is a finite set, and $\emptyset$ is the empty set which is considered to be the empty nominative data and alternatively denoted as []. The elements of $V \xrightarrow{n} X$ are also called nominative data with simple names and simple values, or *Type SS* nominative data over a set of names V and a set of values X.

If $v_1, v_2, \ldots, v_n \in V$ are pairwise different (simple) names, and $a_1, a_2, \ldots, a_n$ are elements of $ND(V,A)$, then $[v_1 \mapsto d_1, v_2 \mapsto d_2, \ldots, v_n \mapsto d_n]$ denotes the unique nominative data $d \in ND(V,A)$ such that the domain of d (the set on which d, as a function, is defined) is $\{v_1, v_2, \ldots, v_n\}$ and $d(v_i) = a_i$ for $i = 1, 2, \ldots, n$.

For example, if a, b, c are distinct names (arbitrary objects), then $[a \mapsto [a \mapsto 1, b \mapsto []], c \mapsto 2] \in ND(\{a,b,c\},\{1,2\})$.

The elements of A are called *atomic nominative data*, while the elements of $ND(V,A)\backslash A$ are called *non-atomic nominative data*.

The basic operations on Type SC nominative data are:

- *Naming* – creating a nominative data of the form $[v \mapsto d]$ from a given nominative data d and a name v (this is denoted as $\Rightarrow v(d)$), or a nominative data of the form $[v_1 \mapsto [v_1 \mapsto [v_2 \mapsto \ldots [v_n \mapsto d] \ldots]$ from a given nominative data d and a finite sequence of names $v_1, v_2, \ldots, v_n$ (this is denoted as $\Rightarrow v_1 v_2 \ldots v_n(d)$). For example, if v_3, v_4 are different names, then $\Rightarrow v_1 v_2([v_3 \mapsto 1, v_4 \mapsto 2]) = [v_1 \mapsto [v_2 \mapsto [v_3 \mapsto 1, v_4 \mapsto 2]]]$.
- *Denaming* – obtaining the value corresponding to a given name v or a sequence of names $v_1, v_2, \ldots, v_n$ in a given non-atomic nominative data d (i.e. $d \notin A$; if $d \in A$, denaming is undefined on d). I.e., if $d = [u_1 \mapsto a_1, u_2 \mapsto a_2, \ldots, u_m \mapsto a_m]$ and $v = u_i$ for some i, then a_i is the result of denaming the name v in d (it is denoted as $v \Rightarrow (d)$, so $u_i \Rightarrow (d) = a_i$; it is assumed that $v \Rightarrow (d)$ is undefined, if v is not among $u_1, \ldots, u_m$). For a sequence of names $v_1, \ldots, v_n$, denaming of a nominative data d is denoted as $v_1 v_2 \ldots v_n \Rightarrow_a (d)$ and has the following

[1]Research on using specialized external systems to increase computational power of the Mizar system is also conducted [19], [20], [21].

[2]The Mizar language is a declarative language designed to write mathematical documents. It contains rules for writing traditional mathematical items (e.g. definitions, theorems, proof steps, etc.). It also provides syntactic constructions to launch specialized algorithms (e.g. term identifications, term reductions [22], flexary connectives [23], definitional expansions [24]) which increase the computational power of the verifier (e.g. equational calculus [25], [26], properties of functors and predicates [27], [28]).

[3]MML contains developments on various domains of mathematics, including set theory, calculus, topology, lattice theory [30], group theory, category theory, algebra [31], rough sets [32], and others.

[4]Because of the size, the MML is also a subject of research on optimization of its structure, including the improvement of legibility of proofs [33], [34], [35] and removing duplications of theorems and definitions [36].

meaning: $v_1 v_2 \dots v_n \Rightarrow_a (d) = v_n \Rightarrow (\dots(v_2 \Rightarrow (v_1 \Rightarrow (d))\dots)$, (it is assumed that $v_1 v_2 \dots v_n \Rightarrow_a (d)$ is undefined, if $v_k \Rightarrow (\dots(v_2 \Rightarrow (v_1 \Rightarrow (d))\dots)$ is undefined for some $k \le n$). For example, if u, w are different names, then
$v \Rightarrow ([v \mapsto [w \mapsto 1], w \mapsto 2]) = [w \mapsto 1]$,
$vw \Rightarrow_a ([v \mapsto [w \mapsto 1], w \mapsto 2]) = 1$,
$u \Rightarrow ([w \mapsto 1])$ is undefined.

- *(Global) overlapping* is an operation with two arguments – non-atomic nominative data d_1, d_2 which means joining d_1 and d_2 and resolving name conflicts in the favor of the second argument d_2. It is denoted as $d_1 \nabla d_2$ or as $d_1 \nabla_a d_2$. So, if $d_1 = [u_1 \mapsto a_1, \dots, u_n \mapsto a_n]$ and $d_2 = [v_1 \mapsto b_1, \dots, v_m \mapsto b_m]$ and $u_{j_1}, u_{j_2}, \dots, u_{j_k}$ is the list of all elements of the set $\{u_1, \dots, u_n\} \setminus \{v_1, \dots, v_m\}$, then $d_1 \nabla d_2 = [u_{j_1} \mapsto a_{j_1}, \dots, u_{j_k} \mapsto a_{j_k}, v_1 \mapsto b_1, \dots, v_m \mapsto b_m]$. For example, if v_1, v_2, v_3, v_4 are pairwise different names, then
$[v_1 \mapsto 1, v_2 \mapsto 2] \nabla [v_2 \mapsto 3, v_4 \mapsto 4] = [v_1 \mapsto 1, v_2 \mapsto 3, v_4 \mapsto 4]$,
- *Local overlapping* is an operation with two arguments – nominative data d_1, d_2 and a name parameter u, which means the global overlapping of d_1 and $\Rightarrow u(d_2)$. It is denoted as $d_1 \nabla_a^u d_2$, so $d_1 \nabla_a^u d_2 = d_1 \nabla [u \mapsto d_2]$. For example, if v_1, v_2, v_3, v_4 are names and v_1, v_2 are different, then
$[v_1 \mapsto 1] \nabla_a^{v_2 v_3} [v_4 \mapsto 2] = [v_1 \mapsto 1, v_2 \mapsto [v_3 \mapsto [v_4 \mapsto 2]]]$.

The set $ND(V, A)$ together with the operations of naming, denaming, and local overlapping is called the algebra of nominative data of the type TND_{SC} [15, Definition 5].

II. Definition of Nominative Data in Mizar

We implemented our library in the form of a Mizar paper entitled NOMIN_1.MIZ [39], in which we formalized the carrier set and the operations of the algebra of nominative data of the type TND_{SC}.

Below we describe the main definitions and results available in this Mizar paper. Because of space limitations we omit the text of full formal proofs of the theorems and correctness conditions of definitions stated below and certain technical lemmas that are used only in these proofs, but note that these formal proofs were checked for correctness with the help of the Mizar system.[5]

We formally defined a nominative set (*NominativeSet*) over an arbitrary set of names V and an arbitrary set of atomic values A as a partial function (`PartFunc` [41]) from V to A as follows:

```
definition
 let V,A be set;
 mode NominativeSet of V,A is PartFunc of V,A;
end;
```

[5]The details on syntax and semantics of the Mizar language can be found in [40].

We defined the notion of a Type SS nominative data (*TypeSSNominativeData*) as a nominative set with the finite graph:

```
registration
 let V,A be set;
 cluster finite for NominativeSet of V,A;
end;
```

```
definition
 let V,A be set;
 mode TypeSSNominativeData of V,A
      is finite NominativeSet of V,A;
end;
```

We defined the set $NDSS(V, A)$ of all Type SS nominative data as follows:

```
definition
 let V,A be set;
 func NDSS(V,A) -> set equals
  the set of all D
    where D is TypeSSNominativeData of V,A;
end;
```

and proved that it is nonempty for all sets V and A:

```
registration
 let V,A be set;
 cluster NDSS(V,A) -> non empty;
end;
```

The following definitions introduce the notion of a type SC nominative (*TypeSCNominativeData*), including nonatomic nominative data (*NonatomicND*).

```
definition
 let V,A be set;
 let S be FinSequence;
 pred S IsNDRankSeq V,A means
 S.1 = NDSS(V,A) &
  for n being Nat st n in dom S & n+1 in dom S
   holds S.(n+1) = NDSS(V,A \/ S.n);
end;
```

```
definition
 let V,A be set;
 mode NonatomicND of V,A -> Function means
 ex S being FinSequence st S IsNDRankSeq V,A &
  it in Union S;
end;
```

```
definition
 let V,A be set;
 mode TypeSCNominativeData of V,A -> set means
 it in A or it is NonatomicND of V,A;
end;
```

```
definition
 let V,A be set;
 let D be TypeSCNominativeData of V,A;
 attr D is atomicND means
 D in A;
 attr D is non-atomicND means
 D is NonatomicND of V,A;
end;
```

```
registration
 let V be set; let A be non empty set;
 cluster atomicND
```

```
  for TypeSCNominativeData of V,A;
end;
```

```
registration
 let V,A be set;
 cluster non-atomicND
  for TypeSCNominativeData of V,A;
end;
```

where FinSequence denotes a finite sequence [42], . (dot) denotes a function application [43], dom is the domain of the function [44], and Union denotes the union of the codomain of the function [45].

Then, we proved several theorems suitable for defining new IsNDRankSeq sequences:

```
theorem
 for S being FinSequence st S IsNDRankSeq V,A
  for n being Nat st n in dom S holds
   S|n IsNDRankSeq V,A;
```

```
theorem
 for S being FinSequence st S IsNDRankSeq V,A
  holds
   S ^ <*NDSS(V,A\/S.len S)*> IsNDRankSeq V,A;
```

```
theorem
 for F being FinSequence st F IsNDRankSeq V,A
  ex S being FinSequence st len S = 1 + len F
   & S IsNDRankSeq V,A &
    for n being Nat st n in dom S holds
     S.n = NDSS(V,A \/ (<*A*>^F).n);
```

and two simple examples of such sequences:

```
theorem
 <*NDSS(V,A)*> IsNDRankSeq V,A;
```

```
theorem
 <*NDSS(V,A),NDSS(V,A \/ NDSS(V,A))*>
  IsNDRankSeq V,A;
```

where len is the length of the finite sequence [42], <* *> constructs finite sequences [42], ^ denotes the concatenation of two finite sequences [42], and | is the restriction of a function to a set [44].

Below we state several examples of the sets and types of nominative data introduced above:

```
theorem
 v in V & a in A implies v.-->a in NDSS(V,A);
```

```
theorem
 v in V & a in A implies
  v.-->a is NonatomicND of V,A;
```

```
theorem
 v in V & v1 in V & a1 in A implies
  v.-->(v1.-->a1) in NDSS(V,A \/ NDSS(V,A));
```

```
theorem
 v in V & v1 in V & a1 in A implies
  v.-->(v1.-->a1) is NonatomicND of V,A;
```

```
theorem
 v in V & v1 in V & a in A & a1 in A implies
  (v,v1)-->(a,a1) in NDSS(V,A);
```

```
theorem
 v in V & v1 in V & a in A & a1 in A implies
  (v,v1)-->(a,a1) is NonatomicND of V,A;
```

where v, v1, a, a1 are arbitrary objects, v.-->a is a one element function $\{[v,a]\}$ [46], and (u,v)-->(a,b) is a two element function $\{[u,a],[v,b]\}$ [47].

III. Operations on Nominative Data

We defined the denaming operation on nonatomic nominative data of Type SC for a single name ($v \Rightarrow (d)$) as follows:

```
definition
 let V,A be set
 let v be object;
 let D be NonatomicND of V,A
  such that v in dom D;
 func denaming(v,D) ->
      TypeSCNominativeData of V,A equals
 D.v;
end;
```

We defined the naming operation on nonatomic nominative data of Type SC for a single name ($\Rightarrow v(d)$) as follows:

```
definition
 let V,A be set;
 let v,D be object;
 assume D is TypeSCNominativeData of V,A;
 assume v in V;
 func naming(V,A,v,D) -> NonatomicND of V,A
  equals
 v .--> D;
end;
```

We defined the naming operation on nonatomic nominative data of Type SC for a sequence of names ($\Rightarrow v_1 v_2 \ldots v_n(d)$) as follows:

```
definition
 let V,A be set;
 let a be object;
 let f be V-valued FinSequence;
 assume len f > 0;
 func namingSeq(V,A,f,a) -> FinSequence means
 len it = len f &
  t.1 = naming(V,A,f.len f,a) &
  for n being Nat st 1 <= n < len it holds
   it.(n+1) = naming(V,A,f.(len f-n),it.n);
end;
```

```
definition
 let V,A be set;
 let f be V-valued FinSequence;
 let a be object;
 func naming(V,A,f,a) -> set equals
 namingSeq(V,A,f,a).len(namingSeq(V,A,f,a));
end;
```

where V-valued states that the range of a relation is included in V [44].

Below we state several basic properties of the introduced operations:

```
theorem
 for f being V-valued FinSequence holds
  1 <= n <= len f implies
   namingSeq(V,A,f,a).n is NonatomicND of V,A;
```

```
theorem
```

```
 for f being V-valued FinSequence st len f > 0
  holds naming(V,A,f,a) is NonatomicND of V,A;
```

```
theorem
 for V being non empty set
  for v being Element of V holds
   naming(V,A,<*v*>,a) = naming(V,A,v,a);
```

```
theorem
 for V being non empty set
  for v1,v2 being Element of V st
   for D being TypeSCNominativeData of V,A
    holds
    naming(V,A,<*v1,v2*>,D) = v1.-->(v2.-->D);
```

The following theorem shows that denaming applied to the result of naming applied to a data d results in d, if denaming and naming concern the same name v:

```
theorem
 for D being TypeSCNominativeData of V,A holds
  v in V implies
   denaming(v,naming(V,A,v,D)) = D;
```

The following theorem states an identity for naming applied to the result of denaming:

```
theorem
 v in dom D implies
  naming(V,A,v,denaming(v,D)) = v .--> D.v;
```

We defined the global overlapping on nominative data of Type SC as follows:

```
definition
 let V,A be set;
 let d1,d2 be object such that
  d1 is TypeSCNominativeData of V,A and
  d2 is TypeSCNominativeData of V,A;
 func global_overlapping(V,A,d1,d2)
      -> TypeSCNominativeData of V,A means
 ex f1,f2 being Function st f1 = d1 & f2 = d2
  & it = f2 \/ (f1|(dom(f1)\dom(f2)))
   if not d1 in A & not d2 in A
    otherwise
     it = the TypeSCNominativeData of V,A;
end;
```

We defined the local overlapping with single name parameter v on nominative data of Type SC as follows:

```
definition
 let V,A be set;
 let d1,d2,v be object;
 func local_overlapping(V,A,d1,d2,v)
      -> TypeSCNominativeData of V,A equals
 global_overlapping(V,A,d1,naming(V,A,v,d2));
end;
```

Below we state several basic properties of the global overlapping:

```
theorem
 for d1,d2 being NonatomicND of V,A
  st not d1 in A & not d2 in A holds
   global_overlapping(V,A,d1,d2)
    = d2 \/ (d1|(dom(d1)\dom(d2)));
```

```
theorem
 for d1,d2 being NonatomicND of V,A
  st not d1 in A & not d2 in A
    & dom d1 c= dom d2
   holds global_overlapping(V,A,d1,d2) = d2;
```

```
theorem
 v in V &
  not v.-->a1 in A & not v.-->a2 in A &
  a1 is TypeSCNominativeData of V,A &
  a2 is TypeSCNominativeData of V,A
   implies
  global_overlapping(V,A,v.-->a1,v.-->a2)
   = v.-->a2;
```

Below we introduce the set $ND(V, A)$ of all nominative data of Type SC over a set of names V and a set of values A. This is the carrier of the algebra of nominative data.

```
definition
 let V,A be set;
 func ND(V,A) -> set equals
 the set of all D
  where D is TypeSCNominativeData of V,A;
end;
```

```
registration
 let V,A be set;
 cluster ND(V,A) -> non empty;
end;
```

$ND(V, A)$ can be also expressed as the union of the range of the function $FNDSC(V, A)$ defined as:

```
definition
 let V,A be set;
 func FNDSC(V,A) -> Function means
 dom it = NAT & it.0 = A &
 for n being Nat holds
  it.(n+1) = NDSS(V,A \/ it.n);
end;
```

Finally, we defined the operations of the algebra of nominative data as functions on $ND(V, A)$ as follows:

```
definition
 let V,A be set;
 let v be object;
 func denaming(V,A,v)
   -> PartFunc of ND(V,A),ND(V,A) means
 dom it = ND(V,A) \ A &
 for D being NonatomicND of V,A st not D in A
  holds it.D = denaming(v,D);
end;
```

```
definition
 let V,A be set;
 let v be object;
 func naming(V,A,v)
   -> Function of ND(V,A),ND(V,A) means
 for D being TypeSCNominativeData of V,A holds
  it.D = naming(V,A,v,D);
end;
```

```
definition
 let V,A be set;
 let v be object;
 func local_overlapping(V,A,v)
   -> PartFunc of [:ND(V,A),ND(V,A):],ND(V,A)
  means
 dom it = [: ND(V,A) \ A , ND(V,A) \ A :] &
```

```
  for d1,d2 being NonatomicND of V,A
   st not d1 in A & not d2 in A holds
  it. [d1,d2] = local_overlapping(V,A,d1,d2,v);
end;
```

where `[:A,B:]` is the Cartesian product of sets `A` and `B` [48].

IV. Conclusion

We have proposed a library of Mizar definitions of the carrier set and the operations of the algebra of nominative data of the type TND_{SC} (nominative data with simple names and complex values) which are essentially partial variable assignments that allow arbitrarily deep nesting. We have also formalized theorems that describe basic properties of nominative data and operations on them in Mizar. The obtained results can be useful for formalizing semantics of programs that operate in real time environment and/or process complex data structures and for reasoning about the behavior of such programs. We plan to formalize an extension of the Floyd-Hoare logic [10] in Mizar that allows reasoning about programs by taking advantage of the formalized notion of a nominative data in further papers. We plan to continue the work described in this paper as follows: 1) To introduce notions of predicates and functions on nominative data in Mizar – predicates on nominative data will be used to represent the semantics of pre- and postconditions and functions on nominative data can serve as semantic models of programs. 2) To define operations on partial functions and predicates on nominative data which represent semantics of common programming language constructs such as sequential execution, branching, cycle, etc. Sets of predicates and functions on nominative data together with such operations form a program algebra. 3) To define a special Floyd-Hoare composition using the introduced notions of predicates and functions (programs) on nominative data. 4) To formulate inference rules for the Floyd-Hoare logic for programs on nominative data with partial pre- and postconditions and prove soundness of this inference system in Mizar.

References

[1] T. Hoare. (2004) The verifying compiler: A grand challenge for computing research. Gresham College, 18/03/2004, Barnard's Inn Hall. http://www.cs.ox.ac.uk/files/6187/Grand.pdf.

[2] J. Shi, J. Wan, H. Yan, and H. Suo, "A survey of cyber-physical systems," in *Wireless Communications and Signal Processing (WCSP)*. IEEE, 2011, pp. 1–6.

[3] E. Lee and S. Seshia, *Introduction to embedded systems: A cyber-physical systems approach*. Lulu.com, 2013.

[4] J. Sifakis, "Rigorous design of cyber-physical systems," in *Embedded Computer Systems (SAMOS), 2012 International Conference on*. IEEE, 2012, pp. 319–319.

[5] D. Liberzon, *Switching in Systems and Control (Systems & Control: Foundations & Applications)*. Birkhauser Boston Inc., 2003.

[6] R. Goebel, R. G. Sanfelice, and A. Teel, "Hybrid dynamical systems," *Control Systems, IEEE*, vol. 29, no. 2, pp. 28–93, 2009.

[7] H. Nielson and F. Nielson, *Semantics with applications – a formal introduction*, ser. Wiley professional computing. Wiley, 1992.

[8] R. Floyd, "Assigning meanings to programs," *Mathematical aspects of computer science*, vol. 19, no. 19–32, 1967.

[9] C. Hoare, "An axiomatic basis for computer programming," *Commun. ACM*, vol. 12, no. 10, pp. 576–580, 1969.

[10] A. Korniłowicz, A. Kryvolap, M. Nikitchenko, and I. Ivanov, "An approach to formalization of an extension of Floyd-Hoare logic," in *Proceedings of the 13th International Conference on ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge Transfer (ICTERI 2017), May 15–18, 2017, Kyiv, Ukraine, 2017.*, ser. CEUR Workshop Proceedings, V. Ermolayev, N. Bassiliades, H.-G. Fill, V. Yakovyna, H. C. Mayr, V. Kharchenko, V. Peschanenko, M. Shyshkina, M. Nikitchenko, and A. Spivakovsky, Eds., vol. 1844. CEUR-WS.org, 2017, pp. 504–523. [Online]. Available: http://ceur-ws.org/Vol-1844/10000504.pdf

[11] J. Ball, "Finite time blow-up in nonlinear problems," *Nonlinear Evolution Equations*, pp. 189–205, 1978.

[12] Y. Zhou, Z. Yang, H. Zhang, and Y. Wang, "Theoretical analysis for blow-up behaviors of differential equations with piecewise constant arguments," *Appl. Math. Comput.*, vol. 274, no. C, pp. 353–361, Feb. 2016. [Online]. Available: http://dx.doi.org/10.1016/j.amc.2015.10.080

[13] H. A. Levine, "The role of critical exponents in blowup theorems," *Siam Review*, vol. 32, no. 2, pp. 262–288, 1990.

[14] A. Goriely, *Integrability and nonintegrability of dynamical systems*. World Scientific Publishing Company, 2001, vol. 19.

[15] V. Skobelev, M. Nikitchenko, and I. Ivanov, "On algebraic properties of nominative data and functions," in *Information and Communication Technologies in Education, Research, and Industrial Applications*, ser. Communications in Computer and Information Science, V. Ermolayev, H. Mayr, M. Nikitchenko, A. Spivakovsky, and G. Zholtkevych, Eds. Springer International Publishing, 2014, vol. 469, pp. 117–138.

[16] F. Wiedijk, "The seventeen provers of the world. Foreword by Dana S. Scott." ser. Lecture Notes in Artificial Intelligence, F. Wiedijk, Ed. Springer-Verlag Berlin Heidelberg, 2006, vol. 3600.

[17] A. Grabowski, A. Korniłowicz, and A. Naumowicz, "Four decades of Mizar," *Journal of Automated Reasoning*, vol. 55, no. 3, pp. 191–198, October 2015. [Online]. Available: http://dx.doi.org/10.1007/s10817-015-9345-1

[18] G. Bancerek, C. Byliński, A. Grabowski, A. Korniłowicz, R. Matuszewski, A. Naumowicz, K. Pąk, and J. Urban, "Mizar: State-of-the-art and beyond," ser. Lecture Notes in Computer Science. Springer, 2015, vol. 9150, pp. 261–279.

[19] A. Naumowicz, "Interfacing external CA systems for Gröbner bases computation in Mizar proof checking," *International Journal of Computer Mathematics*, vol. 87, no. 1, pp. 1–11, January 2010. [Online]. Available: http://dx.doi.org/10.1080/00207160701864459

[20] ——, "SAT-enhanced Mizar proof checking," in *Intelligent Computer Mathematics – International Conference, CICM 2014, Coimbra, Portugal, July 7–11, 2014. Proceedings*, ser. Lecture Notes in Computer Science, S. M. Watt, J. H. Davenport, A. P. Sexton, P. Sojka, and J. Urban, Eds., vol. 8543. Springer, 2014, pp. 449–452. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-08434-3_37

[21] ——, "Automating Boolean set operations in Mizar proof checking with the aid of an external SAT solver," *Journal of Automated Reasoning*, vol. 55, no. 3, pp. 285–294, October 2015. [Online]. Available: http://dx.doi.org/10.1007/s10817-015-9332-6

[22] A. Korniłowicz, "On rewriting rules in Mizar," *Journal of Automated Reasoning*, vol. 50, no. 2, pp. 203–210, February 2013. [Online]. Available: http://dx.doi.org/10.1007/s10817-012-9261-6

[23] ——, "Flexary connectives in Mizar," *Computer Languages, Systems & Structures*, vol. 44, pp. 238–250, December 2015. [Online]. Available: http://dx.doi.org/10.1016/j.cl.2015.07.002

[24] ——, "Definitional expansions in Mizar," *Journal of Automated Reasoning*, vol. 55, no. 3, pp. 257–268, October 2015. [Online]. Available: http://dx.doi.org/10.1007/s10817-015-9331-7

[25] G. Nelson and D. C. Oppen, "Fast decision procedures based on congruence closure," *J. ACM*, vol. 27, pp. 356–364, April 1980. [Online]. Available: http://doi.acm.org/10.1145/322186.322198

[26] A. Grabowski, A. Korniłowicz, and C. Schwarzweller, "Equality in computer proof-assistants," in *Proceedings of the 2015 Federated Conference on Computer Science and Information Systems*, ser. Annals of Computer Science and Information Systems, M. Ganzha, L. A. Maciaszek, and M. Paprzycki, Eds., vol. 5. IEEE, 2015, pp. 45–54. [Online]. Available: http://dx.doi.org/10.15439/2015F229

[27] A. Naumowicz and C. Byliński, "Improving Mizar texts with properties and requirements," in *Mathematical Knowledge Management, Third International Conference, MKM 2004 Proceedings*, ser. MKM'04, Lecture Notes in Computer Science, A. Asperti, G. Bancerek, and

A. Trybulec, Eds., vol. 3119, 2004, pp. 290–301. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-27818-4_21

[28] A. Korniłowicz, "Enhancement of Mizar texts with transitivity property of predicates," in *Intelligent Computer Mathematics – 9th International Conference, CICM 2016, Bialystok, Poland, July 25–29, 2016, Proceedings*, ser. Lecture Notes in Computer Science, M. Kohlhase, M. Johansson, B. R. Miller, L. de Moura, and F. W. Tompa, Eds., vol. 9791. Springer, 2016, pp. 157–162. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-42547-4_12

[29] A. Trybulec, "Tarski Grothendieck set theory," *Formalized Mathematics*, vol. 1, no. **1**, pp. 9–11, 1990. [Online]. Available: http://fm.mizar.org/1990-1/pdf1-1/tarski.pdf

[30] A. Grabowski, "Mechanizing complemented lattices within Mizar type system," *Journal of Automated Reasoning*, vol. 55, no. 3, pp. 211–221, October 2015. [Online]. Available: http://dx.doi.org/10.1007/s10817-015-9333-5

[31] A. Grabowski, A. Korniłowicz, and C. Schwarzweller, "On algebraic hierarchies in mathematical repository of Mizar," in *Proceedings of the 2016 Federated Conference on Computer Science and Information Systems*, ser. Annals of Computer Science and Information Systems, M. Ganzha, L. A. Maciaszek, and M. Paprzycki, Eds., vol. 8. IEEE, 2016, pp. 363–371. [Online]. Available: http://dx.doi.org/10.15439/2016F520

[32] A. Grabowski and M. Jastrzębska, "Rough set theory from a math-assistant perspective," in *Rough Sets and Intelligent Systems Paradigms, International Conference, RSEISP 2007, Warsaw, Poland, June 28–30, 2007, Proceedings*, ser. Lecture Notes in Computer Science, M. Kryszkiewicz, J. F. Peters, H. Rybinski, and A. Skowron, Eds., vol. 4585. Springer, 2007, pp. 152–161. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-73451-2_17

[33] K. Pąk, "Improving legibility of natural deduction proofs is not trivial," *Logical Methods in Computer Science*, vol. 10, no. 3, pp. 1–30, 2014. [Online]. Available: http://dx.doi.org/10.2168/LMCS-10(3:23)2014

[34] ——, "Automated improving of proof legibility in the Mizar system," in *Intelligent Computer Mathematics – International Conference, CICM 2014, Coimbra, Portugal, July 7–11, 2014. Proceedings*, ser. Lecture Notes in Computer Science, S. M. Watt, J. H. Davenport, A. P. Sexton, P. Sojka, and J. Urban, Eds., vol. 8543. Springer, 2014, pp. 373–387. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-08434-3_27

[35] ——, "Improving legibility of formal proofs based on the close reference principle is NP-hard," *Journal of Automated Reasoning*, vol. 55, no. 3, pp. 295–306, October 2015. [Online]. Available: http://dx.doi.org/10.1007/s10817-015-9337-1

[36] A. Grabowski and C. Schwarzweller, "On duplication in mathematical repositories," in *Intelligent Computer Mathematics, 10th International Conference, AISC 2010, 17th Symposium, Calculemus 2010, and 9th International Conference, MKM 2010, Paris, France, July 5–10, 2010. Proceedings*, 2010, pp. 300–314. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-14128-7_26

[37] N. S. Nikitchenko, "A composition nominative approach to program semantics," IT-TR 1998-020, Technical University of Denmark, Tech. Rep., 1998.

[38] M. Nikitchenko and S. Shkilniak, *Mathematical logic and theory of algorithms*. Publishing house of Taras Shevchenko National University of Kyiv, Ukraine (in Ukrainian), 2008.

[39] I. Ivanov, M. Nikitchenko, A. Kryvolap, and A. Korniłowicz, "Simple named-complex valued nominative data – definition and basic operations," *Formalized Mathematics*, vol. 25, no. **3**, 2017. [Online]. Available: http://dx.doi.org/10.1515/forma-2017-0020

[40] A. Grabowski, A. Korniłowicz, and A. Naumowicz, "Mizar in a nutshell," *Journal of Formalized Reasoning, Special Issue: User Tutorials I*, vol. 3, no. 2, pp. 153–245, December 2010.

[41] C. Byliński, "Partial functions," *Formalized Mathematics*, vol. 1, no. **2**, pp. 357–367, 1990. [Online]. Available: http://fm.mizar.org/1990-1/pdf1-2/partfun1.pdf

[42] G. Bancerek and K. Hryniewiecki, "Segments of natural numbers and finite sequences," *Formalized Mathematics*, vol. 1, no. **1**, pp. 107–114, 1990. [Online]. Available: http://fm.mizar.org/1990-1/pdf1-1/finseq_1.pdf

[43] C. Byliński, "Functions and their basic properties," *Formalized Mathematics*, vol. 1, no. **1**, pp. 55–65, 1990. [Online]. Available: http://fm.mizar.org/1990-1/pdf1-1/funct_1.pdf

[44] E. Woronowicz, "Relations and their basic properties," *Formalized Mathematics*, vol. 1, no. **1**, pp. 73–83, 1990. [Online]. Available: http://fm.mizar.org/1990-1/pdf1-1/relat_1.pdf

[45] G. Bancerek, "König's theorem," *Formalized Mathematics*, vol. 1, no. **3**, pp. 589–593, 1990. [Online]. Available: http://fm.mizar.org/1990-1/pdf1-3/card_3.pdf

[46] A. Trybulec, "Binary operations applied to functions," *Formalized Mathematics*, vol. 1, no. **2**, pp. 329–334, 1990. [Online]. Available: http://fm.mizar.org/1990-1/pdf1-2/funcop_1.pdf

[47] C. Byliński, "The modification of a function by a function and the iteration of the composition of a function," *Formalized Mathematics*, vol. 1, no. **3**, pp. 521–527, 1990. [Online]. Available: http://fm.mizar.org/1990-1/pdf1-3/funct_4.pdf

[48] ——, "Some basic properties of sets," *Formalized Mathematics*, vol. 1, no. **1**, pp. 47–53, 1990. [Online]. Available: http://fm.mizar.org/1990-1/pdf1-1/zfmisc_1.pdf

[49] S. M. Watt, J. H. Davenport, A. P. Sexton, P. Sojka, and J. Urban, Eds., *Intelligent Computer Mathematics – International Conference, CICM 2014, Coimbra, Portugal, July 7–11, 2014. Proceedings*, ser. Lecture Notes in Computer Science, vol. 8543. Springer, 2014. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-08434-3