Enhancing B language reasoners with SMT techniques

Vincent Trélat

Université de Lorraine, CNRS, Inria, Loria, Nancy, France

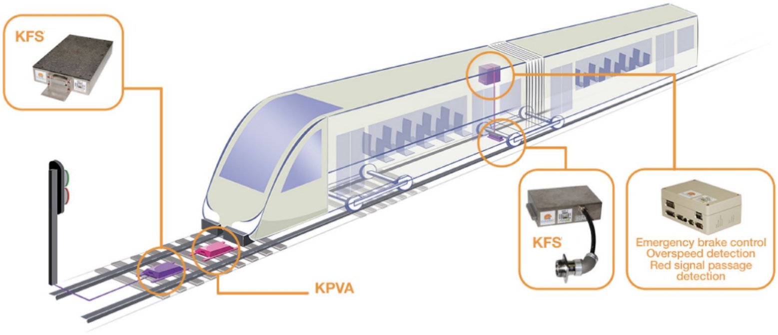
Outline

Context

Formal methods for safety-critical systems, e.g. railways

Context

Formal methods for safety-critical systems, e.g. railways



CLEARSY
Safety Solutions Designer

Context

Subject: Enhancing B language reasoners with SMT techniques

Context

Subject: Enhancing B language reasoners with SMT techniques

```
(declare-fun f (-> Int (Option Int)))
(assert (forall ((x Int))
  (= (not (= (f x) none)) (<= 0 x))))
(assert (forall ((x Int)) (=>
  (not (= (f x) none))
  (exists ((y Int))
    (and (<= a y) (<= y b) (= (f x) (some y)))))))
```

Context

Subject: Enhancing B language reasoners with SMT techniques

```
CONSTANTS
  a, b
VARIABLES
  f
INITIALISATION
  a : INTEGER & b : INTEGER &
  f : NATURAL --> a .. b
```

```
(declare-fun f (-> Int (Option Int)))
(assert (forall ((x Int))
  (= (not (= (f x) none)) (<= 0 x))))
(assert (forall ((x Int)) (=>
  (not (= (f x) none))
  (exists ((y Int))
    (and (<= a y) (<= y b) (= (f x) (some y)))))))
```

Context

Subject: Enhancing B language reasoners with SMT techniques

```
CONSTANTS
  a, b
VARIABLES
  f
INITIALISATION
  a : INTEGER & b : INTEGER &
  f : NATURAL --> a .. b
```

→

```
(declare-fun f (-> Int (Option Int)))
(assert (forall ((x Int))
  (= (not (= (f x) none)) (<= 0 x))))
(assert (forall ((x Int)) (=>
  (not (= (f x) none))
  (exists ((y Int))
    (and (<= a y) (<= y b) (= (f x) (some y)))))))
```

B and Atelier B

B

- Formal method for software and hardware development
- Based on ZFC + Predicate Logic
- Structured around abstract machines, variables, invariants, and operations.

Atelier B

- Suite of tools for B development
- Includes a proof obligation generator and a first-order predicate prover
- Tries to automatically discharge proof obligations

SMT-LIB

- Standard format for SMT solvers (e.g. z3, cvc5, veriT)
- Based on many-sorted first-order logic
- Comes with many theories (e.g. arrays, integer and real linear arithmetic)

Current encoding (ppTrans)

A first-order encoding of sets:

- Based on FOL with equality and uninterpreted functions (EUF)
- Sets are only *specified* through the use of an uninterpreted predicate $\in$
- Only expressions like $x \in S$ are encoded

Current encoding (ppTrans)

A first-order encoding of sets:

- Based on FOL with equality and uninterpreted functions (EUF)
- Sets are only *specified* through the use of an uninterpreted predicate $\in$
- Only expressions like $x \in S$ are encoded

Current encoding (ppTrans)

A first-order encoding of sets:

- Based on FOL with equality and uninterpreted functions (EUF)
- Sets are only *specified* through the use of an uninterpreted predicate $\in$
- Only expressions like $x \in S$ are encoded

Current encoding (ppTrans)

A first-order encoding of sets:

- Based on FOL with equality and uninterpreted functions (EUF)
- Sets are only *specified* through the use of an uninterpreted predicate $\in$
- Only expressions like $x \in S$ are encoded

Current encoding (ppTrans)

A first-order encoding of sets:

- Based on FOL with equality and uninterpreted functions (EUF)
- Sets are only *specified* through the use of an uninterpreted predicate $\in$
- Only expressions like $x \in S$ are encoded

```
SETS
  S = {e1, e2, e3}
```

Current encoding (ppTrans)

A first-order encoding of sets:

- Based on FOL with equality and uninterpreted functions (EUF)
- Sets are only *specified* through the use of an uninterpreted predicate $\in$
- Only expressions like $x \in S$ are encoded

```
SETS
  S = {e1, e2, e3}
```

$\rightarrow$

```
(declare-fun S () (P Int))
(declare-fun e1 () Int)
(declare-fun e2 () Int)
(declare-fun e3 () Int)
(declare-fun ∈0 ((Int) (P Int)) Bool)

(assert (forall ((x Int)) (=
  (∈0 x S)
  (or (= x e1) (= x e2) (= x e3)))))
```

Suggested encoding

A higher-order encoding of sets:

- Uses some extensions of SMT-LIB 2.6 to HOL brought by cvc5 (λ-abstraction, arrow type constructor)
- Sets are represented by their characteristic predicate: no need for membership predicate!

Suggested encoding

A higher-order encoding of sets:

- Uses some extensions of SMT-LIB 2.6 to HOL brought by cvc5 (λ-abstraction, arrow type constructor)
- Sets are represented by their characteristic predicate: no need for membership predicate!

Suggested encoding

A higher-order encoding of sets:

- Uses some extensions of SMT-LIB 2.6 to HOL brought by cvc5 (λ-abstraction, arrow type constructor)
- Sets are represented by their characteristic predicate: no need for membership predicate!

Suggested encoding

A higher-order encoding of sets:

- Uses some extensions of SMT-LIB 2.6 to HOL brought by cvc5 (λ-abstraction, arrow type constructor)
- Sets are represented by their characteristic predicate: no need for membership predicate!

```
SETS
  S = {e1, e2, e3}
```

Suggested encoding

A higher-order encoding of sets:

- Uses some extensions of SMT-LIB 2.6 to HOL brought by cvc5 (λ-abstraction, arrow type constructor)
- Sets are represented by their characteristic predicate: no need for membership predicate!

```
SETS
  S = {e1, e2, e3}
```

$\rightarrow$

```
(declare-const e1 Int)
(declare-const e2 Int)
(declare-const e3 Int)
(define-const S (-> Int Bool) (lambda ((x Int))
  (or (= x e1) (= x e2) (= x e3))))
```

Currently, functions such as $f : A \nrightarrow B$ are represented by:

- the set $\mathcal{P}(A \times B)$
- axiom stating that the relation is functional

$$\forall\, x\, y\, z.\, x \mapsto y \in f \land x \mapsto z \in f \Rightarrow y = z \quad \text{(functionality)}$$

- additional axioms accounting for properties of f (totality, partiality, injectivity...)

$$\forall\, x_1\, y_1\, x_2\, y_2.\, x_1 \mapsto y_1 \in f \land x_2 \mapsto y_2 \in f \land y_1 = y_2 \Rightarrow x_1 = x_2 \quad \text{(injectivity)}$$

$$\vdots$$

Can we find a better way to encode functions in order to preserve their properties and avoid additional overhead?

Currently, functions such as $f\colon A \nrightarrow B$ are represented by:

- the set $\mathcal{P}(A \times B)$
- axiom stating that the relation is functional

$$\forall\, x\, y\, z.\, x \mapsto y \in f \land x \mapsto z \in f \Rightarrow y = z \quad \text{(functionality)}$$

- additional axioms accounting for properties of f (totality, partiality, injectivity...)

$$\forall\, x_1\, y_1\, x_2\, y_2.\, x_1 \mapsto y_1 \in f \land x_2 \mapsto y_2 \in f \land y_1 = y_2 \Rightarrow x_1 = x_2 \quad \text{(injectivity)}$$

$$\vdots$$

Can we find a better way to encode functions in order to preserve their properties and avoid additional overhead?

Currently, functions such as $f : A \nrightarrow B$ are represented by:

- the set $\mathcal{P}(A \times B)$
- axiom stating that the relation is functional

$$\forall\, x\, y\, z \,.\, x \mapsto y \in f \land x \mapsto z \in f \Rightarrow y = z \quad \text{(functionality)}$$

- additional axioms accounting for properties of f (totality, partiality, injectivity...)

$$\forall\, x_1\, y_1\, x_2\, y_2 \,.\, x_1 \mapsto y_1 \in f \land x_2 \mapsto y_2 \in f \land y_1 = y_2 \Rightarrow x_1 = x_2 \quad \text{(injectivity)}$$

$$\vdots$$

Can we find a better way to encode functions in order to preserve their properties and avoid additional overhead?

A non-working example

Something we do not want:

Let $f \in \mathbb{Z} \nrightarrow \mathbb{Z}$. Let op be the following operation:

```
op (x, y) =
PRE
  x : INTEGER & y : INTEGER // x ∈ ℤ ∧ y ∈ ℤ
THEN
  f := f \/ {x |-> y}        // f := f ∪ {x ↦ y}
END
```

A non-working example

Something we do not want:
Let $f \in \mathbb{Z} \rightarrowtail \mathbb{Z}$. Let op be the following operation:

```
op (x, y) =
PRE
  x : INTEGER & y : INTEGER // x ∈ ℤ ∧ y ∈ ℤ
THEN
  f := f \/ {x |-> y}       // f := f ∪ {x ↦ y}
END
```

A non-working example

Something we do not want:
Let $f \in \mathbb{Z} \nrightarrow \mathbb{Z}$. Let op be the following operation:

```
op (x, y) =
PRE
  x : INTEGER & y : INTEGER // x ∈ ℤ ∧ y ∈ ℤ
THEN
  f := f \/ {x |-> y}        // f := f ∪ {x ↦ y}
END
```

Assume $f := \{0 \mapsto 1, 1 \mapsto 0\}$.

A non-working example

Something we do not want:

Let $f \in \mathbb{Z} \nrightarrow \mathbb{Z}$. Let op be the following operation:

```
op (x, y) =
PRE
  x : INTEGER & y : INTEGER // x ∈ ℤ ∧ y ∈ ℤ
THEN
  f := f \/ {x |-> y}       // f := f ∪ {x ↦ y}
END
```

Assume $f := \{0 \mapsto 1, 1 \mapsto 0\}$.

- After op$(2, 3)$, $f = \{0 \mapsto 1, 1 \mapsto 0, 2 \mapsto 3\}$

A non-working example

Something we do not want:
Let $f \in \mathbb{Z} \nrightarrow \mathbb{Z}$. Let op be the following operation:

```
op (x, y) =
PRE
  x : INTEGER & y : INTEGER // x ∈ ℤ ∧ y ∈ ℤ
THEN
  f := f \/ {x |-> y}       // f := f ∪ {x ↦ y}
END
```

Assume $f := \{0 \mapsto 1, 1 \mapsto 0\}$.

- After $\text{op}(2, 3)$, $f = \{0 \mapsto 1, 1 \mapsto 0, 2 \mapsto 3\}$ ✓ (still a function)

A non-working example

Something we do not want:
Let $f \in \mathbb{Z} \mapsto \mathbb{Z}$. Let op be the following operation:

```
op (x, y) =
PRE
  x : INTEGER & y : INTEGER // x ∈ ℤ ∧ y ∈ ℤ
THEN
  f := f \/ {x |-> y}         // f := f ∪ {x ↦ y}
END
```

Assume $f := \{0 \mapsto 1, 1 \mapsto 0\}$.

- After op$(2, 3)$, $f = \{0 \mapsto 1, 1 \mapsto 0, 2 \mapsto 3\}$ ✓ (still a function)
- After op$(0, 2)$, $f = \{0 \mapsto 1, 1 \mapsto 0, 0 \mapsto 2\}$

A non-working example

Something we do not want:
Let $f \in \mathbb{Z} \nrightarrow \mathbb{Z}$. Let op be the following operation:

```
op (x, y) =
PRE
  x : INTEGER & y : INTEGER // x ∈ ℤ ∧ y ∈ ℤ
THEN
  f := f \/ {x |-> y}        // f := f ∪ {x ↦ y}
END
```

Assume $f := \{0 \mapsto 1, 1 \mapsto 0\}$.

- After op$(2, 3)$, $f = \{0 \mapsto 1, 1 \mapsto 0, 2 \mapsto 3\}$ ✓ (still a function)
- After op$(0, 2)$, $f = \{0 \mapsto 1, 1 \mapsto 0, 0 \mapsto 2\}$ ✗ (no longer a function)

Idea of the encoding for "true" functions:

$\cdot^?$ is a (post-fixed) low-priority notation for the `Option` type.
$\uparrow\cdot$ is a (pre-fixed) high-priority notation for the set-lifting operation defined inductively as follows:

- $\uparrow(A \times B) = \uparrow A \times \uparrow B$
- $\uparrow\mathcal{P}(A) = \mathcal{P}(\uparrow A)$
- $\uparrow\{x \in A \mid P\} = \uparrow A$
- $\uparrow\texttt{Bool} = \texttt{Bool}$
- $\uparrow\_ = \mathbb{Z}$ (e.g. $\uparrow\mathbb{N} = \uparrow\{e_i\}_{i \in \mathcal{I}} = \mathbb{Z}$)

Idea of the encoding for "true" functions:

Rule: partial function

$f : A \nrightarrow B$ is encoded as:

- $f \in \uparrow A \rightarrow \uparrow B^?$
- $\forall x \in \uparrow A.\ f\ x \neq \mathtt{none} \Rightarrow x \in A$
- $\forall x \in \uparrow A.\ f\ x \neq \mathtt{none} \Rightarrow \exists y \in B.\ f\ x = \mathtt{some}\ y$

$\cdot^?$ is a (post-fixed) low-priority notation for the `Option` type.

$\uparrow\cdot$ is a (pre-fixed) high-priority notation for the set-lifting operation defined inductively as follows:

- $\uparrow(A \times B) = \uparrow A \times \uparrow B$
- $\uparrow\mathcal{P}(A) = \mathcal{P}(\uparrow A)$
- $\uparrow\{x \in A \mid P\} = \uparrow A$
- $\uparrow\mathtt{Bool} = \mathtt{Bool}$
- $\uparrow\_ = \mathbb{Z}$ (e.g. $\uparrow\mathbb{N} = \uparrow\{e_i\}_{i \in \mathcal{I}} = \mathbb{Z}$)

Idea of the encoding for "true" functions:

Rule: total function

$f : A \rightarrow B$ is encoded as:

- $f \in \uparrow A \rightarrow \uparrow B^?$
- $\forall x \in \uparrow A.\ f\ x \neq \texttt{none} \Leftrightarrow x \in A$
- $\forall x \in \uparrow A.\ f\ x \neq \texttt{none} \Rightarrow \exists y \in B.\ f\ x = \texttt{some}\ y$

$\cdot^?$ is a (post-fixed) low-priority notation for the `Option` type.
$\uparrow\cdot$ is a (pre-fixed) high-priority notation for the set-lifting operation defined inductively as follows:

- $\uparrow(A \times B) = \uparrow A \times \uparrow B$
- $\uparrow\mathcal{P}(A) = \mathcal{P}(\uparrow A)$
- $\uparrow\{x \in A \mid P\} = \uparrow A$
- $\uparrow\texttt{Bool} = \texttt{Bool}$
- $\uparrow\_ = \mathbb{Z}$ (e.g. $\uparrow\mathbb{N} = \uparrow\{e_i\}_{i \in \mathcal{I}} = \mathbb{Z}$)

Example:

$$f : \mathbb{N} \to a..b \quad \hookrightarrow$$

$$\begin{cases} f \in {\uparrow}\mathbb{N} \to {\uparrow}a..b^{?} \\ \forall\, x \in {\uparrow}\mathbb{N}.\ f\ \ x \neq \texttt{none} \Leftrightarrow x \in \mathbb{N} \\ \forall\, x \in {\uparrow}\mathbb{N}.\ f\ \ x \neq \texttt{none} \Rightarrow \exists\, y \in a..b.\ f\ \ x = \texttt{some}\ \ y \end{cases}$$

Example:

$$f : \mathbb{N} \to a..b \quad \hookrightarrow$$

$$\begin{cases} f \in \mathbb{Z} \to \mathbb{Z}^{?} \\ \forall\, x \in \mathbb{Z}.\ f\ x \neq \texttt{none} \Leftrightarrow x \in \mathbb{N} \\ \forall\, x \in \mathbb{Z}.\ f\ x \neq \texttt{none} \Rightarrow \exists\, y \in a..b.\ f\ x = \texttt{some}\ y \end{cases}$$

Example:

$$f : \mathbb{N} \to a..b \quad \hookrightarrow$$

$$\begin{cases} f \in \mathbb{Z} \to \mathbb{Z}^? \\ \forall\, x \in \mathbb{Z}.\ f\ x \neq \texttt{none} \Leftrightarrow x \in \mathbb{N} \\ \forall\, x \in \mathbb{Z}.\ f\ x \neq \texttt{none} \Rightarrow \exists\, y \in a..b.\ f\ x = \texttt{some}\ y \end{cases}$$

Example:

$$f : \mathbb{N} \to a..b \quad \hookrightarrow$$

$$\begin{cases} f \in \mathbb{Z} \to \mathbb{Z}^{?} \\ \forall x \in \mathbb{Z}.\ f\ x \neq \mathsf{none} \Leftrightarrow x \in \mathbb{N} \\ \forall x \in \mathbb{Z}.\ f\ x \neq \mathsf{none} \Rightarrow \exists y \in a..b.\ f\ x = \mathsf{some}\ y \end{cases}$$

```
(declare-const f (-> Int (Option Int)))
(assert (forall ((x Int)) (= (not (= (f x) none)) (<= 0 x))))
(assert (forall ((x Int)) (=> (not (= (f x) none))
  (exists ((y Int)) (and (<= a y) (<= y b) (= (f x) (some y)))))))
```

Example:

$$f : \mathbb{N} \to a..b \quad \hookrightarrow$$

$$\begin{cases} f \in \mathbb{Z} \to \mathbb{Z}^{?} \\ \forall\, x \in \mathbb{Z}.\ f\ \ x \neq \texttt{none} \Leftrightarrow x \in \mathbb{N} \\ \forall\, x \in \mathbb{Z}.\ f\ \ x \neq \texttt{none} \Rightarrow \exists\, y \in a..b.\ f\ \ x = \texttt{some}\ \ y \end{cases}$$

```
(declare-const f (-> Int (Option Int)))
(assert (forall ((x Int)) (= (not (= (f x) none)) (<= 0 x))))
(assert (forall ((x Int)) (=> (not (= (f x) none))
  (exists ((y Int)) (and (<= a y) (<= y b) (= (f x) (some y)))))))
```

Example:

$$f : \mathbb{N} \to a..b \quad \hookrightarrow$$

$$\begin{cases} f \in \mathbb{Z} \to \mathbb{Z}^{?} \\ \forall\, x \in \mathbb{Z}.\ f\ \ x \neq \texttt{none} \Leftrightarrow 0 \leq x \\ \forall\, x \in \mathbb{Z}.\ f\ \ x \neq \texttt{none} \Rightarrow \exists\, y \in a..b.\ f\ \ x = \texttt{some}\ \ y \end{cases}$$

```
(declare-const f (-> Int (Option Int)))
(assert (forall ((x Int)) (= (not (= (f x) none)) (<= 0 x))))
(assert (forall ((x Int)) (=> (not (= (f x) none))
  (exists ((y Int)) (and (<= a y) (<= y b) (= (f x) (some y)))))))
```

Example:

$$f : \mathbb{N} \to a..b \quad \hookrightarrow$$

$$\begin{cases} f \in \mathbb{Z} \to \mathbb{Z}^{?} \\ \forall\, x \in \mathbb{Z}.\ f\ x \neq \texttt{none} \Leftrightarrow 0 \leq x \\ \forall\, x \in \mathbb{Z}.\ f\ x \neq \texttt{none} \Rightarrow \exists\, y \in a..b.\ f\ x = \texttt{some}\ y \end{cases}$$

```
(declare-const f (-> Int (Option Int)))
(assert (forall ((x Int)) (= (not (= (f x) none)) (<= 0 x))))
(assert (forall ((x Int)) (=> (not (= (f x) none))
  (exists ((y Int)) (and (<= a y) (<= y b) (= (f x) (some y)))))))
```

Example:

$$f : \mathbb{N} \to a..b \quad \hookrightarrow$$

$$\begin{cases} f \in \mathbb{Z} \to \mathbb{Z}^? \\ \forall\, x \in \mathbb{Z}.\ f\ x \neq \texttt{none} \Leftrightarrow 0 \leq x \\ \forall\, x \in \mathbb{Z}.\ f\ x \neq \texttt{none} \Rightarrow \exists\, y \in \mathbb{Z}.\ a \leq y \wedge y \leq b \wedge\ f\ x = \texttt{some}\ y \end{cases}$$

```
(declare-const f (-> Int (Option Int)))
(assert (forall ((x Int)) (= (not (= (f x) none)) (<= 0 x))))
(assert (forall ((x Int)) (=> (not (= (f x) none))
  (exists ((y Int)) (and (<= a y) (<= y b) (= (f x) (some y)))))))
```

Conclusion

- Leverage recent extensions to fragments of HOL in SMT-LIB to encode B proof obligations
- This encoding needs to be further developed and tested (in comparison with *ppTrans*).

Conclusion

- Leverage recent extensions to fragments of HOL in SMT-LIB to encode B proof obligations
- This encoding needs to be further developed and tested (in comparison with *ppTrans*).

Conclusion

- Leverage recent extensions to fragments of HOL in SMT-LIB to encode B proof obligations
- This encoding needs to be further developed and tested (in comparison with *ppTrans*). This will be formalized / implemented in LEAN

Conclusion

- Leverage recent extensions to fragments of HOL in SMT-LIB to encode B proof obligations
- This encoding needs to be further developed and tested (in comparison with *ppTrans*). This will be formalized / implemented in LEAN

Questions?