SMT-LIB Proposal: Integer Arithmetic with Exponentials

Florian Frohn[1], Hanna Lachnitt[2], Philipp Schroer[1], and Yoni Zohar[3]

[1]RWTH Aachen University, Aachen, Germany
[2]Stanford University, Stanford, USA
[3]Bar-Ilan University, Ramat Gan, Israel

November 20, 2025

1 Introduction

We propose to extend the SMT-LIB theory `Ints` with exponentiation. Such an extension is motivated by various use cases, e.g.:[1]

- SMT with parametric bit-vectors [1]
- automatic verification of infinite state systems [2]
- automatic verification of solutions for recurrence equations [3]
- deductive verification with Caesar [4]

2 Proposal

The following proposal is based on the SMT-LIB theory `Ints`.[2]

```
(theory Ints

 :sorts ((Int 0))

 :funs ((NUMERAL Int)
        (- Int Int)                    ; negation
        (- Int Int Int :left-assoc)    ; subtraction
        (+ Int Int Int :left-assoc)
        (* Int Int Int :left-assoc)
        (** Int Int Int)               ; exponentiation
```

[1]Further information on use cases and available benchmarks can be found here: https://github.com/ffrohn/exp-smt-lib/issues/5

[2]https://smt-lib.org/theories-Ints.shtml

```
       (div Int Int Int :left-assoc)
       (mod Int Int Int)
       (abs Int Int)
       (<= Int Int Bool :chainable)
       (<  Int Int Bool :chainable)
       (>= Int Int Bool :chainable)
       (>  Int Int Bool :chainable)
      )

:funs-description
"All ranked function symbols of the form
   ((_ divisible n) Int Bool)
 where n is a positive numeral.
"

:values
"The set of values for the sort Int consists of
 - all numerals,
 - all terms of the form (- n) where n is a numeral other than 0.
"

:definition
"For every expanded signature, the instance of Ints with that
 signature is the theory consisting of all Sigma-models that
   interpret:

 - the sort Int as the set of all integer numbers,

 - each numeral as the corresponding natural number,

 - (_ divisible n) as the function mapping to true all and only
   the integers that are divisible by n,

 - abs as the absolute value function,

 - div and mod according to Boute's Euclidean definition [1], that
   is, so as to satify the formula

   (forall ((m Int) (n Int))
     (=> (distinct n 0)
         (let ((q (div m n)) (r (mod m n)))
           (and (= m (+ (* n q) r))
                (<= 0 r (- (abs n) 1))))))

 - ** as exponentiation if the second argument is non-negative.
   So in particular, (= (** 0 0) 1) holds.
   If n is negative, then ** is defined as to satisfy the formula

   (= (** m n) (div 1 (** m (- n))))

   So the following holds for all negative integers n:
   - (= (** 0 n) (div 1 0))
   - (= (** m n) (** m (- n))) if (= (abs m) 1)
   - (= (** m n) 0)            if (> (abs m) 1)

 - the other function symbols of Ints as expected.

```

```
 References:
 [1] Boute, Raymond T. (April 1992).
     The Euclidean definition of the functions div and mod.
     ACM Transactions on Programming Languages and Systems
     (TOPLAS) ACM Press. 14 (2): 127 - 144.
     doi:10.1145/128861.128862.
"

:notes
"Regardless of sign of m,
 when n is positive, (div m n) is the floor of the rational number
 m/n; when n is negative, (div m n) is the ceiling of m/n.

 This contrasts with alternative but less robust definitions of
 div and mod where (div m n) is
 - always the integer part of m/n (rounding towards 0), or
 - always the floor of x/y (rounding towards -infinity).
"

:notes
"See note in the Reals theory declaration about terms of the form
 (/ t 0). The same observation applies here to terms of the form
 (div t 0) and (mod t 0).
"
)
```

3 Justification

While working on this proposal, several question have been discussed by the authors of this document, and may require further discussion in the community.

3.1 Associativity

Exponentiation[3] is usually right-associative. However, the authors agree that the use of exponentiation in an associative manner is far less common than for other associative operators like addition or multiplication. Thus, declaring it as right-associative may cause bugs and confusion, and therefore keeping it strictly binary appears to be the better choice.

3.2 Semantics for Negative Exponents

We[4] discussed several possibilities for the semantics of exponentiation with negative exponents. We think that the following two variants are best:

[3] The details of the discussion are available here: https://github.com/ffrohn/exp-smt-lib/issues/6

[4] The details of the discussion are available here: https://github.com/ffrohn/exp-smt-lib/issues/2

- for all $n > 0$, we define $b^{-n} = 1 \text{ div } b^n$, i.e.,

$$b^{-n} = \begin{cases} 1 \text{ div } 0 & \text{if } b = 0 \\ b^n & \text{if } |b| = 1 \\ 0 & \text{if } |b| > 1 \end{cases}$$

  For the first case, the usual treatment of division by zero in SMT-LIB applies, i.e., the term 1 div 0 is meaningful, but its interpretation is not restricted by the `Ints` theory.

- exponentiation with a negative exponent is treated like an uninterpreted function, as in the case of division by zero

The advantage of the first variant is that it avoids introducing further partial / uninterpreted functions. The advantage of the second version is that it's least surprising from a user's perspective, in particular for users that are aware of the handling of division by zero in SMT-LIB. After consulting with the SMT-LIB coordinators, we propose the first variant.

3.3 Name of the Function Symbol for Exponentiation

We[5] considered several names, and identified `**` and `pow` as the best options. In the proposal, we chose `**`, as SMT-LIB usually uses "operators" (like `+`, `*`, etc.) instead of "abbreviations" if possible. Note that we preferred `**` over `^`, as the latter may be confused with bitwise XOR.

3.4 Unary Exponential Functions

We[6] also considered including unary functions for exponentiation with fixed bases into the proposal:

```
All ranked function symbols of the form
  ((n iexp _) Int Int)
where n is a positive numeral.
```

Here, `iexp` abbreviates "integer exponentiation". The motivation is that the extension of `LIA` with one of these function symbols (which is also known as Semenov Arithmetic) is decidable, whereas the extension of `LIA` with a binary function for exponentiation is undecidable.

We did not include such unary function symbols for simplicity.

3.5 Logics

[5] The details of the discussion are available here: https://github.com/ffrohn/exp-smt-lib/issues/1

[6] The details of the discussion are available here: https://github.com/ffrohn/exp-smt-lib/issues/3

If[7] exponentiation gets incorporated into the `Ints` theory, then all existing logics that refer to it have to be adapted in order to exclude exponentiation. Moreover, new logics that allow exponentiation should be defined. We propose to define (at least) the following additional logics (names to be discussed):

- `EIA`: `NIA` + exponentiation
- `QF_EIA`: `QF_NIA` + exponentiation
- `UFEIA`: `UFNIA` + exponentiation

An extension of `LIA` with exponentiation can simulate multiplication, so presumably, such a logic is not required.

References

[1] Zvika Berger, Yoni Zohar, Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark W. Barrett, and Cesare Tinelli. Bit-precise reasoning with parametric bit-vectors. In *SAT '25*, volume 341 of *LIPIcs*, pages 4:1–4:24, 2025. doi:10.4230/LIPICS.SAT.2025.4.

[2] Florian Frohn and Jürgen Giesl. Satisfiability modulo exponential integer arithmetic. In *IJCAR '24*, volume 14739 of *Lecture Notes in Computer Science*, pages 344–365, 2024. doi:10.1007/978-3-031-63498-7_21.

[3] Florian Frohn and Jürgen Giesl. Satisfiability modulo exponential integer arithmetic (extended version), 2025. arXiv:2402.01501.

[4] Philipp Schröer, Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. A deductive verification infrastructure for probabilistic programs. *Proc. ACM Program. Lang.*, 7(OOPSLA2):2052–2082, 2023. doi:10.1145/3622870.

[7] The details of the discussion are available here: https://github.com/ffrohn/exp-smt-lib/issues/4