Meeting BLaSST

Vincent Trélat

supervised by Stephan Merz and Sophie Tourret

January 11, 2024

Outline

My task from October 2023

Enhancing B language reasoners with SAT and SMT techniques

My task from October 2023

Enhancing B language reasoners with SAT and SMT techniques

Benchmarking

Benchmark

Dataset	POs	Previous	New (CVC5)
0001	49646	4%	81%
0002	99600	0.006%	20%
0003	49763	10%	12%
0030	90	47%	77%

Table: Comparison between Clearsy's benchmark and my benchmark using CVC5 in terms of number of discharged POs (rate w.r.t. total number of POs)

(New) options:

- timeout per query: 20 ms
- timeout: 2 min

My task from October 2023

Enhancing B language reasoners with SAT and SMT techniques

Benchmarking

Read documentation

My task from October 2023

Enhancing B language reasoners with SAT and SMT techniques

Benchmarking

Read documentation

Encoding B to SMT-LIB2.X

My task from October 2023

Enhancing B language reasoners with SAT and SMT techniques

Benchmarking

Read documentation

Encoding B to SMT-LIB2.X

Encoding B to SMT-LIB3 (HOL)

Based on Clearsy's work on *ppTransSMT*[1]:

- based on ZFC, anything is a set (functions are binary relations...)
- translation to FOL with uninterpreted functions

The encoding relies on two abstract sorts declared as follows:

```
(declare-sort P 1)
(declare-sort C 2)
```

[1] Matthias Konrad. *Translation from Set-Theory to Predicate Calculus*. en. Technical Report. ETH Zurich, 2012
David Déharbe. "Integration of SMT-solvers in B and Event-B development environments". In: *Science of Computer Programming* (2013)
David Déharbe et al. "Integrating SMT solvers in Rodin". In: *Science of Computer Programming* (2014)

Example 1

A function $f : \mathbb{Z} \rightarrow \mathbb{N}$ may be specified as follows:

```
(declare-fun f (Int) Int)
(assert (forall ((x Int)) (<= 0 (f x))))
```

Here, f *is* a function (in SMT-LIB).

Example 2

However, to quantify over a function $f : \mathbb{Z} \to \mathbb{N}$ with a quantifier `quantify`, one could write:

```
(quantify f (P (C Int Int))
  (=> (and
        (is_func f) ; f is a function
        (forall ((x Int)) (=> (forall ((y Int))
          (mem x y f)) (<= 0 y)))) ; spec
      (...)))
```

Example 2

However, to quantify over a function $f : \mathbb{Z} \to \mathbb{N}$ with a quantifier `quantify`, one could write:

```
(quantify f (P (C Int Int))
  (=> (and
        (is_func f) ; f is a function
        (forall ((x Int)) (=> (forall ((y Int))
          (mem x y f)) (<= 0 y)))) ; spec
      (...)))
```

Where `is_func f` is a macro for

$$\forall xyz.(x, y) \in f \wedge (x, z) \in f \implies y = z$$

Example 2

However, to quantify over a function $f : \mathbb{Z} \to \mathbb{N}$ with a quantifier `quantify`, one could write:

```
(quantify f (P (C Int Int))
  (=> (and
        (is_func f) ; f is a function
        (forall ((x Int)) (=> (forall ((y Int))
          (mem x y f)) (<= 0 y)))) ; spec
      (...)))
```

The specification of f could be written as:

$$\forall x. f(x) \geq 0$$

Example 2

However, to quantify over a function $f : \mathbb{Z} \to \mathbb{N}$ with a quantifier `quantify`, one could write:

```
(quantify f (P (C Int Int))
  (=> (and
        (is_func f) ; f is a function
        (forall ((x Int)) (=> (forall ((y Int))
          (mem x y f)) (<= 0 y)))) ; spec
      (...)))
```

The specification of f could be written as:

$$\forall x.(\forall y.(x, y) \in f \implies y \geq 0)$$

Example 2

However, to quantify over a function $f : \mathbb{Z} \to \mathbb{N}$ with a quantifier `quantify`, one could write:

```
(quantify f (P (C Int Int))
  (=> (and
        (is_func f) ; f is a function
        (forall ((x Int)) (=> (forall ((y Int))
          (mem x y f)) (<= 0 y)))) ; spec
      (...)))
```

The specification of f could be written as:

$$\forall x.(\forall y.(x, y) \in f \implies y \geq 0)$$

Example 2

However, to quantify over a function $f : \mathbb{Z} \rightarrow \mathbb{N}$ with a quantifier `quantify`, one could write:

```
(quantify f (P (C Int Int))
  (=> (and
        (is_func f) ; f is a function
        (forall ((x Int)) (=> (forall ((y Int))
          (mem x y f)) (<= 0 y)))) ; spec
      (...)))
```

The specification of f could be written as:

$$\forall x.(\forall y.\texttt{mem}\ x\ y\ f \implies y \geq 0)$$

Example 2

However, to quantify over a function $f : \mathbb{Z} \to \mathbb{N}$ with a quantifier `quantify`, one could write:

```
(quantify f (P (C Int Int))
  (=> (and
        (is_func f) ; f is a function
        (forall ((x Int)) (=> (forall ((y Int))
          (mem x y f)) (<= 0 y)))) ; spec
      (...)))
```

The specification of f could be written as:

$$\forall x.(\forall y.\texttt{mem}\ x\ y\ f \implies y \geq 0)$$

Here, f *is specified as* a function *seen as* a set of pairs, even though in SMT-LIB it is an element of type `P (C Int Int)`.

Membership is also uninterpreted. One could want membership to be declared as an SMT-LIB *function* (or predicate) parameterized by some type `X`:

```
(declare-fun mem (X (P X)) Bool)
```

Since SMT-LIB 2.X has no parametric polymorphism or dependent types, we need to declare a new membership symbol / function / predicate for each type.

Example 3

Membership for integers:

```
(declare-fun mem_i (Int (P Int)) Bool)
```

Membership for functions from integers to integers:

```
(declare-fun mem_f ((P (C Int Int))
  ((P (P (C Int Int))))) Bool)
```

Set axiomatization

```
(define-fun |def_B definitions_0| () Bool
  (forall ((x Int)) (=
    (and (<= 0 x) (<= x MaxInt))
    (and (<= 0 x) (<= x MaxInt)))))
```

```
(define-fun |def_B definitions_1| () Bool
  (forall ((x Int)) (=
    (and (>= x MinInt) (<= x MaxInt))
    (and (>= x MinInt) (<= x MaxInt)))))
```

Set axiomatization

```
(define-fun |def_B definitions_0| () Bool
  (forall ((x Int)) (=
    (and (<= 0 x) (<= x MaxInt))
    (and (<= 0 x) (<= x MaxInt)))))
```

$$\forall x\colon \texttt{Int}.0 \leq x \leq \texttt{MaxInt} \iff 0 \leq x \leq \texttt{MaxInt}$$

```
(define-fun |def_B definitions_1| () Bool
  (forall ((x Int)) (=
    (and (>= x MinInt) (<= x MaxInt))
    (and (>= x MinInt) (<= x MaxInt)))))
```

$$\forall x\colon \texttt{Int}.\texttt{MinInt} \leq x \leq \texttt{MaxInt} \iff \texttt{MinInt} \leq x \leq \texttt{MaxInt}$$

Set axiomatization

```
(define-fun |def_B definitions_0| () Bool
  (forall ((x Int)) (=
    (and (<= 0 x) (<= x MaxInt))
    (and (<= 0 x) (<= x MaxInt)))))
```

$$\forall x : \mathtt{Int}.0 \leq x \leq \mathtt{MaxInt} \iff 0 \leq x \leq \mathtt{MaxInt}$$

```
(define-fun |def_B definitions_1| () Bool
  (forall ((x Int)) (=
    (and (>= x MinInt) (<= x MaxInt))
    (and (>= x MinInt) (<= x MaxInt)))))
```

$$\forall x : \mathtt{Int}.\mathtt{MinInt} \leq x \leq \mathtt{MaxInt} \iff \mathtt{MinInt} \leq x \leq \mathtt{MaxInt}$$

Motivation: specify B's finite sets of naturals `NAT` and integers `INT`.

Set axiomatization

```
(define-fun |def_B definitions_0| () Bool
  (forall ((x Int)) (=
    (and (<= 0 x) (<= x MaxInt))
    (and (<= 0 x) (<= x MaxInt)))))
```

$$\forall x\colon \texttt{Int}.0 \le x \le \texttt{MaxInt} \iff 0 \le x \le \texttt{MaxInt}$$

```
(define-fun |def_B definitions_1| () Bool
  (forall ((x Int)) (=
    (and (>= x MinInt) (<= x MaxInt))
    (and (>= x MinInt) (<= x MaxInt)))))
```

$$\forall x\colon \texttt{Int}.\texttt{MinInt} \le x \le \texttt{MaxInt} \iff \texttt{MinInt} \le x \le \texttt{MaxInt}$$

Motivation: specify (axiomatize) sets and membership more generally.

Set axiomatization

Example 4

- Let $S := \{x : \mathtt{X} \mid P(x)\}$ be a set of elements of type `X`.

Set axiomatization

Example 4

- Let $S := \{x : \mathtt{X} \mid P(x)\}$ be a set of elements of type `X`.
- Assume we have a predicate `mem` of type $\mathtt{X} \rightarrow \mathtt{P\ X} \rightarrow \mathtt{Bool}$.

Set axiomatization

Example 4

- Let $S := \{x : \texttt{X} \mid P(x)\}$ be a set of elements of type `X`.
- Assume we have a predicate `mem` of type $\texttt{X} \rightarrow \texttt{P X} \rightarrow \texttt{Bool}$.
- Then, membership on S can be axiomatized as follows:

$$\forall x\colon \texttt{X}. x \in S \iff P(x)$$

Set axiomatization

Example 4

- Let $S := \{x : \mathtt{X} \mid P(x)\}$ be a set of elements of type `X`.
- Assume we have a predicate `mem` of type $\mathtt{X} \to \mathtt{P\ X} \to \mathtt{Bool}$.
- Then, membership on S can be axiomatized as follows:

$$\forall x\colon \mathtt{X}.x \in S \iff P(x)$$

- In SMT-LIB, this may be encoded as:

```
(assert (forall ((x X)) (= (mem x S) (P x))))
```

With the support of HOL in SMT-LIB 3, we could encode B directly to SMT-LIB 3[2] using HO λ-expressions[3].

[2] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. *The Satisfiability Modulo Theories Library (SMT-LIB)*. 2016

[3] David Déharbe, Pascal Fontaine, and Bruno Woltzenlogel Paleo. "Quantifier Inference Rules for SMT proofs". In: 2011
Sophie Tourret et al. "Lifting congruence closure with free variables to λ-free higher-order logic via SAT encoding". In: 2020

With the support of HOL in SMT-LIB 3, we could encode B directly to SMT-LIB 3[2] using HO λ-expressions[3].

Where *ppTrans* intrisically encodes expressions of the form $x \in S$

Any support of HO syntax would allow a direct encoding of the set S and any expression $x \in S$ would be obtained by β-reduction.

[2] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. *The Satisfiability Modulo Theories Library (SMT-LIB)*. 2016

[3] David Déharbe, Pascal Fontaine, and Bruno Woltzenlogel Paleo. "Quantifier Inference Rules for SMT proofs". In: 2011
Sophie Tourret et al. "Lifting congruence closure with free variables to λ-free higher-order logic via SAT encoding". In: 2020

With the support of HOL in SMT-LIB 3, we could encode B directly to SMT-LIB 3[2] using HO λ-expressions[3].

Where *ppTrans* intrisically encodes expressions of the form $x \in S$

Example 5

$$y \in \{x \mid P(x) \vee Q(x)\} \quad \rightsquigarrow \quad \texttt{(or (P y) (Q y))}$$

Any support of HO syntax would allow a direct encoding of the set S and any expression $x \in S$ would be obtained by β-reduction.

[2] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. *The Satisfiability Modulo Theories Library (SMT-LIB)*. 2016

[3] David Déharbe, Pascal Fontaine, and Bruno Woltzenlogel Paleo. "Quantifier Inference Rules for SMT proofs". In: 2011
Sophie Tourret et al. "Lifting congruence closure with free variables to λ-free higher-order logic via SAT encoding". In: 2020

With the support of HOL in SMT-LIB 3, we could encode B directly to SMT-LIB 3[2] using HO λ-expressions[3].

Where *ppTrans* intrisically encodes expressions of the form $x \in S$

Example 5

$$y \in \{x \mid P(x) \vee Q(x)\} \quad \rightsquigarrow \quad \texttt{(or (P y) (Q y))}$$

Any support of HO syntax would allow a direct encoding of the set S and any expression $x \in S$ would be obtained by β-reduction.

Example 6

$$\{x \mid P(x) \vee Q(x)\} \quad \rightsquigarrow \quad \texttt{(lambda (x } \tau\texttt{) (or (P x) (Q x)))}$$

[2] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. *The Satisfiability Modulo Theories Library (SMT-LIB)*. 2016

[3] David Déharbe, Pascal Fontaine, and Bruno Woltzenlogel Paleo. "Quantifier Inference Rules for SMT proofs". In: 2011
Sophie Tourret et al. "Lifting congruence closure with free variables to λ-free higher-order logic via SAT encoding". In: 2020

With the support of HOL in SMT-LIB 3, we could encode B directly to SMT-LIB 3[2] using HO λ-expressions[3].

Where *ppTrans* intrisically encodes expressions of the form $x \in S$

Example 5

$$y \in \{x \mid P(x) \vee Q(x)\} \quad \rightsquigarrow \quad \texttt{(or (P y) (Q y))}$$

Any support of HO syntax would allow a direct encoding of the set S and any expression $x \in S$ would be obtained by β-reduction.

Example 6

$y \in \{x \mid P(x) \vee Q(x)\} \rightsquigarrow$

[2] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. *The Satisfiability Modulo Theories Library (SMT-LIB)*. 2016

[3] David Déharbe, Pascal Fontaine, and Bruno Woltzenlogel Paleo. "Quantifier Inference Rules for SMT proofs". In: 2011
Sophie Tourret et al. "Lifting congruence closure with free variables to λ-free higher-order logic via SAT encoding". In: 2020

With the support of HOL in SMT-LIB 3, we could encode B directly to SMT-LIB 3[2] using HO λ-expressions[3].

Where *ppTrans* intrisically encodes expressions of the form $x \in S$

Example 5

$$y \in \{x \mid P(x) \vee Q(x)\} \quad \rightsquigarrow \quad \texttt{(or (P y) (Q y))}$$

Any support of HO syntax would allow a direct encoding of the set S and any expression $x \in S$ would be obtained by β-reduction.

Example 6

$y \in \{x \mid P(x) \vee Q(x)\} \rightsquigarrow$ `(lambda (x τ) (or (P x) (Q x))) y`

[2] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. *The Satisfiability Modulo Theories Library (SMT-LIB)*. 2016

[3] David Déharbe, Pascal Fontaine, and Bruno Woltzenlogel Paleo. "Quantifier Inference Rules for SMT proofs". In: 2011
Sophie Tourret et al. "Lifting congruence closure with free variables to λ-free higher-order logic via SAT encoding". In: 2020

With the support of HOL in SMT-LIB 3, we could encode B directly to SMT-LIB 3[2] using HO λ-expressions[3].

Where *ppTrans* intrisically encodes expressions of the form $x \in S$

Example 5

$$y \in \{x \mid P(x) \vee Q(x)\} \quad \rightsquigarrow \quad \texttt{(or (P y) (Q y))}$$

Any support of HO syntax would allow a direct encoding of the set S and any expression $x \in S$ would be obtained by β-reduction.

Example 6

$y \in \{x \mid P(x) \vee Q(x)\} \rightsquigarrow$ `(or (P y) (Q y))`

[2] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. *The Satisfiability Modulo Theories Library (SMT-LIB)*. 2016

[3] David Déharbe, Pascal Fontaine, and Bruno Woltzenlogel Paleo. "Quantifier Inference Rules for SMT proofs". In: 2011
Sophie Tourret et al. "Lifting congruence closure with free variables to λ-free higher-order logic via SAT encoding". In: 2020

Membership is also uninterpreted. One could want membership to be declared as an SMT-LIB *function* (or predicate) parameterized by some type `X`:

```
(declare-fun mem (X (P X)) Bool)
```

SMT-LIB 3 also supports polymorphism and dependent types, so membership could be declared as follows:

```
(declare-const mem
  (-> (! Type :var T :implicit) T (P T) Bool))
```

A more direct encoding could also reduce the number of quantifiers that stem from the translation in the resulting SMT-LIB 3 formula.

A more direct encoding could also reduce the number of quantifiers that stem from the translation in the resulting SMT-LIB 3 formula.

Instead of reducing $\{x \mid P(x)\} = \{x \mid Q(x)\}$ to

$$\forall x. P(x) = Q(x)$$

A more direct encoding could also reduce the number of quantifiers that stem from the translation in the resulting SMT-LIB 3 formula.

Instead of reducing $\{x \mid P(x)\} = \{x \mid Q(x)\}$ to

$$\forall x. P(x) = Q(x)$$

we may directly encode it as

$$\lambda x. P(x) = \lambda x. Q(x) \quad \rightarrow_\eta \quad P = Q$$

Remark: SMT-LIB 3 defines `forall` as an abbreviation for a λ-expression, so both encodings are equivalent in SMT-LIB 3. However, the λ-expression actually results from encoding sets directly.

Nice to meet you all and all the best for 2024!