OpenMath and SMT-LIB

James H. Davenport
Department of Computer Science
University of Bath, Bath U.K.
J.H.Davenport@bath.ac.uk

Matthew England
Faculty of Engineering, Environment & Computing
Coventry University, Coventry, U.K.
Matthew.England@coventry.ac.uk

Roberto Sebastiani and Patrick Trentin
Dipartimento di Ingegneria e Scienza dell'Informazione (DISI),
Università di Trento, Treno, Italy
{roberto.sebastiani, patrick.trentin}@unitn.it

Abstract

OpenMath and SMT-LIB are languages with very different origins, but both "represent mathematics". We describe SMT-LIB for the OpenMath community and consider adaptations for both languages to support the growing SC^2 initiative.

1 Motivation: The SC^2 Project

The authors are all members or associates of the EU-funded Horizon 2020 Project SC^2. The overall aim of the project is to create a new research community bridging the gap between **S**atisfiability **C**hecking and **S**ymbolic **C**computation, so that members well informed about both fields can ultimately resolve problems currently beyond the scope of either.

The project was motivated by the movement of the Satisfiability Checking community outside of the Boolean SAT problem to consider how their techniques may perform on other domains, creating the field of Satisfiability Module Theories (SMT). The main idea here is to combine the sophisticated technology built for SAT with calls to domain-specific theory solvers when information beyond the logical structure is required. Most recently SMT has started to include the domain of non-linear polynomials over the reals, a field of study since the early days of Symbolic Computation.

However, as described in [Ábr15] it is not sufficient to call leading Computer Algebra Systems as theory solvers. Rather the algorithms need to be adapted to make them suitable for SMT. The two communities now find themselves addressing similar problems and so will share the challenge to improve their solutions to achieve applicability on complex large-scale applications. For further details we refer to the project introduction paper at CICM 2016 [ABB$^+$16] and the project website: `http://www.sc-square.org/CSA/welcome.html`.

1.1 An SC^2 Goal: Extending SMT-LIB

The increasing variety of the theories considered by SMT solvers created an urgent need for a common input language. The SMT-LIB initiative provided this and a large and increasing number of benchmarks. Once a problem is formulated in the SMT-LIB language, the user can employ any SMT solver to solve the problem. It

Copyright © by the paper's authors. Copying permitted for private and academic purposes.

In: A. Editor, B. Coeditor (eds.): Proceedings of the XYZ Workshop, Location, Country, DD-MMM-YYYY, published at http://ceur-ws.org

is housed at: `htpp://www.smt-lib.org`. The initiative has proved to spur on research, providing the basis for competitions and collaboration.

Although a proven valuable resource for the general SMT community (and far surpasses anything in Symbolic Computation), SMT-LIB has been found lacking on the domains relevant to SC^2. There is an NRA (Non-linear Real Arithmetic) category of the SMT-LIB benchmark library with several thousand problems in, but according to [JdM12] this consists mostly of problems originating from attempts to prove termination of term-rewrite systems. It has been noted in several papers how many of the problems are trivial (solved without calls to theory solvers) or come from a small number of classes and may have some hidden uniformity (see for example [ED16b]).

However, the work needed here is more than a greater variety of benchmark problems. Rather the depth of problems that can be tackled in this domain requires an extension of the SMT-LIB language instead. Indeed, this is one of the specific SC^2 goals, as described in SC^2 [ABB+16]:

> Extend the SMT-LIB language to cover a wider range of interests in the joint SC^2 community. These of course include conjunctive arithmetic fragments on various (maybe mixed) domains. Among other potential extensions are: optimisation (finding a solution maximising a goal function), allowing the use of differential equation theory, simplification of formulas, quantifier elimination.

Hence, at this stage in the development of SMT-LIB we consider what lessons can be learnt from the OpenMath community, and how both languages could be adapted to support the growing SC^2 initiative.

2 Background

2.1 MathML

MathML is described in [Wor14] and is usually encoded in XML. A special aspect of MathML is that there are two main strains of markup: Presentation Markup is used to display mathematical expressions while Content Markup is used to convey mathematical meaning [Wor14, §1.3].

2.2 OpenMath

The OpenMathStandard is in [BCC+04]. A new version is in preparation, but the only substantive change is to clarify the relationship with MathML. OpenMath objects are seen as trees, and can be encoded in XML or in binary — for readability we use the XML encoding here. There are various basic objects: symbols `OMS`, integers `OMI`, 64-bit IEEE [IEE08] floating point objects `OMF`, uninterpreted byte arrays `OMB`, strings `OMSTR` and variables `OMV`, all of which are leaves of the tree; and various constructions: application `OMA`, binding `OMB` and the statement of bound variables `OMBVAR`, errors `OME`, attributes `OMATTR` and attribution pairs `OMATP`, and foreign objects `OMFOREIGN`.

The mathematical expression $x + 1$ would be encoded as the application of the symbol '+' to x and 1:

```
<OMA>
  <OMS name="plus" cd="arith1"/>
  <OMV name="x"/>
  <OMI> 1 </OMI>
</OMA>
```

As can be seen this is somewhat verbose. An alternative syntax, POPCORN [HR09] has been proposed, which would shorten this algorithmically to `arith1.plus($x,1)`, and then, knowing about `arith1`, still further to `$x+1`.

2.2.1 Symbols

We have written this explicitly with the integer 1. If, however, we wanted the multiplicative identity of whatever ambient algebra we were working in, we would have used the nullary symbol `<OMS name="one" cd="alg1"/>`. `alg1` also defines `zero` as the additive identity.

The symbol `plus` comes from the **Content Dictionary** `arith1`, which contains various basic mathematical operators, including a (not necessarily commutative) `times` operator. If one wants an explicitly commutative `times` operator, there is one in `arith2` which has the property that $x*y = y*x$, or, from the Content Dictionary, the **Formal Mathematical Property** in Figure 1. These Formal Mathematical Properties describe (some of) teh semantics of the OpenMath symbols.

Figure 1: Commutatiivity of `times` from `arith2`

```
<FMP>
  <OMOBJ xmlns="http://www.openmath.org/OpenMath" version="2.0"
         cdbase="http://www.openmath.org/cd">
    <OMBIND>
      <OMS cd="quant1" name="forall"/>
      <OMBVAR>
         <OMV name="a"/>
         <OMV name="b"/>
      </OMBVAR>
      <OMA>
        <OMS cd="relation1" name="eq"/>
        <OMA>
          <OMS cd="arith2" name="times"/>
          <OMV name="a"/>
          <OMV name="b"/>
        </OMA>
        <OMA>
          <OMS cd="arith2" name="times"/>
          <OMV name="b"/>
          <OMV name="a"/>
        </OMA>
      </OMA>
    </OMBIND>
  </OMOBJ>
</FMP>
```

2.2.2 Small Type System

OpenMath *per se* is type-agnostic. It is expected that serious type systems will build, parallel with the Content Dictionary system, a set of files describing the type system for the symbols. There is a simple Small Type System described in [Dav00]. The entry for `<OMS name="times" cd="arith2"/>` (which comes from the file `arith2.sts`) is given in Figure 2: it states that the symbol is nary and associative[1], takes arguments from a structure which has the property `AbelianSemiGroup`, and returns an answer in the same `AbelianSemiGroup`.

2.2.3 Binders

OpenMath does not have a fixed set of binders. However, binders are introduced through a fixed syntactic marker `OMBIND` which enables correct recognition of free/bound variables. There is an example in Figure 1, and further discussion in [DK09].

2.3 SMT-LIB

This is described in [BFT15], though a near-final draft of the next version is in [BFT17].

SMT-LIB specifies four languages:

1. a language for writing terms and formulas in a sorted (i.e., typed) version of first-order logic;

2. a language for specifying background theories and fixing a standard vocabulary of sort, function, and predicate symbols for them;

3. a language for specifying logics, suitably restricted classes of formulas to be checked for satisfiability with respect to a specific background theory;

[1] Both properties have to be stated: set construction is nary but not associative, for example.

Figure 2: Small Type System for `times` from `arith2`

```
<Signature name="times">
<OMOBJ xmlns="http://www.openmath.org/OpenMath">
 <OMA>
  <OMS name="mapsto" cd="sts"/>
  <OMA>
   <OMS name="nassoc" cd="sts"/>
   <OMV name="AbelianSemiGroup"/>
  </OMA>
  <OMV name="AbelianSemiGroup"/>
 </OMA>
</OMOBJ>
</Signature>
```

4. a command language for interacting with SMT solvers via a textual interface that allows asserting and retracting formulas, querying about their satisfiability, examining their models or their unsatisfiability proofs, and so on.

Of these, Language 1 corresponds to OpenMath, Languages 2 and 3, essentially, to a typing system as in Section 2.2.2, and Language 4 more to a command system such as SCSCP [LHK+13].

The syntactic encoding of SMT-LIB expressions (Language 1 above) is as LISP S-expressions[2]:

```
<spec_constant> ::= <numeral> | <decimal> | <hexadecimal> | <binary> | <string>
<s_expr> ::= <spec_constant> | <symbol> | <keyword> | ( <s_expr>* )
```

Just as in OpenMath, the mathematical expression $x + 1$ would be encoded as the application of the symbol '+' to x and 1:

```
(+ x 1)
```

where `+` and `1` would be defined in the logic of the file: the logic itself would refer to a theory for arithmetic.

> Their [`spec_constant`] semantics is determined locally by each SMT-LIB theory that uses them. For instance, it is possible for an SMT-LIB theory of sets to use the numerals `0` and `1` to denote respectively the empty set and universal set. Similarly, the elements of `binary` may denote integers modulo n in one theory and binary strings in another; the elements of `decimal` may denote rational numbers in one theory and floating point values in another.

This contrasts with OpenMath's behaviour, as described in Section 2.2.1: OpenMath would use `<OMS name="one" cd="alg1"/>` to get the effect SMT-LIB gets from `1`.

2.3.1 Well-Sorted Terms

The sort declarations in SMT-LIB state the types of `symbol` operators, so that we can state that an SMT-LIB formula is well-sorted with respect to a given sort declaration. Besides the sorts used in the theories, it is possible to declare other sorts, whose domain is uninterpreted, that is, they can be interpreted as any non-empty set of elements. Also, besides the functions and predicates provided by the theories, uninterpreted functions and predicates can be used; these can have uninterpreted sort either as a domain or as range, but SMT-LIB makes it also possible to declare, for instance, a unary function from integer to integer that is arbitrary.

2.3.2 Binders

SMT-LIB has precisely three binders: `let`, `forall` and `exists`. The last two are *sorted*, in the sense that the syntax is

$$(\texttt{forall}\ ((x_1\ \sigma_1)\cdots(x_n\ \sigma_n))\ \phi).$$

[2] An argument for an XML encoding is made in [MJ04], but there has been little progress on this recently.

The semantics are those of nested unary `forall`, so that, while the x_i may be repeated, earlier occurrences are shadowed by later ones. For `let`, the syntax is

$$(\texttt{let}\ ((x_1\ \tau_1)\cdots(x_n\ \tau_n))\ \phi),$$

equivalent to the mathematical $\phi[\tau_1/x_1,\ldots,\tau_n/x_n]$ with simultaneous substitution. Hence the x_i must be distinct in this case.

3 Exists Uniquely

A relatively recent[3] addition to mathematical notation is $\exists!$, meaning "exists uniquely". It is, of course, not logically necessary: two alternative definitions[4] are as follows:

$$\exists! P(x) \Leftrightarrow \exists x\,(P(x) \wedge \forall y(P(y) \Rightarrow x = y)) \tag{1}$$

$$\exists! P(x) \Leftrightarrow (\exists x P(x)) \wedge (\forall y \forall z P(y) \wedge P(z) \Rightarrow y = z) \tag{2}$$

Considered computationally, (1) introduces an alternation, but fewer distinct quantifiers, and fewer repetitions of P than (2).

3.1 OpenMath

Since it is both useful and economical (saving the repetition of P, and the human/computer needing to recognise that it is the same P), there seems no reason not to introduce it.

3.2 SMT-LIB

Here the argument is more finely balanced. The arguments for are the same as for OpenMath (except that an SMT solver is expected to have clever heuristics, and idiom recognition might well be one of those). The converse argument is that adding syntactic sugar is adding noise too.

4 Maxima

4.1 OpenMath

The fundamental construct in OpenMath is `minmax1.max`, which returns the maximum of a set. Hence we could encode $\max_{x\in[0,1]} x(1-x)$ (whose value is $\frac{1}{4}$) as the following.

```
<OMA>
  <OMS cd="minmax1" name="max"/>
  <OMA>
    <OMS cd="set1" name="map"/>
    <OMBIND>
      <OMS cd="fns1" name="lambda"/>
      <OMBVAR>
        <OMV name="x"/>
      </OMBVAR>
      <OMA>
        <OMS cd="arith1" name="times"/>
        <OMV name="x"/>
        <OMA>
          <OMS cd="arith1" name="minus"/>
          <OMI> 1 </OMI>
          <OMV name="x"/>
        </OMA>
      </OMA>
    </OMBIND>
```

[3] A textbook usage is [Alu09, p. 3], but it is more commonly found in research papers.
[4] There are more perverse but more compact ones.

```
    <OMA>
      <OMS cd="interval1" name="interval_cc"/>
      <OMI> 0 </OMI>
      <OMI> 1 </OMI>
    </OMA>
  </OMA>
<OMA>
```

This is a perfectly legitimate encoding, but one could argue that it is not very constructive, since the maximum is being taken over an uncountable set. Essentially, `minmax1.max(set1.map(...))` is an idiom for "use the calculus operational semantics of max", except of course when it isn't. It would be more helpful to have an explicit max constructor that took a set and a function. There are essentially two options here (OpenMath could, of course, adopt both).

1. An operator that took both a set and a function: essentially making explicit the idiom `minmax1.max(set1.map(...))` referred to above.

2. A binder that took both a function body and a predicate, using the same bound variable for both.

```
    <OMBIND>
      <OMS cd="minmax2" name="max"/>
      <OMBVAR>
        <OMV name="x"/>
      </OMBVAR>
      <OMS cd="arith1" name="times"/>
      <OMV name="x"/>
      <OMA>
        <OMS cd="arith1" name="minus"/>
        <OMI> 1 </OMI>
        <OMV name="x"/>
      </OMA>
      <OMA>
        <OMS cd="set1" name="in"/>
        <OMV name="x"/>
        <OMA>
          <OMS cd="interval1" name="interval_cc"/>
          <OMI> 0 </OMI>
          <OMI> 1 </OMI>
        </OMA>
      </OMA>
    <OMA>
```

The second one probably has the abvantage of being closer to common usage.

A further complication is the lack of distinction between max and inf.

4.2 SMT-LIB

SMT-LIB does not have a max operator. However, OptiMathSAT's input language [ST15] is SMT-LIB extended with `maximize`, `minimize` "commands". In fact, these are statements as to the nature of the goal(s), and the goal is achieved by `check-sat`.

4.3 argmax

A relatively recent piece of mathematical notation is argmax, which does not have an extremely formal definition. The Wikipedia[5] definition is that these "are the points of the domain of some function at which the function

[5] `https://en.wikipedia.org/wiki/Arg_max` [20th June 2017].

values are maximized." Hence naïvely,

$$\operatorname*{argmax}_{x \in \mathbf{R}} \sin(x) = \left\{ \frac{\pi}{2} + 2n\pi | n \in \mathbf{Z} \right\}.$$

Though it can be defined in terms of other objects, it might be helpful to have a `argmax` constructor in OpenMath, capable of encoding $\operatorname{argmax}_{x \in [0,1]} x(1-x)$ (whose value is $\left\{\frac{1}{2}\right\}$) .

This is a perfectly sound mathematical definition, but does not really meet the requirements of SC^2, or computation in general. What SC^2 really needs is a *witness* point, i.e. a single value x_0 such that $f(x_0) = \max_{f \in S} f(x)$. For the sake of mathematical notation, we term this argmax$^{(1)}$ — one important point would be that it is not necessarily deterministic. This constructor could be called `argmaxone`.

The OptiMathSAT approach is, *after* calling `check-sat`, to allow `(get-value argument1)` etc. to find the values at which the maximum discovered by `check-sat` was achieved. This is essentially an argmax$^{(1)}$ approach. The precise details are more technical, as multiple maxima can be searched for: see Appendix A.

5 Output formats

OpenMath is agnostic about whether its formulae are input or output: it is aimed at a world where one system's output is the next system's input. SMT-LIB, by contrast, is largelythe input language for an SMT system. The output is, curdely, either `SAT` or `UNSAT`. However, `SAT` should have a model produced, i.e. a constructive demonstration of satisfiability.

The SMT-LIB standard does not specify what a model should be. There is thus the possibility to use any term, possibly to build algebraic numbers, etc... However, the current SMT-LIB requires that two terms with different expressions in the description of the model have a different value in the model. So it would be necessary to use some form of canonicalization of algebraic numbers that guarantees this.

6 Conclusions

6.1 Recommendations to OpenMath

1. Formalise the rôle of POPCORN in the OpenMath stable.
2. Consider an explicit ∃! constructor (Section 3.1).
3. Consider an explicit max constructor that took a set and a function (Section 4.1). OpenMath would then need to decide which variant(s) to adopt.
4. Clarify distinction between max and inf (Section 4.1).
5. Consider an explicit argmax constructor (Section 4.3).
6. Consider an explicit argmax$^{(1)}$ constructor (Section 4.3).

6.2 Recommendations to SMT-LIB

1. Consider having an operator to express quantifier elimination. This would also mean expressing the output, which could be done by reusing the input language[6].
2. Consider an explicit max constructor.
3. Consider an explicit argmax$^{(1)}$ constructor.

Acknowledgements

We are grateful to Pascal Fontaine (LORIA) for many useful comments.

We are grateful for support by the H2020-FETOPEN-2016-2017-CSA project SC^2 (712689).

[6] At least over the reals, by the Tarski–Seidenberg Theorem.

References

[Ábr15] Ábrahám, E.: Building bridges between symbolic computation and satisfiability checking. In: Proceedings ISSAC 2015. pp. 1–6. ACM (2015)

[Alu09] Aluffi,P., *Algebra: Chapter 0*. American Mathematical Society, 2009.

[ABB+16] E. Ábrahám, B. Becker, A. Bigatti, B. Buchberger, C. Cimatti, J.H. Davenport, M. England, P. Fontaine, S. Forrest, D. Kroening, W. Seiler, and T. Sturm. SC^2: Satisfiability Checking meets Symbolic Computation (Project Paper). In *Proceedings CICM 2016*, pages 28–43, 2016.

[BCC+04] S. Buswell, O. Caprotti, D.P. Carlisle, M.C. Dewar, M. Gaëtano, and M. Kohlhase. The OpenMath Standard 2.0. `http://www.openmath.org`, 2004.

[BFT15] C. Barrett, P. Fontaine, and C. Tinelli. The SMT-LIB Standard: Version 2.5. `http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.5-r2015-06-28.pdf`, 2015.

[BFT17] C. Barrett, P. Fontaine, and C. Tinelli. The SMT-LIB Standard: Version 2.6 (draft 5 June 2017). `http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.6-draft-2017-06-05.pdf`, 2017.

[Dav00] J.H. Davenport. A Small OpenMath Type System. *ACM SIGSAM Bulletin 2*, 34:16–21, 2000.

[DK09] J.H. Davenport and M. Kohlhase. Quantifiers and Big Operators in OpenMath. `https://www.researchgate.net/profile/Dan_Roozemond/publication/253932330_OpenMath_in_SCIEnce_Evolving_of_Symbolic_Computation_Interaction/links/00b7d5375cc5cea0ce000000.pdf#page=119`, 2009.

[ED16b] M. England and J.H. Davenport. Experience with heuristics, benchmarks & standards for cylindrical algebraic decomposition. In *Proceedings of the 1st Workshop on Satisfiability Checking and Symbolic Computation (SC^2 2016)*, number 1804 in CEUR Workshop Proceedings, 2016.

[HR09] P. Horn and D. Roozemond. OpenMath in SCIEnce: SCSCP and POPCORN. In J. Carette *et al.*, editors, *Proceedings Intelligent Computer Mathematics*, Springer Lecture Notes in Artificial Intelligence 5625, pages 474–479, 2009.

[IEE08] IEEE. *IEEE Standard for Floating-Point Arithmetic (754-2008)*. IEEE, 2008.

[JdM12] D. Jovanovic and L. de Moura. Solving non-linear arithmetic. In B. Gramlich, D. Miller, and U. Sattler, editors, *Automated Reasoning: 6th International Joint Conference (IJCAR)*, volume 7364 of *Lecture Notes in Computer Science*, pages 339–354. Springer, 2012.

[LHK+13] S. Linton, K. Hammond, A. Konovalov, C. Brown, P.W. Trinder, H.W. Loidl, P. Horn, and D. Roozemond. Easy composition of symbolic computation software using SCSCP: A new Lingua Franca for symbolic computation. *Journal of Symbolic Computation*, 49:95–119, 2013.

[MJ04] F. Marić and P. Janičić. SMT-LIB in XML clothes. *2nd International Workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR-04)*, pages 86–89, 2004.

[ST15] R. Sebastiani and P. Trentin. OptiMathSAT: A Tool for Optimization Modulo Theories. In Daniel Kroening and Corina S. Păsăreanu, editors, *Computer Aided Verification: 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, pages 447–454, Cham, 2015. Springer International Publishing.

[Wor14] World-Wide Web Consortium. Mathematical Markup Language (MathML) Version 3.0: second edition. `http://www.w3.org/TR/2014/REC-MathML3-20140410/`, 2014.

A OptiMathSAT Example

```
(set-option :produce-models true)

(declare-fun cost () Real)
(declare-fun s1 () Bool)
(declare-fun s2 () Bool)
(declare-fun s3 () Bool)
(declare-fun s4 () Bool)
(declare-fun q1 () Real)
(declare-fun q2 () Real)
(declare-fun q3 () Real)
(declare-fun q4 () Real)

; set goods quantity
(assert (= 250 (+ q1 q2 q3 q4)))

; set goods offered by each supplier
(assert (or (= q1 0) (and (<= 50 q1) (<= q1 250))))
(assert (or (= q2 0) (and (<= 100 q2) (<= q2 150))))
(assert (or (= q3 0) (and (<= 100 q3) (<= q3 100))))
(assert (or (= q4 0) (and (<= 50 q4) (<= q4 100))))

; supplier is used if sends more than zero items
(assert (and (=> s1 (not (= q1 0))) (=> s2 (not (= q2 0)))
           (=> s3 (not (= q3 0))) (=> s4 (not (= q4 0)))))

; supply from the largest number of suppliers
(assert-soft s1 :id unused_suppliers)
(assert-soft s2 :id unused_suppliers)
(assert-soft s3 :id unused_suppliers)
(assert-soft s4 :id unused_suppliers)

; set goal (A)
(minimize (+ (* q1 23) (* q2 21) (* q3 20) (* q4 10)))
; set goal (B)
(minimize (+ (* q1 25) (* q2 19) (* q3 10) (* q4 20)))

; box: independent optimization
(set-option :opt.priority box)
(check-sat)

; print model for A
(set-model 0)
(get-value (q1))
(get-value (q2))
(get-value (q3))
(get-value (q4))

; print model for B
(set-model 1)
(get-model)
```