A Proposal for a Theory of Finite Sets, Lists, and Maps for the SMT-Lib Standard

Daniel Kröning Philipp Rümmer Georg Weissenbacher

Abstract

Sets, lists, and maps are elementary data structures used in most programs. Program analysis tools therefore need to decide verification conditions containing variables of such types. We propose a new theory for the SMT-Lib standard as the standard format for such formulae.

1 Motivation

Sets, lists, and maps are elementary data structures used in most programs. All modern high-level languages by now offer libraries with implementations of these data structures. Instances are the Standard Template Library (STL) for C++, and the Java Collection library. In case of low-level languages (such as plain ANSI-C), variables of type set or list are often introduced as *ghost variables* for specification purposes. It is therefore natural to expect program analysis and verification tools to be able to reason about such programs, by means of deciding the validity of formulae containing variables of such types. Componentisation of such tools requires a standard exchange format for these types of formulae.

The standardisation of formats in logic has played a major role in accelerating research in the past. Examples for successful standardisation efforts are the DIMACS format for Boolean formulae in conjunctive normal form (CNF), and the SMT-Lib format [8] dedicated to various first-order theories that are used in verification. The associated SMT-Lib repository includes more than 60000 benchmark problems to date and dozens of supporting tools. As in the case of propositional SAT, annual competitions have resulted in a significant advance in the capacity of the available solvers.

We propose to add a new theory to SMT-Lib, serving as a standard format for formulae that include operations on finite sets, lists, and maps. Our proposal is based on the Vienna Development Method (VDM) [2],[1] but aims at providing a set of core operators suitable for verification conditions arising in other modeling and verification frameworks, such as Z [9], the B method [1], or ASMs [4]. Our proposed theory is less general (more specific)

[1] `http://www.vdmportal.org`

than, e.g., the theory of algebraic datatypes or generalisations of the theory of arrays. This makes it possible to include relevant operations such as the length of lists or the range of finite maps that could not easily be expressed in the more general case. Besides, the structure of verification problems can be preserved to a larger degree in a more specific format.

VDM originates from IBM's Vienna Laboratory in the 1970s. It has grown to include a group of techniques and tools based on a formal specification language—the VDM Specification Language (VDM-SL). It has an extended form, VDM++, which supports the modelling of object-oriented and concurrent systems. Support for VDM includes commercial and academic tools for analysing models, including support for testing and proving properties of models and generating program code from validated VDM models. There is a history of industrial usage of VDM and its tools. There is also a growing body of research on the formalism that has led to notable contributions to the engineering of critical systems, compilers, concurrent systems and to logic for computer science.

2 The Type System and the Domain Universe

The current version of the SMT-Lib language [8] is based on many-sorted first-order logic. On top of this logic, several first-order *theories* (like integer arithmetic or arrays) are included in the standard, each one with specific types, operators, and semantics. Types are specified by means of a set of sort symbols $\mathcal{S}$. The language does not provide any means for type constructors, polymorphism, or composite types.[2] This is at odds with our goal to support container types such as sets and maps. As a solution, we propose to use a grammar that generates $\mathcal{S}$, which is given in Figure 1.

The proposed type system is based on the domain universe of VDM-SL (see [2],[3] Section 1.4), but is sufficiently general to be applicable for other languages. We use the following domains (which are defined in more detail in the next section):

- The basic domains $\mathbb{Z}$ (`Int`) and $\mathbb{R}$ (`Real`). These domains have the same semantics and offer the same operations as the theories already provided by the SMT-Lib [8].

  Since the Boolean domain $\mathbb{B}$ and the natural numbers can be mapped to the integers $\mathbb{Z}$, and the rationals $\mathbb{Q}$ can be mapped to $\mathbb{Z} \times \mathbb{Z}$, we omit these domains in our type system in Figure 1.

[2] It is expected that such extensions will be added to SMT-Lib in the near future, but it should be possible to adapt the following definitions without difficulties.

[3] The corresponding technical report [2] is available at `http://www.vdmportal.org/twiki/pub/Main/VDMpublications/tr-isovdmsl.pdf`.

$$\begin{array}{lll}
\langle \mathit{basic} \rangle & ::= & \texttt{Int} \mid \texttt{Real} \\
\langle \mathit{tuple} \rangle & ::= & \texttt{(}\langle \mathit{type} \rangle\texttt{*}\cdots\texttt{*}\langle \mathit{type} \rangle\texttt{)} \\
\langle \mathit{set} \rangle & ::= & \texttt{Set\_of\_}\langle \mathit{type} \rangle \\
\langle \mathit{list} \rangle & ::= & \texttt{List\_of\_}\langle \mathit{type} \rangle \\
\langle \mathit{map} \rangle & ::= & \texttt{Map\_}\langle \mathit{type} \rangle\texttt{\_to\_}\langle \mathit{type} \rangle \\
\langle \mathit{type} \rangle & ::= & \langle \mathit{basic} \rangle \mid \langle \mathit{tuple} \rangle \mid \langle \mathit{set} \rangle \mid \langle \mathit{list} \rangle \mid \langle \mathit{map} \rangle
\end{array}$$

Figure 1: The grammar generating the set of concrete sort symbols $\mathcal{S}$

- The domain $A_1 \times \cdots \times A_n$ of n-ary tuples (for $n \geq 0$), provided that $A_1, \ldots, A_n$ are domains in our type system.
- The domain $\mathbb{L}(A)$ of finite lists, provided that A is a domain.
- The domain $\mathbb{F}(A)$ of finite sets, provided that A is a domain.
- The domain $\mathbb{M}(A, B)$ of finite maps, provided that A and B are domains.

Finitely enumerable VDM-SL types such as tokens, characters, and quote types (which outside of VDM-SL are also known as *atoms* or *quarks*) can be trivially mapped to a subset of the integers, and records can be mapped to tuples. Therefore, these types are omitted.

For theorem provers and solvers that support the SMT-Lib, it might or might not be a problem that our proposed type system contains infinitely many types. In the first case, approximations of the complete system can be introduced by bounding the recursion depth of types. Even though the SMT-Lib format does not allow user-defined polymorphic functions, it is always possible to find a sufficient finite approximation that contains all types needed for a particular formula.

2.1 Undefinedness in the Proposed SMT-Lib Theories

A recurring issue when defining specification languages is the phenomenon of *undefinedness* or *partiality*, see [5] for an overview. This problem occurs whenever the semantics of "illegal" function applications (like division by zero) needs to be defined. The solution chosen in VDM-SL is to introduce a distinguished individual and truth value $\perp$ that ill-defined expressions evaluate to.

Introducing such a value $\perp$ in the SMT-Lib format is, unfortunately, out of the question: it would be inconsistent with the SMT-Lib base logic, existing SMT-Lib theories, and the semantics assumed by SMT-solvers. We therefore follow the approach known as *under-specification* [5], which means that an ill-defined expression like $\frac{1}{0}$ is defined to evaluate to an ordinary, but otherwise unspecified value. This means that the equation $\frac{1}{0} = \frac{1}{0}$ is valid (because the value of $\frac{1}{0}$ is arbitrary but fixed), and that the equations $\frac{1}{0} = 23$ and $\frac{1}{0} = \frac{2}{0}$ are satisfiable but not valid.

Translations from VDM-SL to SMT-Lib need to be defined such that the meaning of formulae is preserved. Due to the different semantics of VDM-SL and SMT-Lib w.r.t. undefinedness (even though the theories proposed in this paper strongly resemble the datatypes available in VDM-SL), it can be expected that the translation of certain expressions requires the introduction of additional case distinctions to handle undefined cases correctly (see, e.g., [3] on how to construct such case distinctions). It is, however, unlikely that such case distinctions incur a significant overhead.

3 Set Theory

We base our definition of the domain universe, the types, and the respective operations on standard set theory. The following concepts, thus, describe the intended semantics of the theories that are described in the later sections of this proposal.

We write $a \in A$ to denote that a is an element of the set A. Sets can be explicitly defined by means of enumerations $\{a_1, \ldots, a_n\}$, whereby the multiplicity of elements is irrelevant (e.g., $\{a, a\} = \{a\}$). As a special case, $\{\}$ denotes the empty set. The set-theoretic operations $\subseteq, \cap, \cup, \setminus$ are given the usual semantics. The expression $|S|$ denotes the cardinality of a set S. Finally, we use $\mathbb{F}(A)$ to define the set of finite subsets of A, and $\mathscr{P}(A)$ to define the powerset of A (i.e., the set of all subsets):

$$\mathscr{P}(A) = \{B \mid B \subseteq A\} \qquad \mathbb{F}(A) = \{B \in \mathscr{P}(A) \mid B \text{ is finite}\}$$

Cartesian products. An n-tuple $(a_1, \ldots, a_n)$ can be defined by explicitly listing its elements $a_1, \ldots, a_n$. The generalised Cartesian product of n sets $A_1, \ldots, A_n$ is then defined as the set of all n-tuples:

$$\bigtimes_{i=1}^{n} A_i = \{(a_1, \ldots, a_n) \mid a_a \in A_1, \ldots, a_n \in A_n\}$$

For an n-ary Cartesian product $\bigtimes_{i=1}^{n} A_i$, there is a family of n *projections* $\{\pi_1, \ldots, \pi_n\}$ such that for all $(a_1, \ldots, a_n) \in \bigtimes_{i=1}^{n} A_i$ and for all $k \in \{1, \ldots, n\}$ the equation $\pi_k((a_1, \ldots, a_n)) = a_k$ holds. Similarly, for all $t \in \bigtimes_{i=1}^{n} A_i$ the equation $t = (\pi_1(t), \ldots, \pi_n(t))$ holds.

Partial functions. A *binary relation* between sets A and B is an arbitrary subset of the Cartesian product $A \times B$. The sets of partial functions and of finite partial functions from A to B are defined by:

$$\begin{aligned} A \rightharpoonup B &= \{f \in \mathscr{P}(A \times B) \mid \forall (a, b_1), (a, b_2) \in f.\ b_1 = b_2\} \\ \mathbb{M}(A, B) &= \{f \in A \rightharpoonup B \mid f \text{ is finite}\} \end{aligned}$$

We use dom(f) to denote the domain of a (partial) function f, and rng(f) to denote the range:

$$\begin{aligned} \mathrm{dom}(f) &= \{a \in A \mid \exists b \in B \,.\, (a,b) \in f\} \\ \mathrm{rng}(f) &= \{b \in B \mid \exists a \in A \,.\, (a,b) \in f\} \end{aligned}$$

Finite lists. $\mathbb{L}(A)$ denotes the set of all finite lists over a set A:

$$\mathbb{L}(A) \;=\; \{f \in \mathbb{M}(\mathbb{N}, A) \mid \exists n \in \mathbb{N}.\ \mathrm{dom}(f) = \{1, \ldots, n\}\}$$

Thus, a list is a partial function mapping indices $i \in \mathbb{N}$ to elements of the set A. To define a list by means of enumeration of its elements, we use the constructor $[a_1, \ldots, a_n]$. The length of a list $l \in \mathbb{L}(A)$ is denoted by len(l).

4 The Domain of Finite Sets

Finite sets of uniformly typed elements are the most basic collection datatype that we propose as an SMT-Lib theory. Semantically, assuming that the type t denotes the non-empty domain (the set) A, the type `Set_of_`t denotes the domain $\mathbb{F}(A)$. The table below contains all proposed operations on finite sets in mathematical and in concrete SMT-Lib notation. The operations are typed so that it is impossible to construct sets with non-uniformly typed elements. The equality operation on sets is extensional, i.e., two sets are considered equal if and only if they contain the same elements (this holds for all datatypes defined in this proposal).

Math. notation	Proposed SMT-Lib notation	Prop. SMT-Lib typing[4]
$\{a_1, \ldots, a_n\}$	`(set` $\langle \mathit{term} \rangle^*$`)`	`(`α^* `Set_of_`α`)`
$a \in A$	`(in` $\langle \mathit{term} \rangle$ $\langle \mathit{term} \rangle$`)`	`(`α `Set_of_`α`)`
$A \subseteq B$	`(subset` $\langle \mathit{term} \rangle$ $\langle \mathit{term} \rangle$`)`	`(Set_of_`α `Set_of_`α`)`
$A_1 \cap \cdots \cap A_n$	`(inter` $\langle \mathit{term} \rangle^+$`)`	`(Set_of_`α^+ `Set_of_`α`)`
$A_1 \cup \cdots \cup A_n$	`(union` $\langle \mathit{term} \rangle^+$`)`	`(Set_of_`α^+ `Set_of_`α`)`
$A \setminus B$	`(setminus` $\langle \mathit{term} \rangle$ $\langle \mathit{term} \rangle$`)`	`(Set_of_`α `Set_of_`α `Set_of_`α`)`
$\mathscr{P}(A)$	`(power` $\langle \mathit{term} \rangle$`)`	`(Set_of_`α `Set_of_Set_of_`α`)`
$\lvert A \rvert$	`(card` $\langle \mathit{term} \rangle$`)`	`(Set_of_`α `Int)`

We do not include an operator for set comprehensions, because a similar effect can be achieved by introducing a fresh constant that is defined using a quantified axiom.

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

5 Tuples and Cartesian Products

Tuple types (t_1* $\cdots$ *t_n) denote the Cartesian product $\times_{i=1}^{n} A_i$ of the individual domains. The primary operations on tuples are the construction of tuples by listing n terms (of arbitrary types), and the projection of a tuple to one of its components. Because the SMT-Lib standard does not provide dependent types, we declare n projection operations `project[`i`]` for each n-ary tuple type ($i \in \{1, \ldots, n\}$). Again, equality is extensional, i.e., two tuples are equal if and only if they contain the same components.

Math. notation	Prop. SMT-Lib notation	Prop. SMT-Lib typing
$(a_1, \ldots, a_n)$	(tuple $\langle term \rangle^*$)	(α_1 ... α_n (α_1*$\cdots$*α_n))
$\pi_i(a)$	(project[i] $\langle term \rangle$)	((α_1*$\cdots$*α_n) α_i)
$\times_{i=1}^{n} A_i$	(cart $\langle term \rangle^*$)	(Set_of_α_1 ... Set_of_α_n Set_of_(α_1*$\cdots$*α_n))

Besides the basic operations, Cartesian products over n finite sets (of arbitrary types) are also supported, which introduces a connection between the theories of tuples and of finite sets.

6 The Domain of Lists

If the type t denotes the non-empty domain A, then the type List_of_t denotes the domain $\mathbb{L}(A)$ of finite lists. Although lists lend themselves to a direct implementation as an algebraic datatype, we include them explicitly in the proposed SMT-Lib extension to be able to equip them with a set of operations similarly as in VDM-SL: two functions to construct lists from a sequence of elements and from a sequence of lists, as well as various functions for accessing the elements of lists (see [2] for formal definitions of the functions). The value of the SMT-Lib expression (nth $\langle term \rangle$ $\langle term \rangle$) is unspecified if the given index does not occur in the list, as are the values of (hd $\langle term \rangle$) and (tl $\langle term \rangle$) when the functions are applied to empty lists.

Math. notation	Prop. SMT-Lib notation	Prop. SMT-Lib typing
$[a_1, \ldots, a_n]$	(list $\langle term \rangle^*$)	(α^* List_of_α)
$l(i)$	(nth $\langle term \rangle$ $\langle term \rangle$)	(List_of_α Int α)
len(l)	(len $\langle term \rangle$)	(List_of_α Int)
hd(l)	(hd $\langle term \rangle$)	(List_of_α α)
tl(l)	(tl $\langle term \rangle$)	(List_of_α List_of_α)
inds(l)	(inds $\langle term \rangle$)	(List_of_α Set_of_Int)
elems(l)	(elems $\langle term \rangle$)	(List_of_α Set_of_α)
append($l_1, \ldots, l_n$)	(append $\langle term \rangle^*$)	(List_of_α^* List_of_α)

Equality is extensional, i.e., two lists are equal if and only if they have the same length and contain the same elements.

7 Finite Mappings

The theory of finite maps has a similar vocabulary and semantics as McCarthy's arrays, with the difference that finite maps are only defined over finite domains (and therefore only have a finite range as well). Semantically, the map types denote sets $\mathbb{M}(A, B)$ of finite partial functions.

The operations on such maps are defined in the table below. Like sets, maps can explicitly be constructed by enumerations $\{a_1 \mapsto b_1, \ldots, a_n \mapsto b_n\}$. The value of the corresponding SMT-Lib operation (map $\langle \mathit{term} \rangle^*$) is unspecified if $a_i = a_j$ for some $i \neq j$. Likewise, the value of a map access expression (apply $\langle \mathit{term} \rangle$ $\langle \mathit{term} \rangle$) is unspecified if the point to be queried is not an element of the map domain. Overwriting a map at a particular point is defined for arbitrary maps, in contrast, and can also be used to extend the domain of the map. The remaining functions return the domain and the range of maps, and can be used to remove elements from the domain of a map (see [2] for formal definitions of the functions).

Equality is extensional, i.e., two finite maps are equal if and only if they are defined over the same domain and return the same values on all points of the domain.

Math. notation	Prop. SMT-Lib notation	Prop. SMT-Lib typing
$\{a_1 \mapsto b_1, \ldots\}$	(map $\langle \mathit{term} \rangle^*$)	((α β)* Map_α_to_β)
$f(i)$	(apply $\langle \mathit{term} \rangle$ $\langle \mathit{term} \rangle$)	(Map_α_to_β α β)
f_a^b	(overwrite $\langle \mathit{term} \rangle$ $\langle \mathit{term} \rangle$ $\langle \mathit{term} \rangle$)	(Map_α_to_β α β Map_α_to_β)
$\mathrm{dom}(f)$	(dom $\langle \mathit{term} \rangle$)	(Map_α_to_β Set_of_α)
$\mathrm{rng}(f)$	(rng $\langle \mathit{term} \rangle$)	(Map_α_to_β Set_of_β)
$\mathrm{restrict}(l, m)$	(restrict $\langle \mathit{term} \rangle$ $\langle \mathit{term} \rangle$)	(Map_α_to_β Set_of_α Map_α_to_β)
$\mathrm{subtract}(l, m)$	(subtract $\langle \mathit{term} \rangle$ $\langle \mathit{term} \rangle$)	(Map_α_to_β Set_of_α Map_α_to_β)

8 An Example in VDM++

In order to illustrate the usage of the proposed theories, we consider the model of a sorting procedure written in the VDM++, the object-oriented version of VDM-SL (Fig. 2). Models like this can be authored, debugged, and analysed with the VDMTools[5] system. In particular, this tool is able to

[5] http://www.vdmtools.jp/en/

```
class $ExplSort$ is subclass of $Sorter$
operations
public
        $Sort : \mathbb{Z}^* \xrightarrow{o} \mathbb{Z}^*$
        $Sort\,(l) \triangleq$
          let  $r \in Permutations\,(l)$
               be st  $IsOrdered\,(r)$ in
          return $r$
functions
        $Permutations : \mathbb{Z}^* \to \mathbb{Z}^*$-set
        $Permutations\,(l) \triangleq$
          cases $l$ :
            $[], [\text{-}] \to \{l\},$
            others $\to \bigcup \{\{$append$([l\,(i)], j) \mid$
                $j \in Permutations\,(RestSeq\,(l, i))\} \mid$
              $i \in$ inds $l\}$
          end;
        $RestSeq : \mathbb{Z}^* \times \mathbb{N} \to \mathbb{Z}^*$
        $RestSeq\,(l, i) \triangleq$
          $[l\,(j) \mid j \in ($inds $l \setminus \{i\})]$
        pre  $i \in$ inds $l$
        post elems $RESULT \subseteq$ elems $l \wedge$
              len $RESULT =$ len $l - 1$ ;
        $IsOrdered : \mathbb{Z}^* \to \mathbb{B}$
        $IsOrdered\,(l) \triangleq$
          $\forall\, i, j \in$ inds $l \cdot i > j \;\Rightarrow\; l\,(i) \geq l\,(j)$
end $ExplSort$
```

```
class $Sorter$
operations
public
        $Sort : \mathbb{Z}^* \xrightarrow{o} \mathbb{Z}^*$
        $Sort\,(arg) \triangleq$
          is subclass responsibility
end $Sorter$
```

Figure 2: Model of a sorting program in VDM++, taken from the VDMTools distribution. It consists of an abstract superclass *Sorter* and a concrete subclass *ExplSort* that sorts by enumerating all permutations of a list.

generate verification conditions that ensure the well-formedness of models, but it lacks support for powerful automatic reasoners.

In order to verify that the expression $l(j)$ in the definition of *RestSeq* is well-defined, for instance, the following proof obligation is generated:

$$\forall l : \mathbb{L}(\mathbb{Z}), i : \mathbb{N}.\ \big(i \in \text{inds}(l) \Rightarrow \forall j \in \text{inds}(l) \setminus \{i\}.\ j \in \text{inds}(l)\big)$$

Using the proposed SMT-Lib theories, this can equivalently be expressed in SMT-Lib syntax (and eventually be verified automatically using SMT-Solvers):

```
(forall (?l List_of_Int) (?i Int) (implies
        (and (>= ?i 0) (in ?i (inds ?l)))
        (forall (?j Int) (implies
                (in ?j (setminus (inds ?l) (set ?i)))
                (in ?j (inds ?l))))))
```

It has to be stressed, of course, that this is only an example, and that the theories proposed in this document are not restricted to the VDMTools context.

9 Notes on Decision Procedures

In this proposal, we have mainly included those functions that are necessary to naturally translate problems from VDM-SL to SMT-Lib. The existence of decision procedures or efficient calculi for the resulting theories (or fragments of them) has been taken into account, but has not been the primary selection criterion. While this issue will be subject of further research, we sketch how SMT-solvers might be extended to support the new theories.[6]

Reduction to arrays. It is possible to reduce the basic operations of finite sets and maps to McCarthy's theory of arrays [6] (we only consider the case of sets at this point). A set from $\texttt{Set\_of\_}t$ would be represented by an array with index type t and value type `Boolean` (which possibly has to be encoded into `Int` if the booleans are not available as a type). Membership tests $a \in A$ and removing or adding single elements to a set ($A \cup \{a\}$ and $A \setminus \{a\}$) can then be expressed using the array operations `select` and `store`.

The theory of arrays can be extended (while keeping it decidable) by adding arrays with a *default value* as well as conditional operators on arrays that update one array in all points specified by a second array (this has been implemented in the Z3 theorem prover [7]). These additional operators allow to express empty sets (and thus arbitrary enumerations $\{a_1, \ldots, a_n\}$) as well as arbitrary unions, intersections, and differences of sets ($A \cup B$, $A \cap B$, $A \setminus B$). Using extensional arrays, also equality of sets ($A = B$) and the subset relationship ($A \subseteq B$) can be encoded. This covers all operations on sets but powersets ($\mathscr{P}(A)$) and cardinality ($|A|$), which might be added using uninterpreted functions and explicit axioms.

Reduction to algebraic datatypes. Many SMT-Solvers support the theory of algebraic datatypes, which can be used to naturally encode the basic operations of tuple types (the constructor $(a_1, \ldots, a_n)$ as well as the projections $\pi_i(a)$) and of lists (the constructor $[a_1, \ldots, a_n]$ and the destructors hd and tl). For the remaining operations, it may be necessary to use uninterpreted functions and explicit axioms.

Theory combination with shared BAPA. Wies et al. [11] describe an approach to combine theories by exchanging reductions to boolean algebra

[6] Some of the described techniques have been proposed by Nikolaj Bjørner in personal communication.

with Presburger arithmetic (BAPA). The theories for which such a combination is possible cover a large fragment of the language proposed in this paper, and are not restricted to quantifier-free formulae.

10 Conclusions and Related Work

We have proposed a collection of theories for finite datatypes that are of interest for software modelling and verification, and that currently lack adequate support in the SMT-Lib format. We are in the process of collecting examples that stem from VDM-SL specifications in order to create an initial set of benchmarks and to demonstrate the applicability of the new theories.

A related effort to create a problem interchange format is the TPTP format [10], which also includes theories such as set theory or number theory (such theories are defined using incomplete sets of axioms). Because the TPTP support for arithmetic is still in its infant stages, we believe that SMT-Lib currently is a better basis for exchanging verification problems.

References

[1] J.-R. Abrial. *The B Book: Assigning Programs to Meanings.* Cambridge University Press, Aug. 1996.

[2] H. Bruun, F. Damm, J. Dawes, B. Hansen, P. Larsen, G. Parkin, N. Plat, and H. Toetenel. A formal definition of VDM-SL. Technical Report 1998/9, University of Leicester, 1998.

[3] Á. Darvas, F. Mehta, and A. Rudich. Efficient well-definedness checking. In A. Armando, P. Baumgartner, and G. Dowek, editors, *International Joint Conference on Automated Reasoning (IJCAR)*, volume 5195 of *Lecture Notes in Computer Science*, pages 100–115. Springer-Verlag, 2008.

[4] Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.

[5] R. Hähnle. Many-valued logic, partiality, and abstraction in formal specification languages. *Logic Journal of the IGPL*, 13(4):415–433, 2005.

[6] J. McCarthy. Towards a mathematical science of computation. In C. M. Popplewell, editor, *Information Processing 1962: Proceedings IFIP Congress 62*, pages 21–28, Amsterdam, 1963. North Holland.

[7] L. D. Moura and N. Bjørner. Z3: An efficient smt solver. In *In Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2008.

[8] S. Ranise and C. Tinelli. The SMT-LIB standard, August 2006. version 1.2.

[9] J. M. Spivey. *The Z Notation: A Reference Manual.* Prentice Hall, 2nd edition, 1992.

[10] G. Sutcliffe and C. B. Suttner. The TPTP problem library—CNF release v1.2.1. *J. Autom. Reasoning*, 21(2):177–203, 1998.

[11] T. Wies, R. Piskac, and V. Kuncak. On combining theories with shared set operations. Technical report, EPFL, May 9 2009.