CEUR-WS.org/Vol-2393/paper_317.pdf

Extending the SMT-Lib Standard with Theory of Nominative Data

Liudmyla Omelchuk [1][0000-0002-2287-1304], Olena Shyshatska [1][0000-0001-8791-8989]

[1] Taras Shevchenko National University of Kyiv, Akademika Hlushkova Ave, 4d,

Kyiv, 03680 Ukraine

l.omelchuk@knu.ua, shyshatska@knu.ua

Abstract. We describe the theory of nominative data, formulate the basic principles of the composition-nominative approach, and define the class of nominative data and functions. By using nominative data, we can increase the level of adequacy of representation data structures, functions, and compositions that are used in programming languages. Thus, in terms of composition-nominative approach, we can build systems of verification of programs based on a unified conceptual basis. Computer-aided verification of computer programs often uses SMT (satisfiability modulo theories) solvers. A common technique is to translate preconditions, postconditions, and assertions into SMT formulas in order to determine if required properties can hold. The SMT-LIB Standard was created for forming a common standard and library for solving SMT problems. Now, it is one of the most used libraries for SMT systems. Formulas in SMT-LIB format are accepted by the great majority of current SMT solvers. The theory of nominative data is of interest for software modelling and verification, but currently lacks support in the SMT-LIB format. In the article, we propose the declaration for the theory of nominative data for the SMT-LIB Standard 2.6. The goal is the development of SMT solvers with nominative data support.

Keywords: SMT solver, partial logic, nominative data, composition programming.

1 Introduction

Composition programming studies the systems at different levels of abstraction – abstract, Boolean, and nominative (attribute) levels. Systems of the last level, based on the composition-nominative methods [1], are rather expressible for a quite adequate representation of the models of data structures and programs. Thus, the composition-nominative approach provides a unified methodological basis to formalize the concept of program specification. By using nominative data, we can increase the level of adequacy of representation data structures, functions, and compositions that are used in programming languages. The axiomatic theory of nominative data [2] is developed in the spirit of the theory of admissible sets (S. Kripke, R. Platek, J. Barwise, Yu.L. Yershov). This theory has a number of advantages with respect to the adequacy of programming: on the one hand, it is strong enough to generate computable functions over different data structures, on the other hand, it is not so restrictive as different versions of constructive logic, but it is not excessively powerful and does not allow, for example, the use of axiom of constructing the set of all subsets (compared with theory of sets by Zermelo-Frankel). Moreover, this theory uses basic data corresponding to the methods of constructing data in programming. In terms of composition-nominative approach, while using nominative data one can increase the adequacy of representation data structures, functions, and compositions used in programming languages and build the systems of program specifications based on the single conceptual framework. Basic data types of programming languages were specified in [2], in addition, the functions over nominative data were specified in [1-4].

It is therefore natural to expect program analysis and verification tools to be able to reason about programs, by means of deciding the validity of formulas containing variables of such types. Application of such tools requires a standard exchange format for these types of formulas.

Dafny [5] is one of the formal verification languages. It is a hybrid language, with functional and object-oriented features, which can automatically check programs against specifications. Behind the scenes, Dafny converts programs for its users into the mathematical expressions of Hoare logic with the aid of an intermediate verification language called Boogie, and then it sends the code to an automatic proving program called Z3 [6]. Z3 input format is an extension of the one defined by the SMT-LIB 2 standard [7, 8]. This conversion process benefits Dafny users in eliminating the need to write out long proofs in confusing notation while still being able to verify their programs. As a result, Dafny requires that programmers use a strict form of syntax in order to properly convert their code into mathematical expressions.

The standardization of formats in logic has played a major role in accelerating research in the past. Examples for successful standardization efforts are the DIMACS format for Boolean formulas in conjunctive normal form (CNF), and the SMT-LIB format [6] dedicated to various first-order theories that are used in verification.

SMT-LIB was created in 2003 with the expectation that the availability of common standards and a library of benchmarks would greatly facilitate the evaluation and the comparison of SMT (satisfiability modulo theories) systems. Now, SMT-LIB contains more than 100,000 benchmarks and continues to grow. Formulas in SMT-LIB format are accepted by the great majority of current SMT solvers.

In computer science and mathematical logic, the satisfiability modulo theories problem is a decision problem for logical formulas with respect to combinations of background theories expressed in classical first-order logic with equality. Examples of theories typically used in computer science are the theory of real numbers, the theory of integers, and the theories of various data structures such as lists, arrays, bit vectors and so on. SMT can be thought of as a form of the constraint satisfaction problem and thus a certain formalized approach to constraint programming.

The following systems (listed alphabetically) were under active development in 2018: Alt-Ergo, AProVE, Boolector, CVC4, MathSAT 5, OpenSMT 2, raSAT, SMTInterpol, SMT-RAT, STP, veriT, Yices 2, Z3 [6-8].

A new sublogic, or simply logic, is defined in the SMT-LIB language by a logic declaration. Logic declarations have a similar format to theory declarations. Attributes with the following predefined keywords are predefined attributes, with prescribed usage and semantics in logic declarations [7]:

:theories :language :extensions :notes :values .

Additionally a logic declaration can contain any number of user-defined attributes.

```
<logic_attribute>    := :theories ( <symbol> + )
        |   :language <string>
        |   :extensions <string>
        |   :values <string>
        |   :notes <string>
        |   <attribute>
<logic>     ::=    ( logic <symbol> <logic_attribute> + )
```

SMT-LIB logics refer to one or more theories:

– Functional arrays with extensionality (ArraysEx),
– Bit vectors with arbitrary size (FixedSizeBitVectors),
– Core theory, defining the basic Boolean operators (Core),
– Floating point numbers (FloatingPoint),
– Integer numbers (Ints),
– Real numbers (Reals),
– Real and integer numbers (Reals_Ints) [6-8].

We propose to add a theory of nominative data to SMT-LIB, serving as a standard format for formulas that include operations on nominative data.

2 Composition-Nominative Approach

One of the approaches to software specification is composition programming [1-4]. Composition programming studies the systems at different levels of abstraction – abstract, Boolean and nominative (attribute) levels. Systems of the last level based on the composition-nominative methods [1] are rather expressive for adequate representation of the models of data structures and programs.

Thus, the composition-nominative approach provides a single methodological basis to formalize the concept of program specification, bringing their features and their further specification to programming languages of the lower level. This approach is based on the following principles [1, 4]:

Development principle (from abstract to concrete): program notions should be introduced as a process of their development that starts from abstract understanding, capturing essential program properties, and proceeds to more concrete considerations.

The principle of priority of semantics over syntax: program semantic and syntactic aspects should be first studied separately, then in their integrity in which semantic aspects prevail over syntactic ones.

Compositionality principle: programs can be constructed from simpler programs (functions) with the help of special operations, called compositions, which form a kernel of program semantics structures.

Nominativity principle: nominative (naming) relations are basic ones in constructing data and programs.

Here we have formulated only principles relevant to the topic of the article. A richer system of principles is developed in [1-4].

3 Nominative Data

Class ND of nominative data is constructed by the following recursive definition based on some sets of names of V and values of W: $ND = W \cup \left(V \overset{m}{\rightarrow} ND\right)$, where $V \overset{m}{\rightarrow} ND$ is the class of partial multi-valued (non-deterministic) functions.

For nominative data representation we use the form $d = [v_i \mapsto a_i | \ i \in I]$, where I is some set of indices. Nominative membership relation is denoted by $\in_n$. Thus, $v_i \mapsto a_i \in_n d$ means that the value of v_i in d is defined and is equal to a_i; this can be written in another form as $d(v_i) \downarrow = a_i$. The class $ND \backslash W$ is called the class of proper nominative data, or hierarchical nominative data; data from the class $V \overset{m}{\rightarrow} ND$ will be called flat nominative data, or nominative sets

Main functions over the nominative data are the following functions: naming $\Rightarrow v_D$ and denaming of $v \Rightarrow_D$ with a parameter $v \in V$, and binary operations and predicates, such as: union $\cup_D$, subtraction $\backslash_D$, equality $(=_W)_D$ on W. The function of construction of the empty nominative data $\overline{[\ \ ]}_D$, predicate of membership on W: $\in W_D$ are also defined. Operation of renaming r_x^v for the nominative data $([a_1 \mapsto b_1, \dots, v \mapsto b_i, \dots])$ yields $[a_1 \mapsto b_1, \dots, x \mapsto b_i, \dots]$. The main compositions of functions over nominative data are binary compositions: multiplication $\circ_D$, iteration $*_D$, merging Θ_D and branching ternary composition $\Diamond_D$. It is shown that the composition of multiplication corresponds to the consecutive application of functions, composition of branching – to the conditional operator if-then-else of programming languages, composition of iteration $*_D$ – to operator until-do, and composition of merging Θ_D connecs nominative data resulting from function-arguments.

The special kind of computability – nominative computability – is introduced for consideration and studied in [3]. Nominative functions are the functions over the nominative data obtained by closing of functions

$$\left\{\Rightarrow 0, \Rightarrow 1, \overline{[\ \ ]}_D, \trianglerighteq_D, \cup_D, (=_W)_D, as_D, cn_D, \in W_D\right\}$$

under compositions $\{\circ_D, \Diamond_D, *_D, \Theta_D\}$.

It is demonstrated [3] that an arbitrary partial recursive function can be represented by nominative computable functions over the set of natural numbers by modelling in the class of nominative data. In addition, it is shown in [3] that each nominative function can be represented by some binary Σ- predicate $P(x, y)$, i.e. $f(x) = y$ if and only if $P(x, y)$ [2, 3]. For this purpose, the presentation of all functions specified in the definition of nominative computability are built, as well as all the functions obtained by using the compositions.

Axiomatic theory of nominative data [2] is developed in the spirit of the theory of admissible sets (S. Kripke, R. Platek, J. Barwise, Yu.L. Yershov). This theory has a number of advantages with respect to the adequacy of the programming: on the one hand, it is quite powerful to generate computable functions over the different data structures, on the other hand, it is not so restrictive as different versions of constructive logic, but it is not excessively powerful and does not allow, for example, the use of axiom of constructing the set of all subsets (compared with theory of sets by Zermelo-Frankel). Moreover, this theory uses the basic data (elements) corresponding to the methods of constructing data in programming. The unary predicate U is used, true on the elements of the basic set W; the structure $\langle A, \in_n, =, U\rangle$ is considered. The theory of nominative data is constructed as the axiomatic theory of the first-order logic with equality and ternary nominative membership relation (predicate) of the that written in the infix form $x \mapsto \mathrm{y} \in_n a$ (or $(x, \mathrm{y}) \in_n a$).

The class of Δ_0 - formulas is the smallest class Y, containing the basic formulas and closed under the following rules:

1) if $\varphi \in Y$, then also $\neg\varphi \in Y$,

2) if, φ, $\psi \in Y$, then $\varphi \wedge \psi \in Y$ and $\varphi \vee \psi \in Y$,

3) if $\varphi \in Y$, then $\forall x \mapsto y \in_n a\ \varphi$, $\exists x \rightarrow y \in_n a\ \varphi \in \mathrm{Y}$ for all variables x, y, a.

Class of Σ-formulas is the smallest class Z, containing Δ_0-formulas and closed in relation to the conditions 2) and 3) determining the class of Δ_0-formulas and further conditions of existential quantification: if $\varphi \in Z$, then $\exists u\varphi \in Z$.

4 The Domain of Nominative Data

The theory of nominative data is of interest for software modelling and verification, but currently lacks support in the SMT-LIB format. Therefore, we propose the theory of nominative data for the SMT-LIB Standard 2.6 [7, 8].

For convenience, the definition of nominative data in the SMT-LIB Standard is presented via the concepts of nominative pair ($x \mapsto \mathrm{y}$) and nominative set ($[a_1 \mapsto b_1, \ldots, x \mapsto b_i, \ldots]$). Nominative pair of two typed elements are the most basic collection datatype that we propose for an SMT-LIB theory. Semantically, assuming that the type t_1 denotes the non-empty domain (name) a and the type t_2 denotes the domain (value) b, the type $NdPair\ a\ to\ b$ denotes the domain $a \mapsto b$. Table 1 contains all proposed operations on nominative pairs in mathematical and in concrete SMT-LIB notation.

This table gives a signature of the proposed SMT-LIB theory of nominative pairs. In the first column of Table 1, we specify a mathematical notation for the functions used with nominative pairs. In the mathematical notation, we use the following notation: a_i or v are names in the nominative pairs, b_i is the value in the nominative pair, and d is the nominative pair. In the second column of Table 1, we specify a proposed SMT-LIB notation for each operation. In the third column of Table 1, we specify a signature for each operation. In the signature definition, we use the following notation: α is a set of names in nominative pairs, and β is a set of values in nominative pairs.

Table 1. Signature of the SMT-LIB the theory of nominative data for nominative pair

Math. notation	**Proposed SMT-LIB notation**	**Prop. SMT-LIB typing**
$a_1 \mapsto b_1 =_v a_2 \mapsto b_2 =$ $= \begin{cases} T, if\ a_1 = a_2 \\ F, \text{otherwise} \end{cases}$ equality by name =	(ndpair <term> <term>) (pairnameequal <term> <term>)	$(NdPair\ \alpha\ to\ \beta,$ $NdPair\ \alpha\ to\ \beta)$
$a_1 \mapsto b_1 \equiv a_2 \mapsto b_2 =$ $= \begin{cases} T, if\ b_1 = b_2 \\ F, \text{otherwise} \end{cases}$ equality by value $\equiv$	(pairvalueequal <term> <term>)	$(NdPair\ \alpha\ to\ \beta,$ $NdPair\ \alpha\ to\ \beta)$
$a_1 \mapsto b_1 = a_2 \mapsto b_2 =$ $= \begin{cases} T, if\ (a_1 = a_2) and (b_1 = b_2 \\ F, \text{otherwise} \end{cases}$ equality=	(ndpairequal <term> <term>)	$(NdPair\ \alpha\ to\ \beta,$ $NdPair\ \alpha\ to\ \beta)$
$\Rightarrow v_D(d) = (v, d)$ denaming	(naming <term> <term> <term>)	$(\alpha,\ \beta, NdPair\ \alpha\ to\ \beta)$
$r_a^v([v \to b_1]) = [a \to b_1]$ Renaming	<pairrenaming <term> <term> <term>)	$(\alpha,\ \beta, NdPair\ \alpha\ to\ \beta,$ $NdPair\ \alpha\ to\ \beta)$

Note, that the type of an n-ary predicate SMT-LIB is in specified by an $n-$tuple $(t_1 \dots t_n)$, while the type of an n -ary function with result type t_0 is given by an $(n+1)$ - tuple $(t_1 \dots t_n\ t_o)$.

A Declaration for a nominative pair for theory of nominative data for the SMT-LIB Standard is given in Fig. 1.

```
  (theory NominativePair

   :smt-lib-version 2.6
   :written_by "Liudmyla Omelchuk"
   :date "15/01/2019"

   :sorts ((Int 1) (NdPair 2)))

   :funs (
       (par (X Y) (ndpair X Y (NdPair X Y)))
       (par (X Y) (naming X Y (NdPair X Y)))
       (par (X Y) (pairrenaming  (NdPair X Y) X (NdPair X Y)))
       (par (X Y) (pairnameequal (NdPair X Y) (NdPair X Y) Bool))
       (par (X Y) (pairvalueequal (NdPair X Y) (NdPair X Y) Bool))
       (par (X Y) (pairequal (NdPair X Y) (NdPair X Y) Bool))))

   :definition

  "Let Q be a set of sort symbols including NdPair and Bool, and let S
be the set of all (ground) sort terms over Q. For any s in S and any
```

```
function symbol f, let [[s]] and [[f]] respectively denote the inter-
pretation of s and f in some given structure.

  For any S like the above, NominativePair(S) is the theory consisting
of all structures satisfying the following restrictions for all s, s'
in S:
    - [[(NdPair s s')]] is the pair from [[s]] to [[s']].

    - For all name n and data d
        [[naming]](n, d) = [[(NdPair n d)]].

    - For all nominative pair p from [[n]] to [[v]] and all name x
        [[paitrenaming]](p, x) = [[(NdPair x v)]].

     - For all nominative pairs a from [[n1]] to [[v1]] and nominative
pair b from [[n2]] to [[v2]],
                                   true         if n1 = n2
        [[pairnameequal]](a, b) =     false        otherwise.

     - For all nominative pairs a from [[n1]] to [[v1]] and nominative
pair b from [[n2]] to [[v2]],
                                   true         if v1 = v2
        [[pairvalueequal]](a, b) =  false        otherwise.

     - For all nominative pairs a from [[n1]] to [[v1]] and nominative
pair b from [[n2]] to [[v2]],
                             true      if (v1 = v2) and (n1 = n2)
        [[pairequal]](a, b) = false     otherwise.
  "
   :note

   "For any given S, NominativePair(S) has several models which differ
not only on how they interpret the base sorts of S other than Int and
Bool, but also on how they interpret the 'apply' function symbol when
the value of a map is queried at a point outside of its domain.
    ")
```

Fig. 1. A declaration for a theory of nominative pair

Nominative set of nominative pairs is the collection datatype that we propose for SMT-LIB theory. Semantically, the nominative set types denote sets NdSet(A, B) of finite partial functions. The Table 2 contains all proposed operations on nominative sets in mathematical and in concrete SMT-LIB notation. This table gives a signature of the proposed SMT-LIB theory of nominative set.

In the first column of Table 2, we specify a mathematical notation for the functions used with nominative data. In the mathematical notation, we use the following notation: a_i or v are names in the nominative pairs, b_i is the value in the nominative pair, and d is the nominative set. In the second column of Table 2, we specify a proposed SMT-LIB notation for each operation. In the third column of Table 2, we specify a signature for each operation. In the signature definition, we use the following notation: α is a set of names in nominative pairs, β is a set of values in nominative pairs, and γ is a set of nominative pairs.

Table 2. Signature of the SMT-LIB theory of nominative data for nominative set

Math. notation	**Proposed SMT-LIB notation**	**Proposed SMT-LIB typing**
$[a_1 \mapsto b_1, \dots]$	(NdSet <term>*)	$((\gamma)^* NdSet\ \alpha\ to$
$a \to b \in\ _n d$	(ndin <term> <term>)	$(\gamma, NdSet\ \alpha\ to\ \beta)$
$\Rightarrow v_D(d) = [v \to d]$	(naming <term> <term>)	$(\alpha\ \beta\ NdSet\ \alpha\ to$
$f \nabla g = g \cup \{(v,w) \vert$ $(v,w) \in_n f\ \&\ \neg\exists w'^{(v,w') \in_n} g\}$	(ndoverlay <term> <term><term>)	$(NdSet\ \alpha\ to\ \beta$ $NdSet\ \alpha\ to\ \beta$ $NdSet\ \alpha\ to\ \beta\ )$
$\cup_D\ (d_1, \dots, d_n) = [v \mapsto d \vert$ $v \mapsto d \in\ _n d_1 \dots . \vee v \to d \in\ _n d_n]$	(ndunion <term>$^+$)	$(NdSet\ \alpha\ to\ \beta)^+$ $NdSet\ \alpha\ to\ \beta$
$^\backslash D(d_1, d_2) = [v \mapsto d \vert$ $v \mapsto d \in\ _n d_1 \wedge v \to d \notin\ _n d_2]$	(ndsetminus <term><term>)	$(NdSet\ \alpha\ to\ \beta$ $NdSet\ \alpha\ to\ \beta$ $NdSet\ \alpha\ to\ \beta\ )$
$\overline{[\ \ ]}_D(d) = [\ ]$	(emptyNdSet <term>)	$(NdSet\ \alpha\ to\ \beta$ $NdSet\ \alpha\ to\ \beta\ )$
$=_D\ (d_1, d_2) =$ $= \begin{cases} T, & if\ d_1, d_2 \in W, d_1 = d_2 \\ F, & if\ d_1, d_2 \in W, d_1 \neq d_2 \\ \uparrow, & \text{otherwise} \end{cases}$	(ndequal <term><term>)	$(NdSet\ \alpha\ to\ \beta$ $NdSet\ \alpha\ to\ \beta\ )$
$r_x^v([a_1 \mapsto b_1 \dots, v \mapsto b_i, ..]) =$ $= [a_1 \mapsto b_1 \dots, x \mapsto b_i]$	(ndrenaming <term><term><term> <term>)	$(\ \alpha\ \alpha$ $NdSet\ \alpha\ to\ \beta$ $NdSet\ \alpha\ to\ \beta)$
$\subseteq_D\ (d_1, d_2)$	(ndsubset <term><term>)	$(NdSet\ \alpha\ to\ \beta$ $NdSet\ \alpha\ to\ \beta)$

The theory of nominative data defines a parameterized sort and functions to read and write elements of NdSet.

The new sort symbol NdSet takes two sort parameters: the first is the sort of the name, the second is the sort of the value of the nominative data elements.

Two values of the same NdSet sort are equal if the Set elements are equal for every value of the name sort.

A declaration for a nominative set of the theory of nominative data for the SMT-LIB Standard is presented in Fig. 2.

```
    (theory NominativeSet

    :smt-lib-version 2.6
    :written-by "Liudmyla Omelchuk"
    :date "2019-02-25"

    :sorts ((Int 1) (NDSet 2))

    :funs (((par (X) (emptyNdSet (NdSet X)))
   (par X) (naming X X (NdSet X) (NdSet X) :right_assoc)
   (par X) (denaming X (NdSet X) (NdSet X) :right_assoc)
   (par (X) (ndin (NdPair(X Y)) (NdSet Z) Bool))
   (par (X) (ndsubset (NdSet X) (NdSet X) Bool :chainable))
   (par (X) (ndunion (NdSet X) (NdSet X) (NdSet X) :right_assoc))
   (par (X) (ndoverlay (NdSet X) (NdSet X) (NdSet X) :right_assoc))
   (par (X) (ndsetminus (NdSet X) (NdSet X) (NdSet X) :right_assoc))
   (par (X) (ndequal (NdSet X) (NdSet X) Bool :chainable))
   (par (X (renaming X X (NdSet X) (NdSet X) :right_assoc))

    :definition
    "Let Q be a set of sort symbols including Set, Int, and Bool, and
let S be the set of all (ground) sort terms over Q.  For any s in S
and any function symbol f, let [[s]] and [[f]] respectively denote the
Interpretation of s and f in some given structure.

     For any S like the above, NominativeSet(S) is the theory consist-
ing of all structures satisfying the following restrictions for all s
in S:

     - [[(Set s)]] is the set of all finite subsets of [[s]].

     - [[f]] is as expected if f is in {subset, in}.

     - [[(NdSet s s')]] is the nominative set of all finite partial
maps from [[s]] to [[s']].

     - [[emptyNdSet]] is the function from [[s]] to [[s']] undefined
everywhere.

     - For all nominative sets m and name x,
       [[naming]](x, m) = [[(NdPair x m)]] is the nominative set of all
finite partial maps from [[x]] to [[m]].

     - For all nominative set m and name x,
                                b          if [[(NdPair x b)]] in m
        [[denaming]](x, m) =
                                undefined   otherwise.

     - For all nominative set m, name n and nominative set v,
                                      true if [[(NdPair x b)]] in m
        [[ndin]]([(NdPair n v)], m) =
                                      false   otherwise.

     - For all n > 0, nominative set a and nominative set b of [[b1]],
..., [[bn]],
                           true  if bi ndin a for all i = 1, ..., n
     [[ndsubset]](b, a) =
                           false otherwise.

     - For all n, m > 0, nominative set a of [[a1]], ..., [[an]] and
nominative set b of [[b1]], ..., [[bm]],

        [[ndunion]](b, a) = [[a1],…[an],[b1],…,[bm]].
    - For all n, m > 0, nominative set a of [[(NdPair a1 b1)],…[(
NdPair an bn)]] and nominative set b,
         [[ndoverlay]](a, b) = [[ndunion](b,
        { x indn [[(NdPair a1 b1)],…[( NdPair an bn)]] |
        there in not any d (NdPair ai d) ndin b })].

    - For all k > 0, nominative set a of [[a1]], ..., [[an]] and nomi-
native set b of [[b1]], ..., [[bm]],
        [[ndsetminus]](a, b) = {x ndin [[s1],…[sk]] |
        for all i <=k si idin a and not (si indin b)}.

    - For all nominative set a of [[(NdPair a1 b1)],…[( NdPair an
bn)]] and name x,

       [[renaming]](x, y, a) ={x ndin [[(NdPair a1 b1)],…
       [( NdPair am bm)]] | for all i<=m
       ([(NdPair ai bi)] dnin [[(NdPair a1 b1)],…
       [( NdPair an bn)]]) and (not ai = x)  } ndunion
       { [[(NdPair y bj)] | [[(NdPair x bj)] ndin
       [[(NdPair a1 b1)],…[( NdPair an bn)]]}.

    - For all k > 0, nominative set a [[a1]], ..., [[an]] and nomina-
tive set b,
                                true    if for all i <=n ai idin b
        [[ndsubset]](a, b) =
                                false   otherwise.

    - For all nominative sets a, b
        [[ndequal]](a, b) = (a ndsubset b) and (b ndsubset a).

    - [[n]] is as expected if n is a numeral.

    - [[(NdSet s)]] is the set of all finite subsets of [[s]].

    - [[f]] is as expected if f is in
      { emptyNdSet, naming, denaming, ndin, ndsubsetm ndunion,
       ndoverlay, ndsetminus, ndsubset, ndequal, renaming}.
  ")
```

Fig. 2. A declaration for a theory of nominative set

Using proposed by the authors the SMT-LIB theory of nominative data, we can adequately determine the structure of programming data and operations on them.

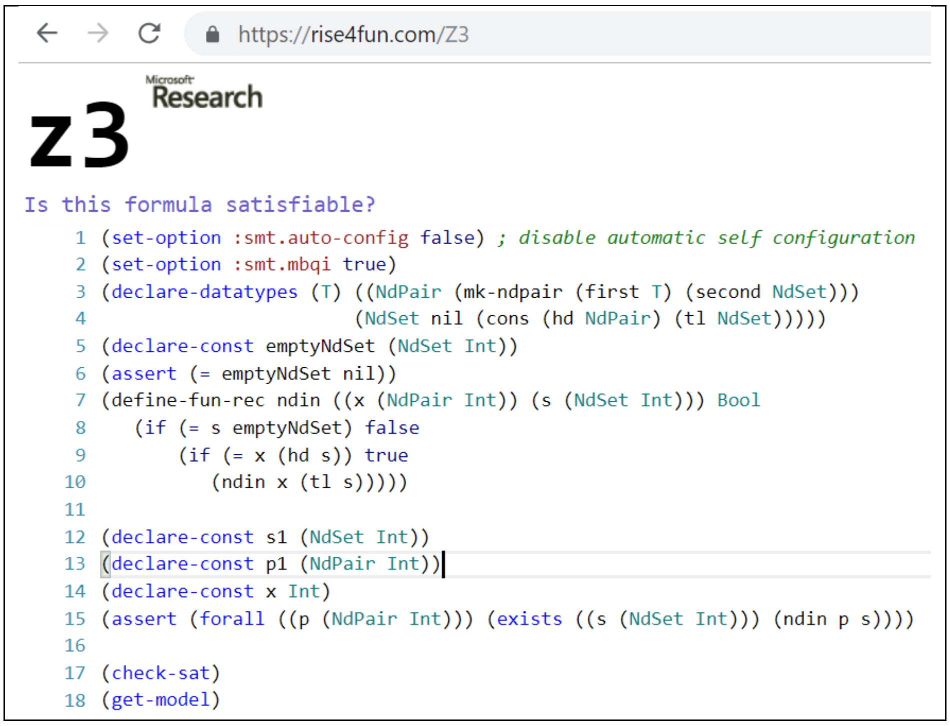
For example, the following statement in terms of the theory of nominative data:

$$\forall a \exists s (a \in_n s)$$

can be specified in the form of the following SMT- LIB notation:

```
(forall ((a (NdPair Int))) (exists ((s (NdSet Int))) (ndin a s)))
```

Fig. 3-4 give an example of the use of nominative data structures in theorem prover Z3, which is one of the SMT solvers.

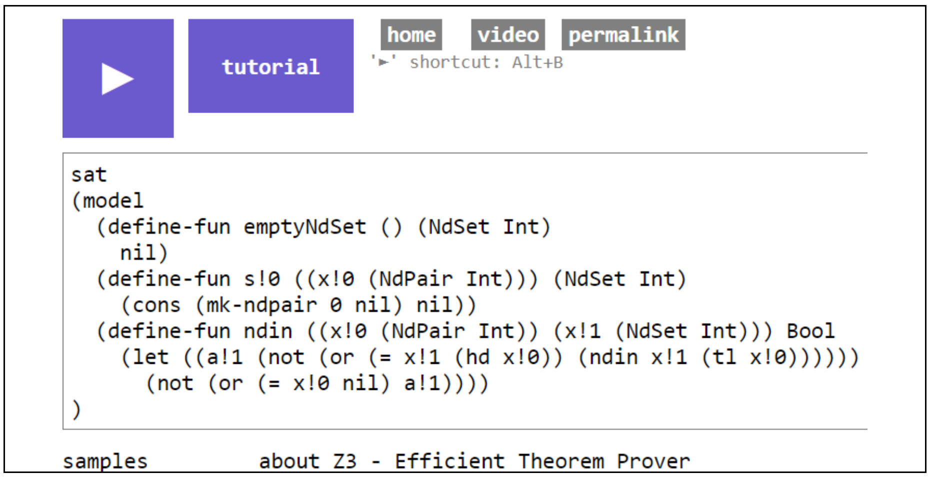


```
(set-option :smt.auto-config false) ; disable automatic self configuration
(set-option :smt.mbqi true)
(declare-datatypes (T) ((NdPair (mk-ndpair (first T) (second NdSet)))
                        (NdSet nil (cons (hd NdPair) (tl NdSet)))))
(declare-const emptyNdSet (NdSet Int))
(assert (= emptyNdSet nil))
(define-fun-rec ndin ((x (NdPair Int)) (s (NdSet Int))) Bool
   (if (= s emptyNdSet) false
       (if (= x (hd s)) true
          (ndin x (tl s)))))

(declare-const s1 (NdSet Int))
(declare-const p1 (NdPair Int))
(declare-const x Int)
(assert (forall ((p (NdPair Int))) (exists ((s (NdSet Int))) (ndin p s))))

(check-sat)
(get-model)
```

Fig. 3. An example of the use of nominative data structures in Z3



```
sat
(model
  (define-fun emptyNdSet () (NdSet Int)
    nil)
  (define-fun s!0 ((x!0 (NdPair Int))) (NdSet Int)
    (cons (mk-ndpair 0 nil) nil))
  (define-fun ndin ((x!0 (NdPair Int)) (x!1 (NdSet Int))) Bool
    (let ((a!1 (not (or (= x!1 (hd x!0)) (ndin x!1 (tl x!0))))))
      (not (or (= x!0 nil) a!1))))
)
```

Fig. 4. An example of a satisfiable of nominative data structures in Z3

The link to https://sites.google.com/knu.ua/nominative-data contains information that relates the description of Extended SMT-Lib Standard with Theory of Nominative Data [10].

5 Conclusion

In the article, we have proposed the extension of the SMT-LIB Standard with theory of nominative data. This theory is of interest for software modelling and verification. At the same time, the declaration for the SMT-LIB Standard for the theory of nominative data has not yet been proposed.

To construct the extension of the SMT-LIB Standard for the constructed theory of nominative data, we have described the main principles of the composition-nominative approach and the definition of the class of nominative data. Such data form a basis for adequate definition of data structures, functions, and compositions of programming languages [2-4].

We are developing SMT solvers with nominative data support. In the forthcoming articles we will demonstrate applications of the proposed theory.

References

1. Nikitchenko, N.: A Composition Nominative Approach to Program Semantics. Techn. Report IT–TR. Technical University of Denmark, Lyngby (1998).
2. Omelchuk, L.: Aksiomatychni systemy specyfikacij program nad nominatyvnymy danymy [Axiomatic Systems of Specifications of Programs over Nominative Data]. Candidate's thesis. Kyiv [in Ukrainian] (2007).
3. Nikitchenko, N., Omelchuk, L., Shkilniak S.: Formalisms for Specification of Programs over Nominative Data. Electronic computers and informatics (ECI 2006). Košice, Herl'any, Slovakia, 134-139 (2006).
4. Kryvolap, A., Nikitchenko, M., Schreiner, W.: "Extending Floyd–Hoare logic for partial pre- and postconditions", CCIS 412, Springer, Heidelberg, pp. 355-378. (2013).
5. Leino, K.R.M.: Dafny: An Automatic Program Verifier for Functional Correctness. In LPAR-16, 6355:348–370. LNCS. Springer. (2010).
6. Moura, L., Bjørner, N.: Z3: An efficient smt solver. In In Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). (2008).
7. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard: Version 2.6. Publishing house of Department of Computer Science, The University of Iowa. (2017). Homepage, http://www.SMT-LIB.org , last accessed 2019/03/11.
8. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT Modulo Theories: From an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). Journal of the ACM. T. 53, vol. 6, pp. 937—977. (2006).
9. Sutcliffe, G., Suttner, C.: The TPTP problem library—CNF release v1.2.1. J. Autom. Reasoning, 21(2). pp. 177–203. (1998).
10. Theory of Nominative Data, https://sites.google.com/knu.ua/nominative-data, last accessed 2019/04/21.